Allgemeine Lizenzbedingungen der NCR GmbH

1. Der Lizenznehmer verpflichtet sich, die Lizenzprogramme vor unbefugtem Gebrauch zu schützen. Das/Die Lizenzprogramm(e) dürfen, soweit dies technisch möglich ist, nur in maschinenlesbarer oder gedruckter Form kopiert werden, wenn die Kopie zur Modifizierung oder als Sicherheitskopie benötigt wird. Jede Kopie muß den Vermerk tragen, daß die NCR GmbH (nachfolgend kurz NCR genannt) das Urheberrecht am Lizenzprogramm besitzt.

2. Der Lizenznehmer darf das/die Lizenzprogramm(e) modifizieren oder zur Benutzung mit anderen Programmen verbinden, wenn deren Benutzung auf dem Computersystem NCR Decision Mate V erfolgt. Die modifizierten oder verbundenen Programme unterliegen diesen Allgemeinen Lizenzbedingungen der NCR. Die Modifikationen und Programmverbindungen müssen die Programm-Nummer des Ursprungsprogrammes und den Vermerk aufweisen, daß die NCR das Urheberrecht daran besitzt.

3. Die Benutzer-Lizenz sowie die sonstigen Rechte und Pflichten aus diesem Vertragsverhältnis dürfen vom Lizenznehmer nur mit vorheriger schriftlicher Zustimmung der NCR an Dritte übertragen werden.

4. Weist der Lizenznehmer innerhalb einer Frist von 6 Monaten, gerechnet nach dem Datum seiner Programmempfangsbestätigung nach, daß das (die) Lizenzprogramm(e) fehlerhaft ist (sind), hat die NCR wahlweise das Recht, ein anderes, gleiches Programm zu liefern, den Fehler nachzubessern oder dem Lizenznehmer zuzugestehen, daß dieser vom Vertrag über das fehlerhafte Programm zurücktritt. Im letzteren Fall werden dem Lizenznehmer die bereits bezahlten Lizenzgebühren gegen Rückgabe der (des) Lizenzprogramme(s) zurückerstattet. Soweit gesetzlich zulässig, sind damit alle weitergehenden Gewährleistungs- und Ersatzansprüche ausgeschlossen.

5. Die NCR übernimmt keine Garantie dafür, daß das (die) Lizenzprogramm(e) den Vorstellungen und Anforderungen des Lizenznehmers entsprechen.

6. Die Benutzer-Lizenz ist zeitlich beschränkt auf die Dauer der Benutzung des in Ziffer 2 genannten NCR-Computersystems. Das (Die) Lizenzprogramm(e) nebst Kopien, Modifikationen oder Programmverbindungen ist (sind) sofort der NCR zurückzugeben oder zu vernichten, sobald der Lizenznehmer dieses Computersystem ständig nicht mehr benutzt.
   Der Lizenznehmer ist verpflichtet, der NCR innerhalb von 30 Tagen nach endgültiger Einstellung der Benutzung schriftlich mitzuteilen, seit wann das NCR-Computersystem nicht mehr verwendet wird und daß er entweder das (die) Lizenzprogramme(e), Kopien, Modifikationen und/oder Programmverbindungen an die NCR zurückgeben wird oder vernichtet hat.

7. Verstößt der Lizenznehmer gegen die vorstehenden Bedingungen, hat die NCR das Recht, den Vertrag über die Benutzer-Lizenz ohne Einhaltung einer Frist zu kündigen und vom Lizenznehmer die sofortige Rückgabe der (des) Lizenzprogramme(s) nebst Kopien, Modifikationen oder Programmverbindungen zu verlangen, ohne zur Erstattung bereits bezahlter Lizenzgebühren verpflichtet zu sein.

8. Für evtl. Streitigkeiten aus diesem Vertragsverhältnis ist Augsburg Gerichtsstand, es sei denn, daß ein ausschließlicher Gerichtsstand besteht.

Sehr geehrter NCR MS-DOS-Benutzer,

dieses Buch enthält die MS-DOS-Betriebssystem-Diskette und die Dokumentation zur Benutzung der Software. Das Benutzerhandbuch besteht aus einer leichtverständlichen Beschreibung der Einrichtungen und Funktionen von MS-DOS, während das Programmier-Handbuch das technische Detailwissen für den Programmierer vermittelt. Da Sie als Programmierer meist über ausreichende Englisch-Kenntnisse verfügen, wurde das Programmier-Handbuch in der Originalfassung belassen.

Damit Sie noch mehr Freude bei der Verarbeitung haben, haben wir noch eine weitere Diskette hinzugefügt. Diese Diskette enthält eine Anwender- und Demonstrationssoftware, die ebenso informativ wie unterhaltend ist. Mit ihr können Sie beispielsweise Liniendiagramme, Balkendiagramme und Kreisdiagramme erstellen. Mit ihr können Sie des weiteren auch Musik hören oder eine Verfolgungsjagd starten.

Eine kurze Erläuterung dieser einfach einzusetzenden Software steht in einem besonderen Anhang am Schluß des Benutzerhandbuches.

Wir hoffen, daß Sie an diesen "Extras" Freude haben und wünschen Ihnen viel Erfolg bei der Benutzung von MS-DOS mit Ihrem NCR DECISION MATE V.

Ihre
NCR GmbH, Augsburg

NCR

NCR DECISION MATE V

MS™-DOS
Benutzer Handbuch

MS-PRODUCT	:	MS (TM)-DOS
MS VERSION NO.	:	2.0
PRODUCT NO.	:	D006-0052-0201
NCR PART NO.	:	017-0031752 B
SERIAL NO.	:	16545

CP/M ist ein eingetragenes Warenzeichen von Digital Research.
MACRO-86, MS-CREF, MS-LINK, MS-LIB und MS-DOS (mit den Programmnamen EDLIN und DEBUG) sind Warenzeichen der Microsoft Corp.
Microsoft ist ein eingetragenes Warenzeichen der Microsoft Corp.

ORDER NUMBERS

Augsburg Stock Number 017-0031752
Software Product I.D. D006-0052-0001

2. Auflage, Oktober 1983

NCR ist ständig bemüht, die Produkte im Zuge der Entwicklung von Technologie, Bauteilen, Soft- und Firmware dem neuesten Stand anzupassen. NCR behält sich deshalb das Recht vor, Spezifikationen ohne vorherige Ankündigung zu ändern.
Nicht alle hier beschriebenen Leistungen werden von NCR in allen Teilen der Welt vertrieben. Nähere Informationen bezüglich eventueller Einschränkungen oder auch Erweiterungen sowie den aktuellen Stand erfahren Sie von Ihrem Händler oder der nächstgelegenen NCR-Geschäftsstelle.

MS-DOS BENUTZER HANDBUCH

INHALTSVERZEICHNIS

1. EINFÜHRUNG

2. STARTHILFE

3. MEHR ÜBER DATEIEN

4. INFORMATION ÜBER BEFEHLE

5. MS-DOS BEFEHLE

6. MS-DOS EDITIER- UND FUNKTIONSTASTEN

7. DER ZEILEN-EDITOR (EDLIN)

8. DAS DIENSTPROGRAMM FÜR DEN DATEIVERGLEICH (FC)

9. LINKPROGRAMM (MS-LINK)

EINFÜHRUNG

MS™ -DOS ist ein Plattenbetriebssystem für den 16-Bit-Prozessor des NCR DECISION MATE V. Über MS-DOS wird mit dem Computer, mit Diskettenlaufwerken und dem Drucker (sofern vorhanden) kommuniziert, wobei diese Betriebsmittel so vorteilhaft wie möglich für den Benutzer verwaltet werden.

WAS IST EIN BETRIEBSSYSTEM?

Ein Betriebssystem ist der "stille Teilhaber" des Benutzers beim Einsatz des Computers. Es ist die Schnittstelle zwischen der Hardware und dem Benutzer einerseits und weiterer Software (Anwenderprogramme und Benutzerprogramme) andererseits. Ein Betriebssystem kann mit der Elektrizität in einem Haushalt verglichen werden: sie wird für die meisten Haushaltsgeräte benötigt, man ist sich ihrer jedoch nicht immer bewußt.

Betriebssysteme bieten verschiedene Möglichkeiten. Mit MS-DOS können Dateien erstellt und verwaltet werden, Programme ausgeführt und gebunden werden. Außerdem kann auf Peripheriegeräte (beispielsweise Drucker und Plattenlaufwerke) zugegriffen werden, die an den Computer angeschlossen sind.

BENUTZUNG DIESES HANDBUCHS

In diesem Handbuch wird MS-DOS und die Benutzung von MS-DOS beschrieben. In dem vorliegenden Kapitel werden einige der grundlegenden MS-DOS-Konzepte dargelegt. In Kapitel 2 wird besprochen, wie MS-DOS gestartet und die Disketten formatiert und gesichert werden.

Kapitel 3 enthält Informationen über die Dateien. In den Kapiteln 4 bis 6 werden die MS-DOS -Befehle erläutert.

In Kapitel 7 wird der Zeilen-Editor (EDLIN) beschrieben. Diese Kapitel sollten sorgfältig durchgelesen werden — sie enthalten Informationen über den Schutz der Daten über die Systembefehle und die MS-DOS-Editierbefehle.

In Kapitel 8 wird das Dienstprogramm für den Dateivergleich (FC) beschrieben. Dieses Dienstprogramm wird für den Vergleich des Inhalts von zwei Quellen- oder Binärdateien benutzt.

Schreibt der Benutzer Programme und möchte er separat erstellte Objektmoduln binden und verschiebliche Moduln erstellen, so wird in Kapitel 9 ein entsprechendes MS-DOS-Dienstprogramm, MS-LINK, beschrieben.

Die Anhänge zu diesem Handbuch enthalten Instruktionen zur Benutzung von MS-DOS und anderer Betriebssysteme bei dem NCR DECISION MATE V, Diskettenfehlermeldungen und besondere Richtlinen für die Ausführung von MS-DOS kompatiblen Anwendungen auf dem Computer.

Werden weitere Informationen benötigt, so enthält ein Begleithandbuch, das *PROGRAMMIERERHANDBUCH*, Informationen über die technischen Aspekte von MS-DOS. In diesem Handbuch werden auch die Systemarchitektur von MS-DOS, zusätzliche Dienstprogramme, sowie Systemaufrufe und Unterbrechungen beschrieben.

SYNTAX BESCHREIBUNG

Die folgende Syntaxbeschreibung wird in diesem ganzen Handbuch bei der Beschreibung von Befehlen und in der Anweisungssyntax benutzt. Die Liste mag auf den ersten Blick verwirrend erscheinen. Nachdem die Befehle jedoch einige Male benutzt wurden, ist man mit der Schreibweise sehr schnell vertraut.

[ ] Eckige Klammern
Geben an, daß der in den eckigen Klammern stehende Eintrag wahlweise ist.

< > Spitze Klammern
Geben an, daß der Benutzer den Text für diesen Eintrag liefert. Steht in den spitzen Klammern in den Beispielen ein Text in Kleinbuchstaben, so muß ein dem Text entsprechender Eintrag eingegeben werden: zum Beispiel <Dateiname>.

{ } Geschweifte Klammern
Geben an, daß der Benutzer zwischen zwei oder mehr Einträgen wählen kann. Mindestens einer der in den geschweiften Klammern stehenden Einträge muß gewählt werden, es sei denn die Einträge selbst stehen in eckigen Klammern.

. . . Auslassungszeichen
Geben an, daß ein Eintrag beliebig oft wiederholt werden kann.

| Senkrechter Trennstrich
Wird der Trennstrich mit einem MS-DOS-Filter benutzt, so gibt er eine Datenübergabe an. (Diese Einrichtung wird im einzelnen in Kapitel 4, Informationen über Befehle, beschrieben.)

CAPS Großbuchstaben
Geben Teile von Anweisungen oder Befehlen an, die genau wie dargestellt eingegeben werden müssen. Großbuchstaben geben auch bestimmte Tasten an, wie beispielsweise <CR>.

Sämtliche anderen Satzzeichen, wie beispielsweise Komma, Doppelpunkt, Schrägstriche und Gleichheitszeichen müssen genau wie dargestellt eingegeben werden.

BETRACHTUNGEN IM ZUSAMMENHANG MIT DISKETTEN- UND FESTPLATTENSYSTEMEN

Um die Erläuterungen in diesem Handbuch zu vereinfachen beruhen die Beispiele auf einem Diskettensystem mit mehreren Laufwerken. Verfügt der Benutzer jedoch über ein Disketten-/Festplattensystem, so benutzt er nur das Diskettenlaufwerk zur Formatierung und Anfertigung von Kopien von Disketten. In diesen Fällen fordert MS-DOS den Benutzer immer auf, die Disketten auszuwechseln und wartet, bis der Benutzer die neue Diskette eingelegt hat. Danach wird die Verarbeitung durch Betätigung einer beliebigen Taste fortgesetzt.

Das nächste Kapitel "STARTHILFE" beschreibt das Starten des MS-DOS-Betriebssystems und das Formatieren und Sichern von Disketten.

STARTHILFE

INBETRIEBNAHME

Die mit diesem Handbuch gelieferte MS-DOS-Master-Diskette enthält die gesamten Dateien der Betriebssystem-Software und sämtliche Befehle. In diesem Kapitel wird die Installation der Software, die Umschaltung von einer Diskette zu einer anderen, das Schützen der Master-Diskette und das Formatieren anderer neuer Disketten beschrieben. Abschließend sind Informationen über die Dateien enthalten.

Bevor jedoch mit dem Laden der Software begonnen wird, sollte der Benutzer etwas mehr über den NCR DECISION MATE V und die so wichtigen Disketten erfahren. Je nach Computermodell verfügt der Benutzer entweder über ein Diskettensystem oder über ein Disketten-/Festplattensystem. Die Plattentypen spielen bei MS-DOS keine Rolle. Für die Software ist nur wichtig, wo die Informationen stehen und wo sie gespeichert werden sollen.

Unabhängig von dem benutzten Plattensystem muß die Verarbeitung immer mit dem Diskettenlaufwerk A gestartet werden. Dies soll nun mit dem Laden von MS-DOS in den Speicher verdeutlicht werden.

LADEN VON MS-DOS

Den Computer einschalten, die MS-DOS-Diskette in Laufwerk A einlegen und die ↵ Taste betätigen. (Diese Abschlußtaste wird auch als die <CR>-Taste bezeichnet.) Hier handelt es sich immer um das Standardverfahren für die Inbetriebnahme. Je nach Speichergröße kann das Laden von MS-DOS bis zu 25 Sekunden dauern.

Nachdem MS-DOS geladen ist, sucht das System die MS-DOS-Diskette nach der COMMAND.COM-Datei ab und lädt sie in den Speicher. Die COMMAND.COM-Datei ist ein Programm, das die vom Benutzer eingegebenen Befehle verarbeitet und dann die entsprechenden Programme ausführt. Sie wird auch als der "Befehlsprozessor" bezeichnet.

Nachdem der Befehlsprozessor geladen ist, wird ein Copyright-Vermerk und eine Meldung zur Kennzeichnung der Software auf dem Bildschirm angezeigt. Gleichzeitig wird auch die Bereitschaftsmeldung auf dem Bildschirm angezeigt. Nun kann mit der Benutzung von MS-DOS begonnen werden.

KOPIEREN DER MASTER-DISKETTE

Als erstes wird eine Sicherheitskopie der Master-Diskette angefertigt. Die Software auf der MS-DOS-Master-Diskette leitet dieses Verfahren automatisch ein. Der Benutzer braucht nur einige einfache Fragen zu beantworten oder einige einfache Anweisungen zu befolgen, die von MS-DOS angezeigt werden.

Die Bildschirmanzeigen erläutern sich von selbst, so daß der Benutzer anhand des ausgedruckten Textes arbeiten kann. Eine Zusammenfassung des Kopierverfahrens wird in der nachstehenden Tabelle dargestellt, wobei die Benutzeraktionen fettgedruckt sind.

Als erstes wird der Benutzer gefragt, über wieviele Diskettenlaufwerke er verfügt. Die Frage muß beantwortet werden. Danach wird zu:

- Seite 2-3 gegangen, wenn der Benutzer über 2 Diskettenlaufwerke verfügt,
- und zu Seite 2-4, wenn er über 1 Diskettenlaufwerk verfügt.

Kopieren der Master-Diskette
Zusammenfassung des Arbeitsablaufes

ZWEI DISKETTENLAUFWERKE

Formatierung beginnt . . .

A> FORMAT B:

Neue Diskette in Laufwerk B einlegen —

eine beliebige Taste betätigen!

xxxxxx bytes total disk space
xxxxxx bytes available on disk

Format another (Y/N)? N

Jetzt keine weiteren Disketten formatieren. N (nein) betätigen.

Formatierung abgeschlossen.

Nun kann die Master-Diskette in Laufwerk A auf die Diskette in Laufwerk B kopiert werden. Nach Aufforderung beliebige Taste betätigen.

A> DISKCOPY A: B:/V

Die Disketten befinden sich schon in den richtigen Laufwerken. Beliebige Taste betätigen!

Copying . . .

Copying . . . Kopieren abgeschlossen.

Copy another (Y/N)? N

Jetzt keine weiteren Disketten kopieren. N (nein) betätigen.

Master-Diskette aus Laufwerk A entnehmen und aufbewahren. Neue Master-Diskette aus Laufwerk B entnehmen und in Laufwerk A einlegen.

Kopieren der Master-Diskette
Zusammenfassung des Arbeitsablaufes

EIN DISKETTENLAUFWERK

Formatierung beginnt . . .

A> FORMAT A:

Neue Diskette in Laufwerk A einlegen und beliebige Taste drücken.

Die Master-Diskette herausnehmen und eine neue Diskette einlegen; beliebige Taste betätigen, wenn bereit.

xxxxxx bytes total disk space
xxxxxx bytes available on disk

Format another (Y/N)? N

Jetzt keine weiteren Disketten formatieren. N (nein) betätigen.

Formatierung abgeschlossen.

Die formatierte Diskette aus Laufwerk A herausnehmen und die Master-Diskette wieder in Laufwerk A einlegen.

Während des darauffolgenden Kopiervorgangs wechseln Sie die Disketten wie angegeben, bis die "Copy complete"-Meldung erscheint.

A> DISKCOPY/V

"Kopie"-Diskette in Laufwerk A einlegen und beliebige Taste drücken.

Disketten auswechseln; beliebige Taste betätigen, wenn bereit

Kopiervorgang abgeschlossen.

Copy another (Y/N)? N

Jetzt keine weiteren Disketten kopieren. N (nein) betätigen.

LAUFWERKBEZEICHNUNGEN

Beim Kopieren der Master-Diskette wurde der Benutzer angewiesen, eine Diskette einzulegen oder auszuwechseln. Dabei wurde die Laufwerkbezeichnung angegeben. Zum Beispiel "Insert a disk into drive A." (Eine Diskette in Laufwerk A einlegen.)

Anhand der Laufwerkbezeichnung, bei der es sich immer um einen Buchstaben handelt, weiß MS-DOS, wo die Informationen stehen und wo sie gespeichert werden müssen. Unabhängig von den benutzten Platteneinheiten verfügt jedes Laufwerk immer über seine eigene Bezeichnung. Hier wird auf die folgenden Beispiele für die Plattenkonfiguration verwiesen:

- Der Benutzer verfügt über zwei Diskettenlaufwerke. Das eine trägt die Bezeichnung (und das Kennzeichen) A. Das andere Laufwerk trägt die Bezeichnung (und das Kennzeichen) B.
- Der Benutzer verfügt über ein Diskettenlaufwerk und über eine Festplatteneinheit. Das Diskettenlaufwerk trägt die Bezeichnung (und das Kennzeichen) A. Was ist jedoch mit der Festplatte? Eine Festplatteneinheit enthält zwei logische Plattenlaufwerke. (Sie sind zwar nicht sichtbar und nicht gekennzeichnet, sind jedoch vorhanden.) In dieser Beispielkonfiguration haben die logischen Plattenlaufwerke der Festplatteneinheit die Bezeichnungen B und C.
- Nun soll davon ausgegangen werden, daß der Benutzer über 2 Diskettenlaufwerke und 3 Festplatteneinheiten verfügt. Wie lauten nun die Laufwerkbezeichnungen? Die Diskettenlaufwerke haben die Bezeichnung A und B. Die Laufwerke der Festplatteneinheit heißen C, D, E, F, G, und H.

Die Laufwerkbezeichnungen werden von MS-DOS zugeordnet. MS-DOS geht davon aus, daß der Benutzer über zwei Diskettenlaufwerke verfügt. Ist dies nicht der Fall, so muß der Benutzer seine Konfiguration für MS-DOS beschreiben.

DEFINITION DER FESTPLATTEN

Verfügt der Benutzer über Festplatten, so muß das Inbetriebnahmeverfahren an dieser Stelle mit der Definition der Plattenkonfiguration fortgesetzt werden. (Verfügt der Benutzer nur über ein Diskettensystem, so kann dieser Abschnitt übergangen werden.)

Die Plattenkonfiguration wird mit der CONFIG-Routine definiert. Mit den sich selbst erklärenden Bildschirmanzeigen ist diese Routine einfach zu benutzen. Es brauchen nur der Name der Routine eingegeben und die Bildschirmanzeigen befolgt zu werden,

wobei eine Art Konversation mit MS-DOS geführt wird.

Um das Verfahren noch einfacher zu gestalten, wird jedoch in Tabelle 1 das gesamte Konfigurationsverfahren zusammengefaßt.

HINWEIS: Bei dem beschriebenen Verfahren wird von einem Diskettenlaufwerk und einer Festplatteneinheit ausgegangen. Die Eingaben müssen der jeweiligen Konfiguration des Benutzers angepaßt werden.

Anzeige/Eingabe	Kommentare
A>	Der Benutzer setzt ein Disketten/Festplattensystem ein, MS-DOS geht jedoch noch immer davon aus, daß der Benutzer nur über Disketten verfügt. Mit dem CONFIG-Dienstprogramm muß die Software nun über die Festplatten informiert werden.
A> CONFIG ↵	Der Benutzer gibt CONFIG ein und betätigt ↵ .

CONFIG UTILITY	Der Benutzer sollte sich dieses Hauptmenü kurz ansehen. Diese Funktionen können alle mit CONFIG ausgeführt werden. (Die Funktionen werden im Einzelnen in dem Kapitel "MS-DOS-Befehle" erläutert.)
1) Modify Function Keys 2) Select Printer (Serial/Parallel) 3) Modify Retry/Restore Counter 4) Modify Serial Printer Interface 5) Modify Disk Configuration 6) Exit Program	
* Enter Function 5	5 eingeben.

CONFIG UTILITY	Eine weitere Bildschirmanzeige! Sie werden diese Bildschirmanzeige für die Plattenkonfiguration mehrmals benutzen. Zuerst muß die Anzahl der Diskettenlaufwerke geändert werden.
Modify Disk Configuration	
1) Modify number of flexible disks 2) Modify number of hard disks 3) Display configuration 4) Return to main menu	In diesem Fall wird diese Funktion gewünscht.
* Enter Function 1	1 eingeben.

Tabelle 1 Definition der Plattenkonfiguration

Anzeige/Eingabe	Kommentare
CONFIG UTILITY Modify Disk Configuration Modify Number of Flexible Disks 1) One Flex Disk 2) Two Flex Disks 3) Return to Main Program * Enter Function 1	 Die Anzahl der dem Benutzer zu Verfügung stehenden Disketten muß hier angegeben werden. In diesem Fall wird diese Funktion gewünscht. 1 eingeben.
– – – – – – – – – – – – CONFIG UTILITY Modify Disk Configuration 1) Modify number of flexible disks 2) Modify number of hard disks 3) Display configuration 4) Return to main menu * Enter Function 2	 Nun wird die Anzahl der Festplatten geändert. In diesem Fall wird diese Funktion gewünscht. 2 eingeben.

cont.

Anzeige/Eingabe	Kommentare
CONFIG UTILITY Modify Disk Configuration Modify Number of Hard Disks 1) No hard disk 2) One hard disk 3) Two hard disks 4) Three hard disks 5) Return to Main Program * Enter Function 2	Hier erhält die Software Informationen über das Festplattenlaufwerk. Diese Funktion wird von Ihnen benötigt, wenn von einer Festplatteneinheit ausgegangen wird. 2 eingeben.
– – – – – – – – – – – – CONFIG UTILITY Modify Disk Configuration 1) Modify number of flexible disks 2) Modify number of hard disks 3) Display configuration 4) Return to main menu * Enter Function 4	Nun erscheint die Bildschirmanzeige für die Plattenkonfiguration . Funktion 4 auswählen. (Möchte der Benutzer die Laufwerk-Zuordnungen sehen, so wählt er Funktion 3. ↵ betätigen, um zu dieser Bildschirmanzeige zurückzukehren.) 4 eingeben.

cont.

Anzeige/Eingabe	Kommentare
CONFIG UTILITY 1) Modify Function Keys 2) Select Printer (Serial/Parallel) 3) Modify Retry/Restores Counter 4) Modify Serial Printer Interface 5) Modify Disk Configuration 6) Exit Program	Das Konfigurieren ist beendet. In diesem Fall wird diese Funktion gewünscht.
* Enter Function 6	6 eingeben.
- - - - - - - - - - - -	
1 Update O.S. disk in drive A 2 Return to main program 3 Exit CONFIG	Die Funktion für das Verlassen des Programms ist wichtig. An dieser Stelle werden die Änderungen permanent gemacht, indem sie auf Platte geschrieben werden.
ATTENTION : Changes to the disk configuration must be written to disk (permanent) and must be followed by a re-start. Update the disk, exit CONFIG, and then turn off and on the computer when the system prompt appears.	Den "Attention"-Hinweis lesen und die nachstehende Folge ausführen:
* Enter Function 1	1 eingeben. (Es kann "gehört" werden, wie die Änderungen auf die Platte geschrieben werden.)
* Enter Function 3	3 eingeben. (Das CONFIG-Dienstprogramm wird verlassen.)
A>	Der Computer muß nun aus- und wieder eingeschaltet werden. Das Inbetriebnahmeverfahren muß wie in dem nachfolgenden Text beschrieben beendet werden.

FORMATIEREN DER PLATTEN

Bis jetzt haben Sie mit DECISION MATE V und MS-DOS schon eine ganze Reihe von Aufgaben durchgeführt. So wurde die Software durch Anfertigung einer Kopie der Master-Diskette geschützt. Für eine Festplatte wurde die Konfiguration definiert. Außerdem hat der Benutzer gesehen, wie eine Diskette formatiert wird. An dieser Stelle sei jedoch nochmals daraufhingewiesen, daß jede neue Platte formatiert werden muß.

Die Festplatten sind beispielsweise noch nicht für MS-DOS formatiert. Die FORMAT-Routine wird im einzelnen in Kapitel 5 beschrieben, so daß die Beschreibung hier nicht wiederholt werden soll. Dennoch sind einige Hinweise über das Formatieren von Festplatten zu beachten.

- Das Formatieren eines logischen Laufwerks einer Festplatteneinheit dauert cirka 20 Minuten. Diese Zeitrechnung beruht auf der "Standard"-Anzahl von fünf Überprüfungen (Lese- und Schreibprüfungen). Die Formel lautet 4 Minuten + (3 Minuten x # Prüfungen). Der Benutzer kann die Anzahl von Überprüfungen vor Beginn der Formatierung erhöhen oder vermindern.
- Vor dem Formatieren einer Festplatte muß der Benutzer immer prüfen, ob das logische Plattenlaufwerk noch nicht von MS-DOS oder von einem anderen Betriebssystem formatiert wurde. Mit einem Befehl wie dem MS-DOS-Befehl CHKDSK muß zuerst der Inhalt der Platte überprüft werden.

DEFINITION EINES SERIELLEN DRUCKERS

Wenn der Benutzer einen Drucker einsetzt, geht MS-DOS davon aus, daß es sich um einen Paralleldrucker handelt. Wird ein serieller Drucker benutzt, so muß er mit dem CONFIG-Dienstprogram für MS-DOS definiert werden. Aufgrund der verschiedenartigen Druckeranforderungen wird auch hier wieder auf Kapitel 5 für eine genaue Beschreibung von CONFIG verwiesen.

Hiermit ist nun das Inbetriebnahmeverfahren beendet. In den nächsten Abschnitten dieses Kapitels werden die Betriebsverfahren und Dateien etwas näher beleuchtet.

EINGABE VON DATUM UND ZEIT

Wird MS-DOS in den Speicher geladen oder der Computer neu gestartet, so werden die Eingabeaufforderungen für Datum und Zeit auf dem Bildschirm angezeigt. Der Benutzer sollte diese Information unbedingt eingeben. Mit ihr kann dann auf einfache Weise ermittelt werden, wann Daten auf der Diskette erstellt oder aktualisiert wurden.

Wird die Eingabeaufforderung:

Enter new date:_ (Neues Datum eingeben:_)

auf dem Bildschirm angezeigt, so muß das aktuelle Datum in dem Format mm-dd-yy eingegeben werden.

- mm ist eine 1- oder 2-stellige Zahl von 1-12 (Monat)
- dd ist eine 1- oder 2-stellige Zahl von 1-31 (Tag des Monats)
- yy ist eine 2-stellige Zahl von 80-99 (wobei 19 angenommen wird), oder eine 4-stellige Zahl von 1980-2099 (Jahr)

Jedes beliebige Datum ist als Antwort auf die Eingabeaufforderung für das neue Datum zulässig, solange es im obigen Format eingegeben wird. Die Trennzeichen zwischen den Zahlen können Bindestriche (-) oder Schrägstriche (/) sein. Zum Beispiel:

5-1-83 oder 05/01/83

In beiden Fällen handelt es sich um zulässige Antworten auf die Eingabeaufforderung für das neue Datum.

Wird ein ungültiges Datum oder eine ungültige Form des Datums eingegeben, so gibt das System die Eingabeaufforderung für das neue Datum erneut aus.

Nachdem die Eingabeaufforderung für das neue Datum beantwortet und die Antwort durch Betätigung der <CR>-Taste eingegeben wurde, wird folgende Eingabeaufforderung auf dem Bildschirm angezeigt:

Current time is 0.00:00.00
Enter new time:_

(Gegenwärtige Zeit ist 0.00:00.00
Neue Zeit eingeben:_)

Die gegenwärtige Zeit wird in dem Format hh:mm eingegeben.

- hh ist eine 1- oder 2-stellige Zahl von 0-23 (Stunden)
- mm ist eine 1- oder 2-stellige Zahl von 0-59 (Minuten)

Mit diesem Zeitwert ermittelt MS-DOS, wann die Dateien in dem System zuletzt aktualisiert und/oder erstellt wurden. Hier muß noch beachtet werden, daß MS-DOS Militärzeit benutzt. 1:30 mittags wird 13:30 geschrieben.

Beispiel:

Current time is: 0:00:00.00
Enter new time: 9:05
(Gegenwärtige Zeit ist 0:00:00.00
Neue Zeit eingeben: 9:05)

Für die Trennung von Stunden und Minuten darf nur der Doppelpunkt (:) benutzt werden. Wird ein ungültiges Trennzeichen eingegeben, so wiederholt MS-DOS die Eingabeaufforderung.

HINWEIS: Wird während der Eingabe ein Fehler gemacht, so wird die Kontrolltaste auf der Tastatur betätigt und gedrückt gehalten und danach die C-Taste betätigt. Mit der <CONTROL-C>-Funktion wird die aktuelle Eingabe abgebrochen. Die Eingabeaufforderung kann dann neu beantwortet oder ein anderer Befehl kann eingegeben werden. Mit der Rücktaste kann eine Zeile korrigiert werden, bevor <CR> betätigit wird. Mit der Rücktaste kann jeweils ein Buchstabe gelöscht werden.

ÄNDERUNG DES STANDARDLAUFWERKS

Das A> ist die MS-DOS-Systemmeldung vom Befehlsprozessor. Mit ihr wird dem Benutzer mitgeteilt, daß MS-DOS für die Aufnahme von Befehlen bereit ist.

Das A in der vorhergehenden Systemmeldung stellt das Standardplattenlaufwerk dar. Dies bedeutet daß MS-DOS nur die Platte in Laufwerk A nach den vom Benutzer eingegeben Dateinamen absucht, und Dateien nur auf diese Platte schreibt, es sei denn, der Benutzer gibt ausdrücklich ein anderes Laufwerk an. MS-DOS kann aufgefordert werden, eine Platte in einem anderen Laufwerk

abzusuchen. In diesem Fall wird die Laufwerkbezeichnung geändert oder ein anderes Laufwerk in einem Befehl angegeben. Zur Änderung der Laufwerkbezeichnung wird der neue Laufwerkbuchstabe gefolgt von einem Doppelpunkt eingegeben. Zum Beispiel:

```
A>
A>B: <CR>   (als Antwort auf die Systemmeldung wurde B:
             eingegeben)
B>
```

Die Systemmeldung B> wird auf dem Bildschirm angezeigt und Laufwerk B ist nun das Standardlaufwerk. MS-DOS sucht nur die Platte in Laufwerk B ab, bis der Benutzer wieder ein anderes Standardlaufwerk angibt. Um wieder zu Laufwerk A zurückzukehren, wird einfach A: angegeben (der Doppelpunkt darf nicht vergessen werden.)

```
A>B:
B>A: <CR>
A>
```

SICHERUNG DER DISKETTEN

Von der Master-Diskette wurde eine Sicherheitskopie angefertigt. Es sollten jedoch von allen Disketten Sicherheitskopien angefertigt werden. Wird eine Diskette beschädigt oder werden Dateien versehentlich gelöscht, so hat der Benutzer die Informationen immer noch auf seiner Sicherheitsdiskette.

Die Sicherheitskopien von Disketten werden mit dem DISKCOPY-Befehl angefertigt. Die Sicherheitskopien der Festplatten werden mit dem BACKUP-Befehl angefertigt. (Beide Befehle werden im einzelnen in Kapitel 5, MS-DOS-Befehle beschrieben.)

BENUTZUNG DER PROGRAMMIERBAREN FUNKTIONSTASTEN

Die Tastatur verfügt über eine Reihe mit Sondertasten. Diese Tasten sind mit F1 bis F20 markiert. Sie befinden sich in der obersten Reihe auf der Tastatur. Bei diesen Tasten handelt es sich um Sondertasten, da sie für jede vom Benutzer gewünschte Funktion definiert (programmiert) werden können.

Wie die Einrichtung der automatischen Programmausführung (siehe nächster Abschnitt) sind die programmierbaren Funktions-

tasten, insbesondere für die Ausführung von häufig benutzten oder schwierigen Funktionen, besonders bequem. So möchte der Benutzer beispielsweise den Inhalt einer Diskette überprüfen, bevor er auf sie zugreift. Zu diesem Zweck könnte er den Befehl für die Anzeige des Inhaltsverzeichnises (DIR) einer Funktionstaste zuordnen. Möchte er den Befehl benutzen, so betätigt er einfach die Taste und braucht den Befehl nicht mehr über die Tastatur einzugeben.

Die Funktionstasten werden zusammen mit dem CONFIG-Dienstprogramm von MS-DOS definiert. (Siehe Kapitel 5.)

AUTOMATISCHE PROGRAMMAUSFÜHRUNG

Soll ein bestimmtes Programm automatisch bei jedem Starten von MS-DOS ausgeführt werden, so kann dies mit der Automatischen Programmausführung geschehen. Dies gilt beispielsweise für den Fall, in dem der Benutzer möchte, daß MS-DOS die Namen der Dateien jedesmal angezeigt, wenn MS-DOS geladen wird.

Beim Starten von MS-DOS sucht der Befehlsprozessor nach einer Datei namens AUTOEXEC.BAT auf der MS-DOS-Diskette. Bei dieser Datei handelt es sich um ein Programm, das MS-DOS jedesmal ausführt, wenn es gestartet wird.

In Kapitel 4, Informationen über Befehle, wird beschrieben, wie eine AUTOEXEC.BAT-Datei erstellt wird.

AUSSCHALTEN DES SYSTEMS

In MS-DOS gibt es keinen "Abmelde"-Befehl. Zur Beendigung der Terminal-Sitzung werden einfach die Laufwerk-Türen geöffnet und die Disketten herausgenommen. Als Antwort auf die Systemmeldung in Form des Standardlaufwerkbuchstabens wird das Terminal einfach ausgeschaltet.

DATEIEN

Eine Datei ist eine Sammlung von zusammengehörigen Informationen. Eine Datei auf der Diskette kann mit einem Aktenordner in einem Schreibtisch verglichen werden. So kann ein Aktenordner beispielsweise die Namen und Adressen der in dem Büro beschäftigten Mitarbeiter enthalten. Diese Akte könnte als Haupt-Angestelltenakte bezeichnet werden. Eine Datei auf der Diskette könnte ebenfalls die Namen und Adressen der in dem

Büro beschäftigten Angestellten enthalten. Sie könnte als Haupt-Angestelltendatei bezeichnet werden.

Sämtliche Programme, Texte und Daten auf der Diskette stehen in Dateien. Jede Datei verfügt über einen einmaligen Namen. Auf die Dateien wird mit ihrem Namen bezug genommen. In Kapitel 3, "Mehr Über Dateien", wird beschrieben, wie die Dateien benannt werden.

Eine Datei wird immer dann erstellt, wenn Daten oder Texte am Terminal eingegeben und gesichert werden. Dateien werden auch erstellt, wenn Programme geschrieben und benannt und auf den Disketten gesichert werden.

VERWALTUNG DER DATEIEN DURCH MS-DOS

Die Namen der Dateien stehen in Inhaltsverzeichnissen auf einer Diskette. Diese Inhaltsverzeichnisse enthalten auch Informationen über die Größe der Dateien, über ihre Adresse auf der Diskette und über das Erstellungs- und Aktualisierungsdatum. Das Inhaltsverzeichnis, mit dem der Benutzer arbeitet, wird als das gegenwärtige Verzeichnis oder Arbeitsverzeichnis bezeichnet.

Ein zusätzlicher Systembereich wird als Dateizuordnungstabelle bezeichnet. In dieser Tabelle werden die Adressen der Dateien auf der Diskette verwaltet. Sie weist auch den freien Platz auf den Disketten zu, sodaß neue Dateien erstellt werden können.

Mit diesen beiden Systembereichen, dem Inhaltsverzeichnis und der Dateizuordnungstabelle, kann MS-DOS die Dateien auf den Disketten erkennen und organisieren. Die Dateizuordnungstabelle wird auf eine neue Diskette kopiert, wenn sie mit dem MS-DOS-Befehl FORMAT formatiert wird. Außerdem wird ein leeres Inhaltsverzeichnis erstellt, das als Stammverzeichnis bezeichnet wird.

DIR-BEFEHL (INHALTSVERZEICHNIS ANZEIGEN)

Mit dem DIR-Befehl kann ermittelt werden, welche Dateien auf der Diskette stehen. Mit diesem Befehl wird MS-DOS angewiesen, sämtliche Dateien in dem gegenwärtigen Verzeichnis auf der angegebenen Diskette anzuzeigen. Ist die MS-DOS-Diskette in Laufwerk A eingelegt, und möchte der Benutzer eine Auflistung des gegenwärtigen Verzeichnisses auf dieser Platte, so gibt er beispielsweise:

DIR A: <CR>

ein.

MS-DOS antwortet mit einer Verzeichnisauflistung sämtlicher Dateien in dem gegenwärtigen Verzeichnis auf der MS-DOS-Diskette. Um die Bildschirmanzeige zur Überprüfung der Dateien zu stoppen, wird die Kontrolltaste betätigt und gedrückt gehalten. Danach wird die S-Taste betätigt. Um die Bildschirmanzeige wieder aufzunehmen, wird eine beliebige Taste betätigt.

HINWEIS: Zwei MS-DOS-Systemdateien, IO.SYS und MSDOS.SYS sind "versteckte" Dateien. Sie werden bei Ausgabe des DIR-Befehls nicht auf dem Bildschirm angezeigt.

Durch Eingabe von DIR und eines Dateinamens können auch Informationen über jede Datei auf der Diskette abgefragt werden. Hat der Benutzer beispielsweise eine Datei namens MYFILE.TXT erstellt, so erhält er durch den Befehl:

DIR MYFILE.TXT <CR>

eine Anzeige sämtlicher Verzeichnisinformationen (Dateiname, Dateigröße, Datum der letzten Aufbereitung) für die Datei MYFILE.TXT.

Für weitere Informationen über den DIR-Befehl sei auf Kapitel 5, MS-DOS-Befehle verwiesen.

PRÜFEN DER DISKETTEN

Mit dem MS-DOS-Befehl CHKDSK werden die Disketten auf Konsistenz und Fehler überprüft, genau wie eine Sekretärin einen Brief noch einmal liest. CHKDSK analysiert die Inhaltsverzeichnisse und die Dateizuordnungstabelle auf der vom Benutzer angegebenen Diskette. Dann erstellt er eine Statusliste sämtlicher Unstimmigkeiten, wie beispielsweise der Dateien, für die in dem Inhaltsverzeichnis eine von Null abweichende Größe angeben wird, die jedoch tatsächlich überhaupt keine Daten enthalten.

Zur Überprüfung der Diskette in Laufwerk A wird:

CHKDSK A: <CR>

eingegeben.

MS-DOS zeigt eine Statusliste und eventuell gefundene Fehler an. Ein Beispiel für diese Anzeige und weitere Informationen über CHKDSK stehen in der Beschreibung des CHKDSK-Befehls in Kapitel 5. CHKDSK sollte gelegentlich für jede Diskette ausgeführt werden, um die Integrität der Dateien zu gewährleisten.

ZUSAMMENFASSUNG DES KAPITELS

- Die Verarbeitung beginnt immer in Plattenlaufwerk A. Den Computer einschalten, eine Master-Diskette in Laufwerk A einlegen und <CR> betätigen.
- Die Eingabeaufforderungen für Datum und Zeit werden angezeigt, wenn MS-DOS in den Speicher gelesen wird. Obwohl diese Eingabeaufforderungen umgangen werden können, liefern sie doch wichtige Informationen über das Erstellungs- oder Aktualisierungsdatum der Daten auf der Diskette.
- Die mit diesem Handbuch gelieferte MS-DOS-Diskette ist "schreibgeschützt." Der Benutzer hat eine Sicherheitskopie der Diskette angefertigt. Für die Verarbeitung sollte er nur diese neue Kopie benutzen. (Die Original-Master-Diskette soll an einem sicheren Ort zum Schutz des Systems aufbewahrt werden.)
- Die Software auf der MS-DOS-Diskette geht davon aus, daß der Benutzer über zwei Diskettenlaufwerke und einen Paralleldrucker verfügt. Abweichende Angaben werden MS-DOS mit Hilfe des CONFIG-Dienstprogramms mitgeteilt, mit dem die Konfigurationsbeschreibung auf der neuen Master-Diskette definiert und beschrieben wird.
- Jedes Plattenlaufwerk verfügt über einen einmaligen Namen (Laufwerkbezeichnung), bei dem es sich um einen Buchstaben handelt. Eine Festplatteneinheit verfügt über zwei logische Plattenlaufwerke und demzufolge über zwei Laufwerkbezeichnungen.
- Das A> ist die Standard-Systemmeldung. Mit ihr wird angegeben, welches Laufwerk MS-DOS benutzt. Außerdem weist sie darauf hin, daß MS-DOS auf die Anweisungen des Benutzers wartet. Die Standardlaufwerkbezeichnung kann durch Eingabe einer neuen Laufwerkbezeichnung, gefolgt von einem Doppelpunkt (:) geändert werden.
- Die meisten über die Tastatur vorgenommenen Eingaben müssen durch Betätigung der <CR>-Taste beendet werden. Mit dieser Funktionstaste wird MS-DOS mitgeteilt, daß der Benutzer eine Eingabe beendet hat.
- Jede neue Platte muß mit dem FORMAT-Befehl formatiert werden, bevor sie von MS-DOS benutzt werden kann. Aufgrund der hohen Speicherkapazität einer Festplatte kann die Formatierung einige Minuten dauern.
- Sämtliche Daten auf der Diskette werden in Dateien gespeichert. Jeder Dateiname ist in einem Inhaltsverzeichnis aufgelistet. Mit dem DIR-Befehl wird das Inhaltsverzeichnis an-

gezeigt.

- Es muß stets eine Kopie der wichtigen Daten auf der Diskette mit den entsprechenden Kopierbefehlen von MS-DOS angefertigt werden: DISKCOPY, COPY oder BACKUP.
- Die Tasten F1-F20 in der obersten Reihe der Tastatur stehen dem Benutzer für den eigenen Gebrauch zur Verfügung. Er kann sie für beliebige Funktionen (mit dem CONFIG-Dienstprogramm) "programmieren."
- MS-DOS verfügt über die Einrichtung der automatischen Programmausführung. Jedes Mal, wenn MS-DOS geladen wird, wird das definierte Programm automatisch ausgeführt.
- Nachdem die Verarbeitung beendet ist, wird die Diskette herausgenommen und der Computer ausgeschaltet.

MEHR ÜBER DATEIEN

In Kapitel 2 wurde erläutert, daß die Inhaltsverzeichnisse die Namen der Benutzerdateien enthalten. In diesem Kapitel wird beschrieben, wie die Dateien benannt und kopiert werden. Außerdem wird die hierarchische Verzeichnisstruktur bei MS-DOS dargelegt. Mit dieser Verzeichnisstruktur wird die Organisation und das Auffinden von Dateien für den Benutzer vereinfacht.

BENENNEN DER DATEIEN

Der Name einer typischen MS-DOS Datei sieht folgendermaßen aus:

```
NEWFILE.EXE
```

Der Name einer Datei besteht aus zwei Teilen. Der Dateiename ist NEWFILE und die Dateinamenerweiterung ist .EXE.

Ein Dateiname kann eine Länge von 1 bis 8 Zeichen aufweisen. Die Dateinamenerweiterung kann aus drei oder weniger Zeichen bestehen. Der Dateiname kann in Groß- oder Kleinbuchstaben eingegeben werden. MS-DOS übersetzt die Buchstaben in jedem Fall in Großbuchstaben um.

Zusätzlich zu dem Dateinamen und der Dateinamenerweiterung kann der Name der Benutzerdatei eine Laufwerkbezeichnung umfassen. Wird eine Laufwerkbezeichnung angegeben, so sucht MS-DOS auf der Platte in dem angegebenen Laufwerk nach dem eingegebenen Dateinamen. Um beispielsweise Verzeichnisinformationen über die Datei NEWFILE.EXE auf der Platte in Laufwerk A zu suchen (wobei Laufwerk A *nicht* das Standardlaufwerk ist), wird folgender Befehl eingegeben:

```
DIR A:NEWFILE.EXE
```

Nun werden Verzeichnisinformationen über die Datei NEWFILE.EXE auf dem Bildschirm angezeigt.

Ist Laufwerk A das Standardlaufwerk, so sucht MS-DOS nur die Platte in Laufwerk A nach dem Dateinamen NEWFILE ab. In diesem Fall ist also die Laufwerkbezeichnung nicht erforderlich. Mit einer Laufwerkbezeichnung wird MS-DOS angewiesen, eine Datei in einem anderen Laufwerk zu suchen.

Ein Dateiname besteht im allgemeinen aus Buchstaben und Zahlen. Es sind jedoch auch andere Zeichen zulässig. Für Dateinamenerweiterungen und Dateinamen sind dieselben Zeichen zulässig. Nachfolgend eine Aufstellung der Zeichen die in Dateinamen und in Dateinamenerweiterungen benutzt werden können:

A - Z 0 - 9 $ & #

% ' ( ) - @

\ ^ [ ] ~ ` !

Ein Dateiname umfaßt eine Dateispezifikation. Der Ausdruck Dateispezifikation (oder Dateispez) wird in diesem Handbuch zur Angabe des folgenden Dateinamenformats benutzt:

[<Laufwerkbezeichnung:>] <Dateiname> [<Dateinamenerweiterung>]

Hier wird nochmals darauf hingewiesen, daß wahlweise Elemente in eckigen Klammern stehen. Mit spitzen Klammern (< >) wird angegeben, daß der Benutzer den Text für das jeweilige Element liefert. Die Laufwerkbezeichnung wird nur benötigt, wenn MS-DOS eine bestimmte Datei auf einer bestimmten Platte suchen soll. Der Dateiname muss nicht unbedingt eine Dateinamenerweiterung aufweisen.

Hier einige Beispiele für Dateispezifikationen:

```
B:MYPROG.COB
A:YOURPROG.EXT
A:NEWFILE.
TEXT
```

DATEIGRUPPENSYMBOLE

In Dateinamen und Dateinamenerweiterungen können zwei Sonderzeichen benutzt werden (die als Dateigruppensymbole bezeichnet werden): das Sternchen (*) und das Fragezeichen (?).

Durch diese Sonderzeichen wird die Benutzung von Dateinamen in MS-DOS Befehlen flexibler.

Dateigruppensymbol ?
Ein Fragezeichen (?) in einem Dateinamen oder einer Dateinamenerweiterung gibt an, daß an dieser Stelle jedes beliebige Zeichen stehen kann. Durch den MS-DOS Befehl

```
DIR TEST?RUN.EXE
```

werden alle Verzeichniseinträge in dem Standardlaufwerk aufgelistet, die eine Länge von acht Zeichen haben, mit TEST beginnen, als nächstes Zeichen ein beliebiges Zeichen aufweisen, mit den Buchstaben RUN enden und eine Dateinamenerweiterung .EXE aufweisen.

Hier einige Beispiele für Dateien die aufgrund des vorhergehenden DIR-Befehls aufgelistet werden:

```
TEST1RUN.EXE
TEST2RUN.EXE
TEST6RUN.EXE
```

Dateigruppensymbol *
Ein Sternchen (*) in einem Dateinamen oder einer Dateinamenerweiterung gibt an, daß ein beliebiges Zeichen an dieser Stelle oder an irgendeiner der restlichen Stellen in dem Dateinamen oder der Dateinamenerweiterung stehen kann. Durch:

```
DIR TEST*.EXE
```

werden beispielsweise sämtliche Verzeichniseinträge in dem Standardlaufwerk mit Dateinamen aufgelistet, die mit TEST beginnen und eine Erweiterung .EXE aufweisen. Hier einige Beispiele für Dateien, die aufgrund dieses DIR-Befehls aufgelistet werden können:

```
TEST1RUN.EXE
TEST2RUN.EXE
TEST6RUN.EXE
TESTALL.EXE
```

Die Bezeichnung *.* bezieht sich auf sämtliche Dateien auf der Platte. Diese Bezeichnung kann in MS-DOS Befehlen sehr wir-

kungsvoll und gleichermaßen gefährlich sein. Durch den Befehl DEL*.* werden beispielsweise sämtliche Dateien im Standardlaufwerk gelöscht, unabhängig von Dateinamen oder Dateinamenerweiterung.

Beispiele

Zur Auflistung der Verzeichniseinträge für sämtliche Dateien namens NEWFILE in Laufwerk A (unabhängig von den Dateinamenerweiterungen) wird einfach:

```
DIR A:NEWFILE.*
```

eingegeben.

Zur Auflistung der Verzeichniseinträge für sämtliche Dateien mit einer Dateinamenerweiterung .TXT (unabhängig von den Dateinamen) auf der Platte in Laufwerk A wird:

```
DIR B:????????.TXT
```

eingegeben.

Dieser Befehl ist besonders nützlich wenn beispielsweise sämtliche Textprogramme eine Dateinamenerweiterung von .TXT aufweisen. Wird der DIR-Befehl mit Dateigruppensymbolen benutzt, so ist eine Auflistung sämtlicher Textdateien erhältlich, selbst wenn sich der Benutzer nicht an alle Dateinamen erinnert.

UNZULÄSSIGE DATEINAMEN

Einige Namen für logische Ein- und Ausgabegeräte werden von MS-DOS auf besondere Weise behandelt. Bestimmte, aus drei Buchstaben bestehende Namen sind für diese Einheiten reserviert. Die folgenden, aus drei Buchstaben bestehenden Namen dürfen nicht als Dateinamen oder Erweiterungen benutzt werden.

AUX
: Wird bei der Bezugnahme auf die Eingabe von oder die Ausgabe an eine Zusatzeinheit (wie beispielsweise einen Drucker oder ein Plattenlaufwerk) benutzt.

CON
: Wird bei der Bezugnahme auf die Tastatureingabe oder die Ausgabe an die Terminalkonsole (den Bildschirm) benutzt.

LST oder PRN
: Wird bei der Bezugnahme auf die Druckeinheit benutzt.

NUL
Wird benutzt wenn keine besondere Datei erstellt werden soll, der Befehl jedoch einen Eingabe- oder Ausgabedateinamen benötigt.

Selbst wenn Laufwerkbezeichnungen oder Dateinamenerweiterungen zu diesen Dateinamen hinzugefügt werden, bleiben sie dennoch mit den oben aufgeführten Einheiten verknüpft. So bezieht sich A:CON.XXX immer auf die Konsole und ist nicht der Name einer Plattendatei.

KOPIEREN DER DATEIEN

Von einer Plattendatei werden häufig mehrere Kopien benötigt. Mit dem COPY-Befehl können eine oder mehrere Dateien auf eine andere Platte kopiert werden. Der Kopie kann auch ein anderer Name zugewiesen werden, wenn der neue Name in dem COPY-Befehl angegeben wird.

Mit dem COPY-Befehl können auch Kopien von Dateien auf derselben Platte angefertigt werden. In diesem Fall muss ein anderer Dateiname zugewiesen werden, ansonsten wird die Datei überschrieben. Eine Kopie einer Datei auf derselben Platte kann nur angefertigt werden, wenn ein anderer Dateiname für die neue Kopie angegeben wird.

Der COPY-Befehl hat folgendes Format:

COPY Dateispez [Dateispez]

Zum Beispiel:

COPY A:MYFILE.TXT B:MYFILE.TXT

kopiert die Datei MYFILE.TXT auf Platte A in eine Datei namens MYFILE.TXT auf Platte B. Auf diese Weise ist die Datei MYFILE.TXT doppelt vorhanden.

In Abbildung 3.1 wird dargestellt, wie Dateien auf eine andere Platte kopiert werden:

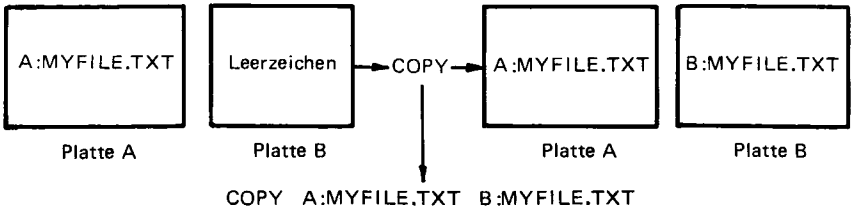


Abbildung 3.1 Kopieren von Dateien auf eine andere Platte

Soll die Datei namens MYFILE.TXT auf dieselbe Platte kopiert werden, so wird:

```
COPY A:MYFILE.TXT  A:NEWNAME.TXT
```

eingegeben. Auf diese Weise sind zwei Kopien der Datei auf Platte A vorhanden. Die eine trägt den Namen MYFILE.TXT und die andere den Namen NEWNAME.TXT. In der folgenden Abbildung wird dieses Beispiel dargestellt.

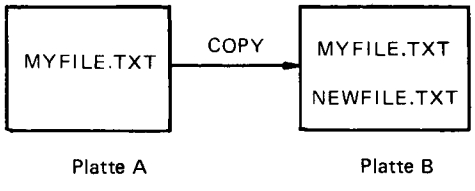


Abbildung 3.2 Kopieren von Dateien auf derselben Platte

Mit dem COPY-Befehl können auch sämtliche Dateien auf einer Platte auf eine andere Platte kopiert werden. In diesem Fall spricht man von einer Sicherheitskopie. Für weitere Informationen über dieses Verfahren wird auf Kapitel 5, MS-DOS Befehle, verwiesen.

SCHÜTZEN DER DATEIEN

MS-DOS erweist sich bei der Verarbeitung von persönlichen und geschäftlichen Daten als leistungsfähiges und nützliches System. Wie bei jedem Informationssystem kann es auch hier zu versehentlichen Fehlern und zu einem Mißbrauch von Informationen kommen. Werden Informationen verarbeitet, die nicht ersetzt werden

können oder einer besonderen Geheimhaltung unterliegen, so müssen besondere Schritte unternommen werden, um sicherzustellen, daß die Daten und Programme vor versehentlicher oder unberechtigter Benutzung, Änderung oder Zerstörung geschützt werden. Schon durch einfache Maßnahmen wie beispielsweise das Herausnehmen von Platten, wenn sie nicht benutzt werden, das Anfertigen von Sicherheitskopien bei wichtigen Informationen und die Installation der Anlage an einem sicheren Ort, kann die Integrität der Dateien gewahrt werden. Darüberhinaus kann ein MS-DOS Befehl, CIPHER, zu Verschlüsselung der Dateien im Hinblick auf größte Sicherheit benutzt werden. Für weitere Informationen über CIPHER sei auf Kapitel 5, MS-DOS Befehle, verwiesen.

INHALTSVERZEICHNISSE

Wie schon in Kapitel 2 beschrieben, sind die Namen der Dateien in einem Inhaltsverzeichnis auf jeder Platte festgehalten. Das Inhaltsverzeichnis enthält außerdem Informationen über die Grösse der Dateien, ihre Adressen auf der Platte und das Datum des Erstellens und Aktualisierens der Dateien.

Bei mehreren Benutzern oder bei der Arbeit an verschiedenen Projekten, kann die Anzahl von Dateien in dem Inhaltsverzeichnis sehr umfangreich und unhandlich werden. Unter Umständen möchte der Benutzer seine eigenen Dateien von denen seiner Mitarbeiter getrennt halten oder seine Programme in Kategorien unterteilen die dem jeweiligen Bedarf gerecht werden.

In einem Büro können Akten in verschiedenen Aktenschränken getrennt gehalten werden. Dies ist mit dem Erstellen von verschiedenen Inhaltsverzeichnissen vergleichbar. Mit MS-DOS können die Dateien auf den Platten in verschiedenen Inhaltsverzeichnissen gespeichert werden. Durch Inhaltsverzeichnisse sind die Dateien in entsprechende Dateigruppen unterteilbar. So können beispielsweise alle Buchhaltungsprogramme in einem Inhaltsverzeichnis und alle Textdateien in einem anderen gespeichert werden. Jedes Inhaltsverzeichnis kann eine bestimmte Anzahl von Dateien enthalten. Es kann außerdem weitere Inhaltsverzeichnisse umfassen (die als Unterverzeichnisse bezeichnet sind). Diese Methode der Dateiorganisation wird als hierarchische Verzeichnisstruktur bezeichnet.

Eine hierarchische Verzeichnisstruktur ist mit einer "Baum"-Struktur vergleichbar: die Inhaltsverzeichnisse sind die Zweige und die Dateien die Blätter. Allerdings befindet sich der "Stamm" nicht in der Mitte, sondern an der Spitze. Der Stamm ist die erste Ebene in der Verzeichnisstruktur. Dieses Verzeichnis wird automatisch erstellt wenn eine Platte formatiert und die ersten Dateien auf die Platte gespeichert werden. Weitere Inhaltsverzeichnisse und Unterverzeichnisse können anhand der Instruktionen in Kapitel 4, Informationen über Befehle, erstellt werden.

Der Baum oder die Dateistruktur wächst mit dem Erstellen neuer Inhaltsverzeichnisse für Dateigruppen oder für andere Benutzer in dem System. Innerhalb jedes neuen Inhaltsverzeichnisses können Dateien hinzugefügt oder neue Unterverzeichnisse erstellt werden.

Auf diesem Baum kann "geklettert" werden. So ist beispielsweise jede beliebige Datei in dem System auffindbar, indem beim Stamm begonnen und auf die Zweige zu der gewünschten Datei geklettert wird. Umgekehrt kann auch von dem jeweiligen Dateisystem zu dem Stamm geklettert werden.

Die am Anfang dieses Kapitels besprochenen Dateinamen beziehen sich auf das gegenwärtige Inhaltsverzeichnis. Sie gelten nicht für das gesamte System. Beim Einschalten des Computers befindet sich der Benutzer also "in" seinem Inhaltsverzeichnis. Unternimmt man keine besonderen Schritte beim Erstellen einer Datei, so wird die neue Datei in dem Inhaltsverzeichnis erstellt, in dem gerade gearbeitet wird. So können Dateien mit dem selben Namen vorhanden sein, die in keiner Beziehung zueinander stehen da sie in verschiedenen Inhaltsverzeichnissen stehen.

Abbildung 3.3 stellt eine typische hierarchische Verzeichnisstruktur dar.

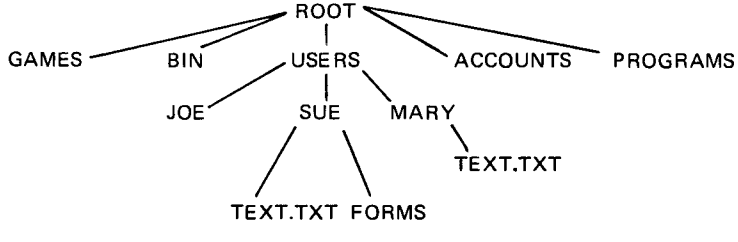


Abbildung 3.3 Beispiel für eine hierarchische Verzeichnisstruktur

Das ROOT-Inhaltsverzeichnis (Stammverzeichnis) stellt die erste Ebene in der Verzeichnisstruktur dar. Mit dem MKDIR-Befehl können Unterverzeichnisse ausgehend von ROOT erstellt werden. (Für weitere Informationen über MKDIR sei auf Kapitel 5, MS-DOS Befehle, verwiesen.) In diesem Beispiel wurden fünf Unterverzeichnisse von ROOT erstellt. Sie umfassen im einzelnen folgende Verzeichnisse:

- Ein Verzeichnis mit Computerspielen namens GAMES.
- Ein Verzeichnis mit sämtlichen externen Befehlen namens BIN (für weitere Informationen über das BIN-Inhaltsverzeichnis siehe Kapitel 4, Informationen über Befehle).
- Ein USER-Inhaltsverzeichnis mit separaten Unterverzeichnissen für sämtliche Benutzer des Systems.
- Ein Inhaltsverzeichnis mit Abrechnungsinformationen namens ACCOUNTS.
- Ein Inhaltsverzeichnis mit Programmen namens PROGRAMS.

Joe, Sue und Mary haben jeweils ihre eigenen Inhaltsverzeichnisse. Hier handelt es sich um Unterverzeichnisse des USER-Inhaltsverzeichnisses. Sue hat ein Unterverzeichnis unter dem \USER\SUE Inhaltsverzeichnis namens FORMS. Sue and Mary haben beide Dateien namens TEXT.TXT in ihren Inhaltsverzeichnissen. Hier muss darauf geachtet werden, daß die Textdatei von Mary in keiner Beziehung zu der von Sue steht.

Diese Organisation von Dateien und Inhaltsverzeichnissen ist nicht wichtig, wenn nur mit Dateien in dem eigenen Inhaltsverzeichnis gearbeitet wird. Wird jedoch mit einem anderen Benutzer oder mit verschiedenen Projekten gleichzeitig gearbeitet, so wird die hierarchische Verzeichnisstruktur äußerst nützlich. So könnte man beispielsweise durch Eingabe von:

```
DIR \ USER \ SUE \ FORMS
```

eine Auflistung der Dateien in dem FORMS-Inhaltsverzeichnis von Sue erhalten. Der umgekehrte Schrägstrich (\) wird dazu benutzt, Inhaltsverzeichnisse von anderen Inhaltsverzeichnissen und Dateien zu trennen.

Um die Dateien in dem Inhaltsverzeichnis von Mary zu ermitteln, wird:

```
DIR \ USER \ MARY
```

eingegeben.

DATEINAMEN UND PFADE

Werden hierarchische Inhaltsverzeichnisse benutzt, so muss der Benutzer MS-DOS mitteilen, wo die Dateien in der Verzeichnisstruktur stehen. So verfügen beispielsweise sowohl Mary als auch Sue über Dateien namens TEXT.TXT. Jede der beiden muss MS-DOS sagen, in welchem Inhaltsverzeichnis ihre Datei steht. Zu diesem Zweck ist MS-DOS ein Pfadname zu der Datei anzugeben.

Pfadnamen

Ein einfacher Dateiname ist eine Folge von Zeichen, vor der wahlweise eine Laufwerkbezeichnung und nach der eine Erweiterung stehen kann. Ein Pfadname ist eine Folge von Verzeichnisnamen, gefolgt von einem einfachen Dateinamen. Die einzelnen Namen werden durch einen umgekehrten Schrägstrich (\) voneinander getrennt.

Die Pfadnamen haben folgende Syntax:

[<d>:] [<Inhaltsverzeichnis>] \ [<Inhaltsverzeichnis. . . >] \ [<Dateiname>]

Beginnt ein Pfadname mit einem Schrägstrich, so sucht MS-DOS ab dem Stamm des Baumes nach der Datei. Ansonsten beginnt MS-DOS die Suche in dem gegenwärtigen Inhaltsverzeichnis des Benutzers, das als Arbeitsverzeichnis bezeichnet wird. Von dort an wird nach unten weiter gesucht. Der Pfadname der TEXT.TXT-Datei von Sue lautet \USER\SUE\TEXT.TXT.

Wird im Arbeitsverzeichnis gearbeitet, so können ein Dateiname und der entsprechende Pfadname abwechselnd benutzt werden. Nachfolgend einige Beispiele für Pfadnamen:

\
: Zeigt das Stammverzeichnis an.

\PROGRAMS
: Unterverzeichnis unter dem Stammverzeichnis mit Programmdateien.

\USER\MARY\FORMS\1A
: Ein typischer vollständiger Pfadname. Hier handelt es sich um eine Datei namens 1A in dem Inhaltsverzeichnis namens FORMS das dem USER namens MARY gehört.

USER\SUE
Ein relativer Pfadname. Mit ihm wird die Datei oder das Inhaltsverzeichnis SUE in dem Unterverzeichnis USER des Arbeitsverzeichnisses benannt. Ist das Arbeitsverzeichnis das Stammverzeichnis (\), so lautet der Name \BIN\SUE.

TEXT.TXT
Name einer Datei oder eines Inhaltsverzeichnisses im Arbeitsverzeichnis.

Bei MS-DOS stehen bestimmte Abkürzungen für das Arbeitsverzeichnis und das übergeordnete Inhaltsverzeichnis des Arbeitsverzeichnisses zur Verfügung. Das übergeordnete Verzeichnis des in diesem Kapitel beispielsweise angeführten Verzeichnisses JOE heißt USER.

. (einzelner Punkt)
MS-DOS benutzt diese Abkürzung, um den Namen des Arbeitsverzeichnisses in allen hierarchischen Verzeichnisauflistungen anzugeben. MS-DOS erstellt diesen Eintrag automatisch wenn ein Inhaltsverzeichnis angefertigt wird. Beispiel: Ihr Arbeitsverzeichnis sei JOE. Wenn Sie das DIR-Kommando eingeben, zeigt MS-DOS einen einzelnen Punkt anstelle des eigentlichen Namens Ihres Arbeitsverzeichnisses an.

. . (doppelter Punkt)
Die Abkürzung für das übergeordnete Verzeichnis des Arbeitsverzeichnisses. Im oben angegebenen Beispiel zeigt MS-DOS zwei Punkte (ohne Leerraum) anstelle des übergeordneten Verzeichnisses USER an. Wenn Sie MS-DOS anweisen wollen, zu der vorherigen Verzeichnisebene zurückzukehren, können Sie demgemäß zwei Punkte bei Angabe des Pfads benutzen. Beispiel: Ihr Arbeitsverzeichnis sei JOE. Es gibt zwei mögliche Verfahrensweisen, die Datei FORMS im Verzeichnis USER SUE aufzufinden:

```
\USER\SUE\FORMS
```

oder

```
..SUE\FORMS
```

Die zwei Punkte bewirken eine Rückkehr zur vorherigen Ebene. Von dieser aus wird der Pfad weiterverfolgt.

Pfade und externe Befehle – Externe Befehle stehen als Prorammdateien auf den Platten. Sie müssen vor der Ausführung von der Platte gelesen werden. (Für weitere Informationen über die externen Befehle siehe Kapitel 4, Informationen über Befehle.)

Wenn Sie mit mehr als einem Inhaltsverzeichnis arbeiten, so empfiehlt es sich, alle externen Befehle von MS-DOS in ein separates Inhaltsverzeichnis zu setzen. Auf diese Weise kann es zu keiner Verwechslung mit anderen Inhaltsverzeichnissen kommen. Bei Ausgabe eines externen Befehls an MS-DOS , prüft MS-DOS sofort das Arbeitsverzeichnis, um diesen Befehl zu finden. Der Benutzer muss MS-DOS mitteilen, in welchem Inhaltsverzeichnis diese externen Befehle stehen. Dies geschieht mit dem PATH-Befehl.

Wird beispielsweise in einem Arbeitsverzeichnis namens \BIN\ PROG gearbeitet, und stehen sämtliche externen Befehle von MS-DOS in \BIN, so muss der Benutzer MS-DOS anweisen, den FORMAT-Befehl über den Pfad \BIN zu wählen, um den FORMAT-Befehl zu suchen. Durch den Befehl:

```
PATH \ BIN
```

wird MS-DOS angewiesen, nach allen Befehlen im Arbeitsverzeichnis des Benutzers und dem \BIN Inhaltsverzeichnis zu suchen. Dieser Pfad braucht während der Terminalsitzung nur ein Mal für MS-DOS angegeben werden. MS-DOS sucht nun nach den externen Befehlen in \BIN. Möchte der Benutzer den gegenwärtigen Pfad ermitteln, so gibt er das Wort PATH ein. Dann wird der gegenwärtige Wert von PATH ausgedruckt.

Für weitere Informationen über den MS-DOS Befehl PATH, siehe Kapitel 5, MS-DOS Befehle.

Pfade und Interne Befehle – Die internen Befehle sind die einfachsten und gebräuchlichsten Befehle. Sie werden sofort ausgeführt, da sie in den Befehlsprozessor eingebaut sind. (Für weitere Informationen über die internen Befehle siehe Kapitel 4, Informationen über Befehle.)

Einige interne Befehle können Pfade benutzen. Die vier Befehle COPY, DIR, DEL und TYPE werden flexibler, wenn im Anschluss an den Befehl ein Pfadname angegeben wird.

COPY <Pfadname Pfadname>

Handelt es sich bei dem zweiten Pfadnamen für COPY um ein Inhaltsverzeichnis, so werden sämtliche Dateien in dieses Inhaltsverzeichnis kopiert. Der erste Pfadname kann nur Dateien im Arbeitsverzeichnis angeben.

DEL <Pfadname>

Handelt es sich bei dem Pfadnamen um ein Inhaltsverzeichnis, so werden sämtliche Dateien in diesem Inhaltsverzeichnis gelöscht. Versucht der Benutzer, einen Pfad zu löschen, so wird die Systemmeldung "Are you sure (Y/N)?" (Sind Sie sicher (J/N)?) angezeigt. Soll der Befehl ausgeführt werden, so wird Y eingegeben. Soll er abgebrochen werden, wird N eingegeben.

DIR <Pfadname>

Zeigt das Inhaltsverzeichnis für einen bestimmten Pfad an.

TYPE <Pfadname>

Für diesen Befehl muss eine Datei in einem Pfad angegeben werden. Als Antwort auf den TYPE-Pfadname-Befehl zeigt MS-DOS die Datei auf dem Bildschirm an.

ANZEIGE DES ARBEITSVERZEICHNISSES

Wird in dem Arbeitsverzeichnis gearbeitet, so werden sämtliche Befehle ausgeführt. Durch Ausgabe des MS-DOS Befehls CHDIR (Inhaltsverzeichnis ändern), ohne Optionen, läßt sich der Name des gegenwärtigen Inhaltsverzeichnisses ermitteln. Ist das gegenwärtige Inhaltsverzeichnis des Benutzers beispielsweise \USER\JOE, so erhält der Benutzer nach Eingabe von:

CHDIR <RETURN>

die Bildschirmanzeige:

A: \ USER \ JOE

Hier handelt es sich um die gegenwärtige Laufwerkbezeichnung plus dem Arbeitsverzeichnis (\USER\JOE).

Wollen Sie nun den Inhalt des Verzeichnisses \USER\JOE feststellen, so kann der MS-DOS Befehl DIR ausgegeben werden. Nachfolgend ein Beispiel für eine Bildschirmanzeige im Anschluss an einen DIR-Befehl für ein Unterverzeichnis:

```
Volume in drive A has no ID (Datenträger in Laufwerk A ohne
                                Kennsatz)
Directory of A: \USER\JOE (Inhaltsverzeichnis von A:
                                \USER\JOE)

.           <DIR>          5-09-83   10:09a
..          <DIR>          5-09-83   10:09a
TEXT        <DIR>          5-09-83   10:09a
FILE1  COM          5243   5-04-83    9:30a
      4 File(s)      250518 bytes free
(4 Datei(en)         250518 freie Bytes)
```

Beim Formatieren der Platte wurde kein Datenträgerkennsatz für diese Platte zugewiesen. Hier wird darauf hingewiesen, daß MS-DOS in dieser Ausgabe sowohl Dateien als auch Inhaltsverzeichnisse auflistet. Wie aus dieser Ausgabe ersichtlich, verfügt Joe über ein weiteres Inhaltsverzeichnis namens TEXT in dieser Baumstruktur. Der '.' zeigt das Arbeitsverzeichnis \USER\JOE an. '. .' ist die Abkürzung für das übergeordnete Verzeichnis \USER.FILE1 .COM ist eine Datei im Inhaltsverzeichnis \USER\JOE. Alle diese Inhaltsverzeichnisse und Dateien befinden sich auf der Platte in Laufwerk A.

Da Dateien und Inhaltsverzeichnisse zusammen aufgelistet werden (siehe vorhergehende Bildschirmanzeige), darf ein Unterverzeichnis nicht denselben Namen wie eine Datei in diesem Inhaltsverzeichnis haben. Bei einem Pfad \BIN\USER\JOE, wobei JOE ein Unterverzeichnis ist, darf beispielsweise keine Datei namens JOE in dem USER-Inhaltsverzeichnis erstellt werden.

ERSTELLEN EINES INHALTSVERZEICHNISSES

Mit dem MKDIR-Befehl (Verzeichnis erstellen) läßt sich ein Unterverzeichnis in dem Arbeitsverzeichnis erstellen. Um ein neues Inhaltsverzeichnis namens NEWDIR unter dem Arbeitsverzeichnis zu erstellen, wird einfach:

```
MKDIR NEWDIR
```

eingegeben.

Nachdem dieser Befehl von MS-DOS ausgeführt wurde, ist ein neues Inhaltsverzeichnis in der Baumstruktur unter dem Arbeitsverzeichnis vorhanden. Es lassen sich auch Inhaltsverzeichnisse an beliebiger Stelle in der Baumstruktur erstellen, indem MKDIR

und danach ein Pfadname angegeben wird. MS-DOS erstellt automatisch die Einträge . und . . im neuen Inhaltsverzeichnis.

Mit dem MS-DOS Zeilen-Editor, EDLIN, werden Dateien in das neue Inhaltsverzeichnis gesetzt. Kapitel 7, Zeilen-Editor (EDLIN), beschreibt, wie Dateien mit EDLIN erstellt und gesichert werden.

ÄNDERN DES ARBEITSVERZEICHNISSES

Der Wechsel vom Arbeitsverzeichnis zu einem anderen Inhaltsverzeichnis ist bei MS-DOS sehr einfach. Es braucht nur der CHDIR-Befehl (Inhaltsverzeichnis ändern) ausgegeben und ein Pfadname angegeben zu werden. Zum Beispiel:

```
A:CHDIR \USER
```

ändert das Arbeitsverzeichnis von \USER\JOE in \USER. Im Anschluss an den Befehl kann ein beliebiger Pfadname angegeben werden, um zu den verschiedenen Zweigen und Blättern des Verzeichnisbaumes zu "klettern." Durch den Befehl "CHDIR. ." gelangt man immer in das übergeordnete Verzeichnis des Arbeitsverzeichnisses.

LÖSCHEN EINES INHALTSVERZEICHNISSES

Mit dem MS-DOS Befehl RMDIR (Inhaltsverzeichnis löschen) wird ein Inhaltsverzeichnis in der Baumstruktur gelöscht. Um beispielsweise das Inhaltsverzeichnis NEWDIR aus dem Arbeitsverzeichnis zu löschen, wird:

```
RMDIR NEWDIR
```

eingegeben. Hier sei darauf hingewiesen, daß das Inhaltsverzeichnis NEWDIR mit Ausnahme der Einträge . und . . leer sein muss, bevor es gelöscht werden kann. Dadurch wird verhindert, daß Dateien und Inhaltsverzeichnisse versehentlich gelöscht werden. Jedes Inhaltsverzeichnis läßt sich durch Angabe des Pfadnamens löschen. Um das Inhaltsverzeichnis \BIN\USER\JOE zu löschen, muss sich der Benutzer vergewissern, daß nur noch die Einträge . und . . vorhanden sind. Danach wird:

```
RMDIR \BIN\USER\JOE
```

eingegeben.

Um sämtliche Dateien in einem Inhaltsverzeichnis zu löschen (mit Ausnahme der Einträge . und . .), wird DEL und danach der Pfadname des Inhaltsverzeichnisses eingegeben. Um beispielsweise sämtliche Dateien im Inhaltsverzeichnis \BIN\USER\SUE zu löschen, wird:

```
DEL \ BIN \ USER \ SUE
```

eingegeben.

Die Einträge . und . . können nicht gelöscht werden. Sie werden von MS-DOS als Teil der hierarchischen Verzeichnisstruktur erstellt.

Im nächsten Kapitel werden die MS-DOS Befehle näher beschrieben.

INFORMATIONEN ÜBER BEFEHLE

ALLGEMEINE INFORMATION

Die Befehle stellen die Verbindung mit dem Computer her. Nach Eingabe der entsprechenden MS-DOS Befehle am Terminal führt das System folgende Aufgaben aus:

- Vergleichen, Kopieren, Anzeigen, Löschen und Umbenennen von Dateien.
- Kopieren und Formatieren von Platten.
- Ausführen von Systemprogrammen, wie beispielsweise EDLIN, sowie von Benutzerprogrammen.
- Analysieren und Auflisten von Inhaltsverzeichnissen.
- Eingeben von Datum und Zeit, sowie von Bemerkungen.
- Festlegen verschiedener Drucker- und Bildschirmoptionen.
- Kopieren von MS-DOS Systemdateien auf eine andere Platte.
- Versetzen des Betriebssystems MS-DOS in den Wartestatus während einer bestimmten Zeitdauer.

ARTEN VON MS-DOS BEFEHLEN

Es gibt zwei Arten von MS-DOS Befehlen: interne Befehle und externe Befehle.

Interne Befehle

Die internen Befehle sind die einfachsten und gebräuchlichsten Befehle. Diese Befehle werden bei der Auflistung eines Inhaltsverzeichnisses auf der MS-DOS Platte nicht sichtbar. Sie sind Teil des Befehlsprozessors. Wenn diese Befehle eingegeben werden, werden sie sofort ausgeführt. Die folgenden internen Befehle sind in Kapitel 5 beschrieben:

BREAK	DEL (ERASE)	MKDIR (MD)	SET
CHDIR (CD)	DIR	PATH	SHIFT
CLS	ECHO	PAUSE	TIME
COPY	EXIT	PROMPT	TYPE
CTTY	FOR	REM	VER
DATE	GOTO	REN (RENAME)	VERIFY
	IF	RMDIR (RD)	VOL

Externe Befehle

Externe Befehle stehen als Programmdateien auf Platte. Sie müssen von der Platte gelesen werden bevor sie ausgeführt werden können. Liegt die Platte mit dem Befehl nicht im Laufwerk, so kann MS-DOS den Befehl nicht finden und ausführen.

Jeder Dateiname mit einer Dateinamenerweiterung von .COM, .EXE oder .BAT wird als externer Befehl betrachtet. Programme wie beispielsweise FORMAT.COM sind externe Befehle. Da sämtliche externen Befehle auf Platte stehen, können Befehle erstellt und zum System hinzugefügt werden. Programme die mit den meisten Sprachen (einschließlich der Assemblersprache) erstellt werden, sind .EXE-Dateien (ausführbare Dateien).

Bei der Eingabe eines externen Befehls muß man die Dateinamenerweiterung nicht angeben. Die folgenden externen Befehle sind in Kapitel 5 beschrieben:

BACKUP	LOCATE
CHKDSK	MORE
CIPHER	PRINT
CONFIG	RDCPM
DISKCOPY	RECOVER
FIND	SORT
FORMAT	SYS

BEFEHLSOPTIONEN

In den MS-DOS Befehlen können Optionen zur Angabe von zusätzlichen Informationen für das System enthalten sein. Werden keine Optionen angegeben, so liefert MS-DOS einen Standardwert. Für die Standardwerte wird auf die einzelnen Befehlsbeschreibungen in Kapitel 5 verwiesen.

Sämtliche MS-DOS Befehle weisen das folgende Format auf:

Befehl [Optionen . . . . .]

Erläuterung der Optionen:

d:
: Bezieht sich auf die Laufwerkbezeichnung.

Dateiname
: Bezieht sich auf einen beliebigen, gültigen Namen für eine Plattendatei, einschließlich einer wahlweisen Dateinamener-

weiterung. Dateiname bezieht sich nicht auf eine Einheit oder auf eine Laufwerkbezeichnung.

.erw
Bezieht sich auf eine wahlweise Dateinamenerweiterung, die aus einem Punkt und 1 bis 3 Zeichen besteht. Die Dateinamenerweiterungen stehen direkt hinter dem Dateinamen.

Dateispez
Bezieht sich auf eine wahlweise Laufwerkbezeichnung, einen Dateinamen und eine wahlweise aus drei Buchstaben bestehende Dateinamenerweiterung in folgendem Format:

[<d:>] <Dateiname> [<.erw>]

Pfadname
Bezieht sich auf einen Pfadnamen oder Dateinamen in folgendem Format:

[<Inhaltsverzeichnis>] \ [<Inhaltsverzeichnis. . .>] \ [<Dateiname>]

Schalter
Schalter sind Optionen zur Steuerung der MS-DOS Befehle. Vor den Schaltern steht ein Schrägstrich (z.B. /P).

Parameter
Sie liefern weitere Informationen für MS-DOS Befehle. Normalerweise wird zwischen Parametern gewählt, z.B. ON oder OFF. (EIN oder AUS.)

ALLGEMEINGÜLTIGE EINGABEKONVENTIONEN

Die folgenden Informationen gelten für alle MS-DOS Befehle:

1. Auf die Befehle folgen im allgemeinen eine oder mehrere Optionen.
2. Befehle und Optionen können in Groß- oder Kleinbuchstaben oder einer Kombination der beiden eingegeben werden.
3. Befehle und Optionen müssen durch Abgrenzungszeichen voneinander getrennt werden. Am besten werden Leerzeichen und Komma als Abgrenzungszeichen benutzt.

```
DEL MYFILE.OLD NEWFILE.TXT
RENAME,THISFILE THATFILE
```

Das Semikolon (;), das Gleichheitszeichen (=) oder die Tabulatortaste können in MS-DOS Befehlen ebenfalls als Abgren-

zungszeichen benutzt werden. (In diesem Handbuch wird das Leerzeichen als Abgrenzungszeichen in Befehlen benutzt.)

4. Eine Dateispezifikation darf nicht mit Abgrenzungszeichen begrenzt werden, da der Doppelpunkt und der Punkt schon als Abgrenzungszeichen dienen.
5. Bei Instruktionen wie "Press any key" (Betätigen einer beliebigen Taste), kann jede beliebige Buchstabentaste (A-Z) oder Zifferntaste (0-9) betätigt werden.
6. Bei Bezugnahme auf eine Datei mit einer Dateinamenerweiterung muß diese Dateinamenerweiterung angegeben werden.
7. Während der Ausführung können Befehle durch Betätigen von <CONTROL-C> abgebrochen werden.
8. Befehle werden erst nach Betätigung der RETURN-Taste wirksam.
9. Dateigruppensymbole (globale Dateinamenzeichen) und Einheitennamen (z.B. PRN oder CON) dürfen in den Namen von Befehlen nicht benutzt werden.
10. Bei Befehlen mit einer umfangreichen Ausgabe auf dem Bildschirm läuft die Anzeige automatisch zu der nächsten Bildschirmanzeige. Durch Betätigung von <CONTROL-S> kann die Anzeige angehalten werden. Zur Wiederaufnahme der Bildschirmanzeige ist eine beliebige Taste zu betätigen.
11. Die MS-DOS Editier- und Funktionstasten können bei der Eingabe von Befehlen benutzt werden. Für eine genaue Beschreibung dieser Tasten wird auf Kapitel 6, MS-DOS Editier- und Funktionstasten, verwiesen.
12. Die Eingabeaufforderung des Befehlsprozessors besteht aus der Standardlaufwerkbezeichnung zusammen mit dem Grösser-als-Zeichen (>). Zum Beispiel A>.
13. Plattenlaufwerke werden als Quellenlaufwerke und Bestimmungslaufwerke bezeichnet. Ein Quellenlaufwerk ist das Laufwerk, von dem aus Informationen übertragen werden. Ein Bestimmungslaufwerk ist das Laufwerk, an das Informationen übertragen werden.

ABARBEITUNG IM BATCH-BETRIEB

Sehr häufig wird man dieselbe Folge von Befehlen zur Ausführung von häufig benutzten Aufgaben immer wieder eingeben. Mit MS-DOS kann die Befehlsfolge in eine Sonderdatei, die als Batch-Datei bezeichnet wird, gesetzt werden. Die gesamte Folge kann dann einfach durch Eingabe des Namens der Batch-Datei ausgeführt werden. "Stapel" von Befehlen in derartigen Dateien werden so verarbeitet,

als würden sie am Terminal eingegeben. Jede Batch-Datei muss eine Dateinamenerweiterung .BAT aufweisen. Sie wird ausgeführt, indem der Dateiname ohne die Erweiterung eingegeben wird.

Eine Batch-Datei kann mit Hilfe des Zeilen-Editors (EDLIN) oder durch Eingabe des COPY-Befehls erstellt werden. Für weitere Informationen über die Benutzung des COPY-Befehls für das Erstellen einer Batch-Datei wird auf den Abschnitt "Erstellen einer AUTOEXEC.BAT-Datei" später in diesem Kapitel verwiesen.

Zwei MS-DOS Befehle stehen ausdrücklich für die Benutzung in Batch-Dateien zur Verfügung: REM und PAUSE. Mit REM können Bemerkungen und Kommentare in den Batch-Dateien aufgenommen werden, ohne daß diese Bemerkungen als Befehle ausgeführt werden. PAUSE gibt eine Eingabeaufforderung in Form einer wahlweisen Meldung an den Benutzer aus. Im Anschluß an diese Meldung kann der Benutzer die Abarbeitung im Batch-Betrieb fortsetzen oder an einer bestimmten Stelle abbrechen. REM und PAUSE werden im einzelnen in Kapitel 5 beschrieben.

Die Abarbeitung im Batch-Betrieb ist besonders nützlich wenn verschiedene MS-DOS Befehle mit einem Batch-Befehl ausgeführt werden sollen, wie beispielsweise beim Formatieren und Überprüfen einer neuen Platte. Eine Batch-Datei für diesen Zweck könnte beispielsweise folgendermaßen aussehen:

```
1: REM — Dies ist eine Datei zur Überprüfung neuer Platten
2: REM — Sie trägt den Namen NEWDISK.BAT
3: PAUSE — Neue Platte in Laufwerk B: einlegen
4: FORMAT B:
5: DIR B:
6: CHKDSK B:
```

Zur Ausführung dieser .BAT-Datei wird einfach der Dateiname ohne die .BAT-Erweiterung eingegeben:

```
NEWDISK
```

Das Ergebnis ist dasselbe, als wäre jede Zeile in der .BAT-Datei am Terminal als einzelner Befehl eingegeben worden.

In Abbildung 4.1 werden die drei Schritte für das Schreiben, Sichern und Ausführen einer MS-DOS Batch-Datei dargestellt.

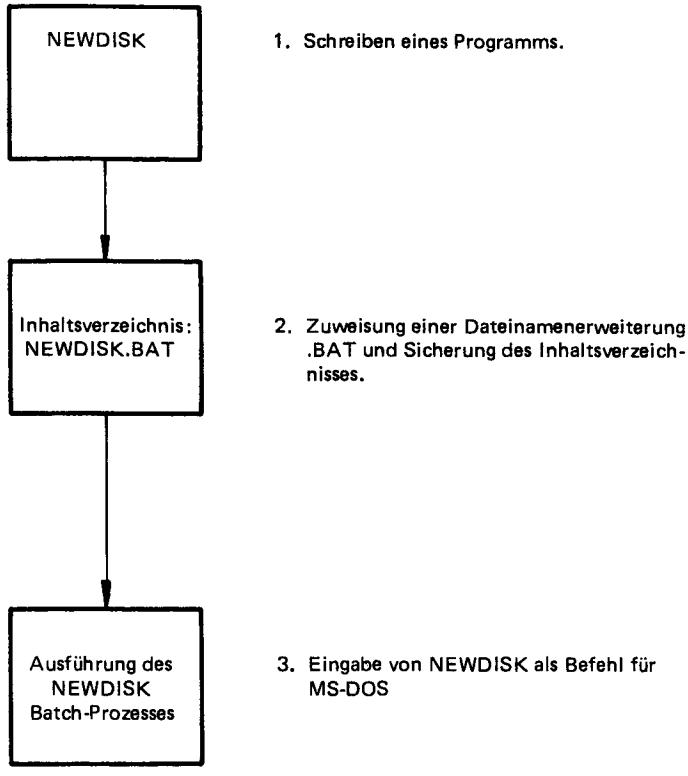


Abbildung 4.1 Schritte in einer MS-DOS Batch-Datei
Laden der MS-DOS Platte

Die folgende Aufstellung enthält beim Batch-Betrieb mit MS-DOS zu berücksichtigende Informationen.

1. Der Dateiname BATCH darf nicht eingegeben werden (es sei denn der Name der auszuführenden Datei lautet BATCH.BAT).
2. Zur Ausführung der Batch-Datei darf nur der Dateiname eingegeben werden. Die Dateinamenerweiterung wird nicht eingegeben.
3. Die Befehle in der Datei mit der Bezeichnung <Dateiname>. BAT werden ausgeführt.
4. Wird <CONTROL-C> während des Batch-Modus betätigt, so wird folgende Eingabeaufforderung auf dem Bildschirm angezeigt:

   Terminate batch job (Y/N)?
   Batch-Betrieb beenden (J/N)?

Wird Y betätigt, so werden die restlichen Befehle in der Batch-Datei ignoriert und die Systemmeldung wird angezeigt.

Wird N betätigt, so wird nur der gegenwärtige Befehl beendet und die Abarbeitung im Batch-Betrieb wird mit dem nächsten Befehl in der Datei forgesetzt.

5. Wird die Platte mit der gerade ausgeführten Batch-Datei herausgenommen, so fordert MS-DOS den Benutzer auf, die Platte erneut einzulegen, bevor der nächste Befehl gelesen werden kann.
6. Bei dem letzten Befehl in einer Batch-Datei kann es sich um den Namen einer anderen Batch-Datei handeln. Dadurch kann eine Batch-Datei aus einer anderen Batch-Datei heraus aufgerufen werden, wenn die erste Datei beendet ist.

AUTOEXEC.BAT-Datei

Wie in Kapitel 2 erläutert, können mit der AUTOEXEC.BAT-Datei beim Starten von MS-DOS automatisch Programme ausgeführt werden. Die automatische Programmausführung ist besonders nützlich, wenn ein bestimmtes Anwendungspaket unter MS-DOS ausgeführt werden soll, oder wenn MS-DOS bei jedem Starten des Systems automatisch ein Batch-Programm ausführen soll. Wird eine AUTOEXEC.BAT-Datei benutzt, so brauchen nicht mehr zwei separate Platten zur Ausführung dieser Aufgaben eingelegt werden.

Beim Starten von MS-DOS sucht der Befehlsprozessor die MS-DOS Platte nach einer Datei namens AUTOEXEC.BAT ab. Die AUTOEXEC.BAT-Datei ist eine Batch-Datei, die bei jedem Starten des Systems automatisch ausgeführt wird.

Findet MS-DOS die AUTOEXEC.BAT-Datei, so wird die Datei sofort vom Befehlsprozessor ausgeführt. In diesem Fall werden die Eingabeaufforderungen für Datum und Zeit umgangen.

Findet MS-DOS beim ersten Laden der MS-DOS Platte keine AUTOEXEC.BAT-Datei, so werden die Eingabeaufforderungen für Datum und Zeit ausgegeben. In Abbildung 4.2 wird dargelegt, wie MS-DOS die AUTOEXEC.BAT-Datei benutzt.

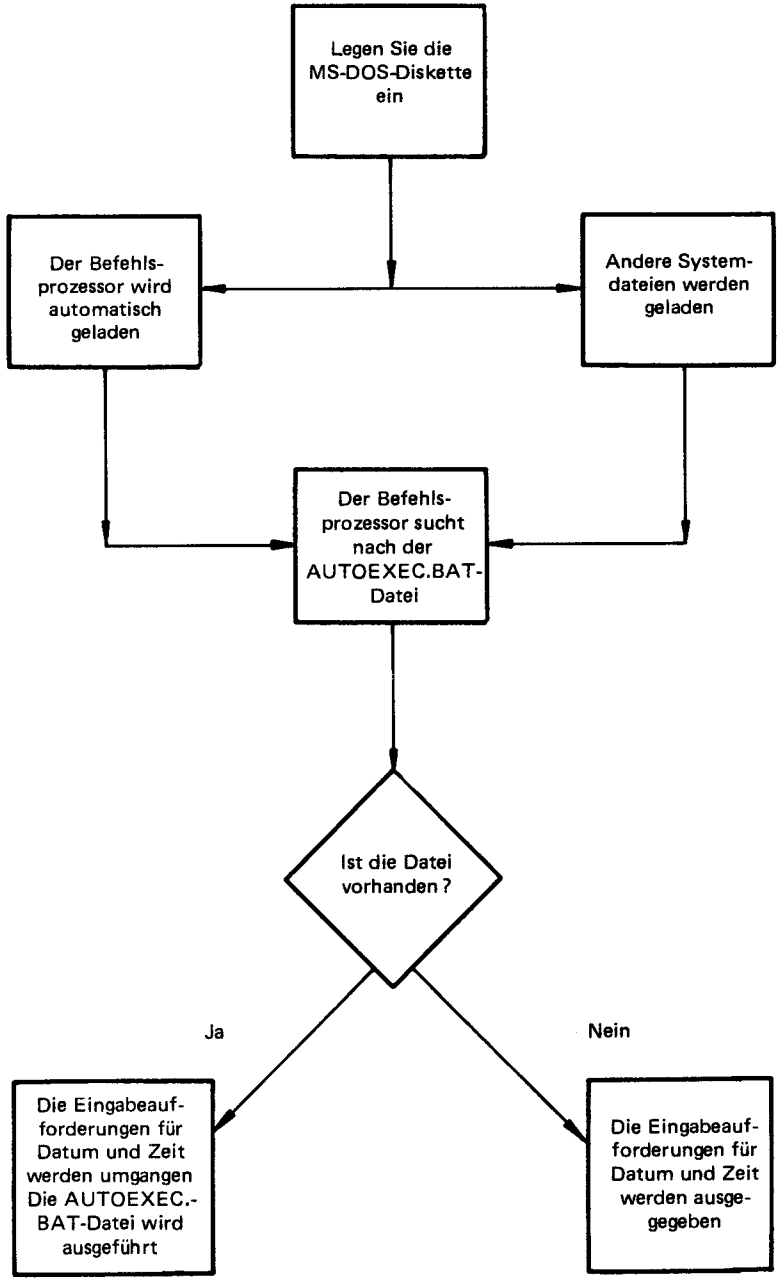


Abbildung 4.2 Benutzung der AUTOEXEC.BAT-Datei durch MS-DOS

ERSTELLEN EINER AUTOEXEC.BAT-DATEI

Für das Erstellen einer AUTOEXEC.BAT-Datei wird hier davon ausgegeangen, daß bei jedem Starten von MS-DOS automatisch BASIC geladen und ein Programm namens MENU ausgeführt werden soll. Die AUTOEXEC.BAT-Datei wird in diesem Fall wie folgt erstellt:

1. Eingabe von:

   COPY CON: AUTOEXEC.BAT

   Aufgrund dieser Anweisung kopiert MS-DOS die Information von der Konsole (Tastatur) in die AUTOEXEC.BAT-Datei. Die AUTOEXEC.BAT-Datei ist im Stammverzeichnis der MS-DOS Platte zu erstellen.
2. Nun wird:

   BASIC MENU

   eingegeben. Diese Anweisung geht in die AUTOEXEC.BAT-Datei. Mit ihr wird MS-DOS angewiesen, bei jedem Starten von MS-DOS BASIC zu laden und das MENU-Programm auszuführen.
3. Betätigen Sie die <CONTROL-Z> Taste. Danach wird die RETURN-Taste betätigt, um den Befehl BASIC MENU in die AUTOEXEC.BAT-Datei zu setzen.
4. Das MENU-Programm wird nun automatisch bei jedem Starten von MS-DOS ausgeführt.

Zur Ausführung eines BASIC-Programms des Benutzers, wird der Name dieses Programms anstelle von MENU in der zweiten Zeile des Beispiels angegeben. In die AUTOEXEC.BAT-Datei kann jeder beliebige MS-DOS Befehl oder jede beliebige Folge von Befehlen eingegeben werden.

HINWEIS: Bei Benutzung einer AUTOEXEC.BAT-Datei gibt MS-DOS keine Eingabeaufforderung für Datum und Zeit aus, es sei denn, die Befehle DATE und TIME sind in der AUTOEXEC. BAT-Datei enthalten. Diese beiden Befehle sollten in der AUTOEXEC.BAT-Datei aufgenommen werden, da MS-DOS diese Information zur Aktualisierung des Inhaltsverzeichnisses benötigt.

ERSTELLEN EINER .BAT-DATEI MIT AUSTAUSCHBAREN PARAMETERN

Gelegentlich möchte man ein Anwendungsprogramm erstellen und mit verschiedenen Daten ausführen. Diese Daten lassen sich in verschiedenen MS-DOS Dateien speichern.

In MS-DOS Befehlen ist ein Parameter eine vom Benutzer definierte Option. In MS-DOS kann eine Batch-Datei (.BAT) mit Pseudoparametern (austauschbaren Parametern) erstellt werden. Diese Parameter mit der Bezeichnung %0-%9 können durch Werte ersetzt werden, die bei der Ausführung der Batch-Datei geliefert werden.

Wird beispielsweise die Befehlszeile COPY CON: MYFILE.BAT eingegeben, so werden die nachfolgend eingegebenen Zeilen von der Konsole in eine Datei namens MYFILE.BAT im Standardlaufwerk kopiert:

```
A>COPY CON MYFILE.BAT
COPY %1.MAC %2.MAC
TYPE %2.PRN
TYPE %0.BAT
```

Nun wird <CONTROL-Z> und danach <RETURN> betätigt. MS-DOS antwortet mit folgender Meldung:

```
1 File(s) copied
A>
(1 Datei(en) kopiert
A>)
```

Die Datei MYFILE.BAT, die aus drei Befehlen besteht, steht nun auf der Platte im Standardlaufwerk.

Die Pseudoparameter %1 und %2 werden sequentiell durch die Parameter ersetzt, die der Benutzer bei der Ausführung der Datei liefert. Der Pseudoparameter %0 wird immer durch die Laufwerkbezeichnung, sofern vorhanden, und den Dateinamen der Batch-Datei (z.B. MYFILE) ersetzt.

HINWEIS: Bis zu zehn Pseudoparameter (%0 bis %9) können angegeben werden. Sollen mehr als zehn Parameter angegeben werden, so sei auf den MS-DOS Befehl SHIFT in Kapitel 5 verwiesen. Wird das Prozentzeichen als Teil eines Dateinamens

innerhalb einer Batch-Datei benutzt, so muß es zweimal eingegeben werden. So ist beispielsweise die Datei ACB%.EXT als ABC%%.EXE in der Batch-Datei anzugeben.

AUSFÜHREN EINER .BAT-DATEI

Zur Ausführung der Batch-Datei MYFILE.BAT und zur Angabe der Parameter, mit denen die Pseudoparameter ersetzt werden, muß der Name der Batch-Datei (ohne die Erweiterung) gefolgt von den Parametern eingegeben werden, die MS-DOS an die Stelle der Parameter %1, %2 usw. setzen soll.

Hier verweisen wir nochmals darauf, daß die Datei MYFILE.BAT aus drei Zeilen besteht:

```
COPY %1.MAC %2.MAC
TYPE %2.PRN
TYPE %0.BAT
```

Zur Ausführung des MYFILE Batch-Prozesses ist:

```
MYFILE A:PROG1  B:PROG2
```

einzugeben. MYFILE tritt an die Stelle von %0, A:PROG1 tritt an die Stelle von %1 und B:PROG2 an die Stelle von %2.

Das Ergebnis ist dasselbe, als wäre jeder der Befehle in MYFILE mit den entsprechenden Parametern wie folgt eingegeben worden:

```
COPY A:PROG1.MAC B:PROG2.MAC
TYPE B: PROG2.PRN
TYPE MYFILE.BAT
```

Die folgende Tabelle zeigt wie MS-DOS jeden der obigen Parameter ersetzt:

BATCH FILENAME	PARAMETER1 (%0) (MYFILE)	PARAMETER2 (%1) (PROG1)	PARAMETER3 (%2) (PROG2)
MYFILE	MYFILE.BAT	PROG1.MAC	PROG2.MAC PROG2.PRN

Erinnern wir uns nochmals, daß der Pseudoparameter %0 immer durch die Laufwerkbezeichnung (sofern vorhanden) und den Dateinamen der Batch-Datei zu ersetzen ist.

MS-DOS geht immer davon aus, daß die Eingabe von der Tastatur stammt und daß die Ausgabe an den Bildschirm geht. Der Fluß der Befehlseingabe und -ausgabe kann jedoch umgeleitet werden. Die Eingabe kann von einer Datei und nicht von einer Terminaltastatur stammen. Gleichermaßen kann die Ausgabe an eine Datei oder einen Zeilendrucker und nicht an das Terminal gehen. Darüberhinaus kann eine sogenannte "Übergabe" vorgesehen werden, mit der die Ausgabe von einem Befehl zur Eingabe für einen anderen Befehl wird. Dieses Umleiten und die Übergabe sind in den folgenden Abschnitten beschrieben:

UMLEITEN DER AUSGABE

Bei den meisten Befehlen geht die Ausgabe an das Terminal. Durch Benutzung eines Grösser-als-Zeichens (>) im Befehl kann diese Information jedoch auch an eine Datei gesendet werden. So zeigt der Befehl

```
DIR
```

beispielsweise eine Verzeichnisauflistung der Platte im Standardlaufwerk auf dem Terminalbildschirm an. Derselbe Befehl kann diese Ausgabe an eine Datei namens MYFILES senden, indem die Ausgabedatei in der Befehlszeile angegeben wird:

```
DIR >MYFILES
```

Ist die Datei MYFILES noch nicht vorhanden, so erstellt MS-DOS diese Datei und speichert die Verzeichnisauflistung in der Datei. Ist MYFILES schon vorhanden, so überschreibt MS-DOS die Daten in der Datei mit den neuen Daten.

Soll das Inhaltsverzeichnis oder eine Datei an eine andere Datei angehängt werden (anstatt die gesamte Datei zu ersetzen), so können zwei Grösser-als-Zeichen (> >) benutzt werden. Mit ihnen wird MS-DOS angewiesen, die Befehlsausgabe (wie beispielsweise die Verzeichnisauflistung) an das Ende der angegebenen Datei anzuhängen. Durch den Befehl:

```
DIR > >MYFILES
```

wird die Verzeichnisauflistung an eine bestehende Datei MYFILES angehängt. Ist MYFILES nicht vorhanden, so wird sie erstellt.

Soll die Eingabe für einen Befehl von einer Datei und nicht vom Terminal stammen, so wird das Kleiner-als-Zeichen (<) in diesem Befehl benutzt. Durch den Befehl:

```
SORT <NAMES> LIST1
```

wird die Datei NAMES sortiert und die sortierte Ausgabe an eine Datei namens LIST1 gesendet.

FILTER

Ein Filter ist ein Befehl, der die Eingabe liest, sie auf irgendeine Art und Weise umwandelt und sie dann normalerweise an das Terminal oder an eine Datei ausgibt. Hier sagt man, daß die Daten von dem Programm "gefiltert" wurden. Da Filter auf viele verschiedene Arten miteinander verbunden werden können, können wenige Filter den Platz einer grossen Anzahl von Befehlen einnehmen.

Die MS-DOS Filter umfassen die Befehle CIPHER, FIND, MORE und SORT. Sie führen die folgenden Funktionen aus:

CIPHER
: Verschlüsselt/entschlüsselt eine Datei.

FIND
: Sucht nach einer konstanten Textfolge in einer Datei.

MORE
: Nimmt die Standardterminalausgabe und zeigt sie mit jeweils einer Bildschirmanzeige an.

SORT
: Sortiert Text.

Die Benutzung dieser Filter wird im nächsten Abschnitt behandelt.

BEFEHLSÜBERGABE

Soll mehr als ein Befehl gleichzeitig eingegeben werden, so können die Befehle an MS-DOS "übergeben" werden. Gelegentlich mag es erforderlich sein, die Ausgabe eines Programms als Eingabe für ein anderes Programm zu benutzen. Ein typisches Beispiel hierfür ist ein Programm mit Ausgabe in Spalten. Diese Spalten sind zu sortieren.

Bei der Befehlsübergabe werden die Befehle mit einem Trennzeichen voneinander getrennt. Hier handelt es sich um den senkrechten Trennstrich (|). Durch den Befehl:

DIR | SORT

erhält der Benutzer eine alphabetisch sortierte Auflistung des Inhaltsverzeichnisses. Durch den senkrechten Trennstrich wird die links vom Trennstrich generierte Ausgabe an die Seite rechts vom Trennstrich zu Verarbeitung übergeben.

Diese Übergabe ist auch zu benutzen, wenn eine Ausgabe an eine Datei vorgenommen werden soll. Soll das Inhaltsverzeichnis sortiert und an eine neue Datei gesendet werden (beispielsweise an DIREC.FIL), so wird:

DIR | SORT >DIREC.FIL

eingegeben.

MS-DOS erstellt eine Datei namens DIREC.FIL im Standardlaufwerk des Benutzers. DIREC.FIL enthält eine sortierte Auflistung des Inhaltsverzeichnisses im Standardlaufwerk, da kein anderes Laufwerk im Befehl angegeben wurde. Zur Angabe eines anderen Laufwerks wird:

DIR | SORT > B:DIREC.FIL

eingegeben. Dadurch werden die sortierten Daten an eine Datei namens DIREC.FIL in Laufwerk B gesendet.

Eine derartige Übergabefolge kann auch aus mehr als zwei Befehlen bestehen. Durch:

DIR | SORT | MORE

wird beispielsweise das Inhaltsverzeichnis sortiert und mit einer Bildschirmanzeige nach der anderen auf dem Bildschirm dargestellt. Außerdem wird "- - MORE - -" an das untere Ende des Bildschirms gesetzt, wenn noch eine weitere Ausgabe folgt.

Befehlsübergabe und Filter kommen in diesem Handbuch sehr häufig vor. Weitere Informationen über die Benutzung von Filtern finden Sie im nächsten Kapitel, MS-DOS Befehle.

MS-DOS BEFEHLE

In diesem Kapitel wird jeder der MS-DOS-Befehle beschrieben. Die Befehle sind in alphabetischer Reihenfolge geordnet, um das Nachschlagen zu vereinfachen. Bestimmte Befehle werden nur beim Schreiben von Batch-Programmen benutzt. Diese Befehle, ECHO, FOR, GOTO, IF und SHIFT, sind in der Beschreibung als Befehle für die Batch-Verarbeitung gekennzeichnet. Vor der Beschreibung der einzelnen Befehle steht eine Tabelle, in welcher der gesamte Befehlsvorrat zusammengefaßt wird.

Vor dem Lesen dieses Kapitels sollte sich der Benutzer sicher sein, daß er mit der Schreibweise für das Format der Befehle vertraut ist. (Diese Schreibweise, die bereits in einem vorhergehenden Kapitel beschrieben wurde, wird nachstehend noch einmal erläutert.)

- In Großbuchstaben angegebene Wörter müssen unbedingt eingegeben werden. Diese Wörter werden als Schlüsselwörter bezeichnet. Sie sind genau wie dargestellt einzugeben. Sie können diese Schlüsselwörter in einer beliebigen Kombination von Groß- und Kleinbuchstaben eingeben. MS-DOS wandelt alle Schlüsselwörter in Großbuchstaben um.
- Für in spitzen Klammern (< >) stehende Elemente liefert der Benutzer den Text. So muß der Benutzer beispielsweise den Namen seiner Datei eingeben, wenn in dem Format <Dateiname> enthalten ist.
- Elemente in eckigen Klammern ([ ]) sind wahlweise. Wenn Sie wahlweise Informationen eingeben, dürfen Sie die eckigen Klammern nicht angeben, sondern nur die Information innerhalb der eckigen Klammern.
- Die Auslassungszeichen (. . .) geben an, daß der Benutzer ein Element beliebig oft wiederholen kann.
- Sämtliche Satzzeichen wie Kommas, Gleichheitszeichen, Fragezeichen, Doppelpunkte oder Schrägstriche müssen wie angegeben übernommen werden. (Dies gilt nicht für die eckigen Klammern.)

ZUSAMMENFASSUNG DER MS-DOS-BEFEHLE

Name (Synonym)	Zweck	Syntax
BACKUP	Kopiert Festplatten auf Disketten	BACKUP
BREAK	Bestimmt, ob die Überprüfung auf CONTROL-C aktiviert ist.	BREAK ON BREAK OFF
CHDIR (CD)	Ändert Inhaltsverzeichnisse; druckt Arbeitsverzeichnisse aus	CHDIR [Pfadname]
CHKDSK	Sucht das Inhaltsverzeichnis des Standardlaufwerks oder des angegebenen Laufwerks ab und prüft auf Konsistenz	CHKDSK [d:] <Dateispez> [/F] [/V]
CIPHER	Verschlüsselt/entschlüsselt eine Datei	CIPHER <Schlüsselwort> [<Dateiname>]
CLS	Löscht den Bildschirminhalt	CLS
CONFIG	Definiert Konfigurationsinformationen	CONFIG
COPY	Kopiert die angegebene(n) Datei(en)	COPY <Dateispez> [Dateispez] [Pfadname] [Pfadname] [/V]
CTTY	Ändert die Konsoleinheit	CTTY \ DEV \ DEV
DATE	Zeigt das Datum an und legt es fest	DATE [<mm>-<dd>-<yy>]
DEL (ERASE)	Löscht die angegebene(n) Datei(en)	DEL [Dateispez] [Pfadname]
DIR	Listet die angeforderten Verzeichniseinträge auf	DIR [Dateispez] [Pfadname] [/P] [/W]
DISKCOPY	Kopiert Disketten	DISKCOPY [d:] [d:]
ECHO	Schaltet die Echoeinrichtung für die Batch-Datei ein/aus	ECHO [ON-Meldung] ECHO [OFF-Meldung]
EXIT	Beendet den Befehl und kehrt zur aufrufenden Ebene zurück	EXIT
FIND	Sucht nach einer konstanten Textfolge	FIND [/V/C/N] <Zeichenfolge> [<Dateiname. . .>]

ZUSAMMENFASSUNG DER MS-DOS-BEFEHLE (Fortsetzung)

Name (Synonym)	Zweck	Syntax
FOR	Erweiterung für einen Batch-Befehl	Für die Batch-Verarbeitung: FOR %%<c> IN <Gruppe> DO <Befehl> Bei der interaktiven Verarbeitung: FOR % <c> IN <Gruppe> DO <Befehl>
FORMAT	Formatiert eine Diskette für die Aufnahme der MS-DOS-Datei • Festplatte • Diskette	 FORMAT [d] : [/V] FORMAT [d:] [/V/J/D/I/O/S]
GOTO	Erweiterung für einen Batch-Befehl	GOTO <Kennzeichen>
IF	Erweiterung für einen Batch-Befehl	IF <Bedingung> <Befehl>
LOCATE	Wandelt ausführbare Dateien in das Binärformat um	LOCATE <Dateispez> [d:] [<Dateiname> [<.erw>]]
MKDIR (MD)	Erstellt ein Inhaltsverzeichnis	MKDIR <Pfadname>
MORE	Zeigt einen Bildschirminhalt nach dem anderen an	MORE
PATH	Setzt einen Suchpfad für den Befehl fest	PATH [<Pfadname>[;<Pfadname>] . . .]
PAUSE	Wartet auf die Eingabe für eine Batch-Datei	PAUSE [Kommentar]
PRINT	Einrichtung für das Drucken im Hintergrundbetrieb	PRINT [Dateispez] [/T] [/C] [/P]]. . .
PROMPT	Legt die Zeichenfolge für die Eingabeaufforderung für einen Befehl fest	PROMPT [<Text der Eingabeaufforderung>]
RECOVER	Stellt eine fehlerhafte Diskette wieder her	RECOVER <Dateiname> RECOVER <d:>
REM	Zeigt einen Kommentar in einer Batch-Datei an	REM [Kommentar]

ZUSAMMENFASSUNG DER MS-DOS-BEFEHLE (Fortsetzung)

Name (Synonym)	Zweck	Syntax
REN (RENAME)	Benennt die erste Datei als zweite Datei um	REN <Dateispez><Dateiname>
RDCPM	Überträgt CP/M-Dateien auf eine für MS-DOS formatierte Diskette	RDCPM DIR d: RDCPM d: Dateiename [d:]
RMDIR (RD)	Löscht ein Inhaltsverzeichnis	RMDIR [d:] <Pfadname>
SET	Setzt einen Zeichenfolgenwert auf einen anderen fest	SET [<Zeichenfolge = Zeichenfolge>]
SHIFT	Erhöht die Anzahl von austauschbaren Parametern in einem Batch-Verfahren.	SHIFT
SORT	Sortiert die Daten alphabetisch, wobei vorwärts oder rückwärts sortiert werden kann	SORT [/R] [/+n]
SYS	Überträgt MS-DOS Systemdateien von Laufwerk A: auf das angegebene Laufwerk	SYS <d>:
TIME	Zeigt die Uhrzeit an und legt sie fest	TIME [<hh>[:<mm>]]
TYPE	Zeigt den Inhalt der angebenen Datei an	TYPE <Dateispez>
VER	Druckt die MS-DOS-Versionsnummer aus	VER
VERIFY	Bestimmt, ob das Schreiben auf Diskette geprüft wird	VERIFY [ON] VERIFY [OFF]
VOL	Druckt die Nummer des Datenträgerkennsatzes aus	VOL [d:]

BACKUP

NAME

BACKUP

TYP

Extern

ZWECK

Kopiert den Inhalt einer der logischen Festplatten im Quellenlaufwerk auf Disketten; stellt die Festplatte auch wieder her.

SYNTAX

BACKUP

KOMMENTARE

Vor dem Kopieren fragt BACKUP nach den Bezeichnungen für das Quellen- und Bestimmungslaufwerk, sowie dem aus sechs Zeichen bestehenden Datenträgerkennsatz, der auf jede Diskette gesetzt wird. Außerdem fragt BACKUP, ob ein Schreiben mit Überprüfen ausgeführt werden soll.

Nach Beantwortung der Fragen muß der Benutzer eine formatierte Diskette in das Bestimmungslaufwerk einlegen und <CR> betätigen, um das Kopieren zu starten. (Die Disketten müssen bereits ohne Schalter formatiert sein.) Sobald eine Diskette gefüllt ist, fordert BACKUP den Benutzer auf, die nächste Diskette einzulegen.

HINWEIS: BACKUP kopiert den gesamten Inhalt einer logischen Festplatte. Wenn Sie nur bestimmte, ausgewählte Dateien kopieren wollen, müssen Sie den COPY-Befehl benutzen.

Soll eine Festplatte wiederhergestellt werden, so wird das Kopierverfahren einfach umgekehrt: Das Diskettenlaufwerk ist als Quellenlaufwerk und das Festplattenlaufwerk als Bestimmungslaufwerk anzugeben. Stimmt der vom Benutzer eingegebene Kennsatz nicht mit dem Kennsatz auf der Diskette überein, oder legt der Benutzer eine Diskette ein, die der Reihenfolge nicht entspricht, so werden Fehlermeldungen angezeigt.

BREAK

NAME

BREAK

TYP

Intern

ZWECK

Legt die Überprüfung auf CONTROL-C fest.

SYNTAX

BREAK ON
BREAK OFF

KOMMENTARE

Während der Ausführung eines Anwenderprogramms das CONTROL-C Funktionstasten benutzt, darf die MS-DOS-Funktion CONTROL-C nicht aktiv sein, damit bei Betätigung von <CONTROL-C> das Programm und nicht das Betriebssystem reagiert. Für das Ausschalten von CONTROL-C wird BREAK OFF angegeben. Nachdem das Anwenderprogramm beendet ist und MS-DOS wieder benutzt wird, sollten Sie BREAK ON angeben.

CHDIR

NAME

CHDIR (CHANGE DIRECTORY)

TYP

Intern

SYNONYM

CD

ZWECK

Ändert das Inhaltsverzeichnis auf einen anderen Pfad; zeigt das gegenwärtige Inhaltsverzeichnis (Arbeitsverzeichnis) an.

SYNTAX

CHDIR [Pfadname]

KOMMENTARE

Ist das Arbeitsverzeichnis \BIN\USER\JOE und soll der Pfad auf ein anderes Inhaltsverzeichnis zeigen (wie beispielsweise \BIN\USER\JOE\FORMS) so wird

CHDIR \BIN\USER\JOE\FORMS

eingegeben. MS-DOS setzt den Benutzer dann in das neue Inhaltsverzeichnis. Für diesen Befehl gibt es auch eine Abkürzung:

CHDIR . . .

Mit diesem Befehl wird der Benutzer immer in das übergeordnete Inhaltsverzeichnis des Arbeitsverzeichnisses gesetzt.

Wird CHDIR ohne Pfadnamen benutzt, so wird das Arbeitsverzeichnis angezeigt. Ist das Arbeitsverzeichnis \BIN\USER\JOE in Laufwerk B, und gibt der Benutzer CHDIR <CR> ein, so zeigt MS-DOS folgendes an:

B: \BIN\USER\JOE

Dieser Befehl ist besonders dann nützlich, wenn der Benutzer den Namen des Arbeitsverzeichnisses vergessen hat.

CHKDSK

NAME

CHKDSK (CHECK DISK)

TYP

Extern

ZWECK

Sucht das Inhaltsverzeichnis des angegebenen Plattenlaufwerks ab und überprüft es auf Konsistenz.

SYNTAX

CHKDSK [d:] <Dateispez> [/F] [/V]

KOMMENTARE

Sie sollten CHKDSK gelegentlich für jede Diskette ausführen, um sie auf Fehler im Inhaltsverzeichnis zu überprüfen. Werden Fehler gefunden, so zeigt CHKDSK Fehlermeldungen an. Danach erscheint am Bildschirm eine Statusliste entsprechend der nachfolgend dargestellten Liste.

```
160256  bytes total disk space
        (Gesamtplatz auf der Diskette in Bytes)
  8192  bytes in 2 hidden files
        (Bytes in 2 versteckten Dateien)
   512  bytes in 2 directories
        (Bytes in 2 Inhaltsverzeichnissen)
 30720  bytes in 8 user files
        (Bytes in 8 Benutzerdateien)
121344  bytes available on disk
        (auf Diskette verfügbare Bytes)

 65536  bytes total memory
        (Gesamtspeicherplatz in Bytes)
 53152  bytes free
        (unbenutzte Bytes)
```

CHKDSK korrigiert im Inhaltsverzeichnis gefundene Fehler nicht, es sei denn der Benutzer gibt den Schalter /F (Fehler be-

heben) an. Wird /V eingegeben, so zeigt CHKDSK während der Ausführung Meldungen an.

Die Ausgabe von CHKDSK kann zu einer Datei umgeleitet werden. Zu diesem Zweck geben Sie einfach

CHKDSK A:>Dateiname

ein. Die Fehler werden zu der Datei mit dem angegebenen Namen gesendet. Wenn Sie die Ausgabe von CHKDSK leiten, dürfen Sie den Schalter /F nicht benutzen.

Die folgenden Fehler werden automatisch korrigiert, wenn Sie den Schalter /F angegeben haben:

Invalid drive specification
(Ungültige Laufwerkbezeichnung)

Invalid parameter
(Ungültiger Parameter)

Invalid sub-directory entry
(Ungültiger Eintrag in einem Unterverzeichnis)

Cannot CHDIR to <filename>
Tree past this point not processed
(CHDIR zu <Dateiname> nicht möglich.
Baum nach diesem Punkt nicht verarbeitet)

First cluster number is invalid
entry truncated
(Erste Gruppennummer ist ungültig:
Eintrag abgeschnitten)

Allocation error, size adjusted
(Zuordnungsfehler, Größe angepaßt)

Has invalid cluster, file truncated
(Ungültige Gruppe: Datei abgeschnitten)

Disk error reading FAT
(Plattenfehler beim Lesen der FAT)

Disk error writing FAT
(Plattenfehler beim Schreiben der FAT)

<filename> contains
non-contiguous blocks
(<Dateiname> enthält
nicht fortlaufende Blöcke)

All specified file(s) are contiguous
(Alle angegebenen Dateien sind fortlaufend)

Die folgenden von CHKDSK angegebenen Fehler müssen vom Benutzer korrigiert werden, selbst wenn er den Schalter /F angegeben hat:

Incorrect DOS version
(Falsche DOS-Version)

CHKDSK kann nur mit der MS-DOS-Version 2.0 oder höheren Versionen benutzt werden.

Insufficient memory
Processing cannot continue
(Nicht ausreichender Speicherplatz:
Verarbeitung kann nicht fortgesetzt werden)

Der vorhandene Speicherplatz ist zur Verarbeitung von CHKDSK für diese Diskette nicht ausreichend. Zur Ausführung von CHKDSK muß mehr Speicherplatz zur Verfügung stehen.

Errors found, F parameter not specified
Corrections will not be written to disk
(Fehler gefunden, aber F-Parameter nicht angegeben:
Korrekturen werden nicht auf Diskette geschrieben)

Sie müssen den Schalter /F angeben, wenn die Fehler von CHKDSK korrigiert werden sollen.

Invalid current directory
Processing cannot continue
(Ungültiges gegenwärtiges Inhaltsverzeichnis:
Verarbeitung kann nicht fortgesetzt werden)

Das System erneut starten und CHKDSK erneut ausführen.

Cannot CHDIR to root
Processing cannot continue
(CHDIR zum Stammverzeichnis nicht möglich:
Verarbeitung kann nicht fortgesetzt werden)

Die gerade überprüfte Diskette ist fehlerhaft. Starten Sie MS-DOS erneut und versuchen Sie die Diskette mit RECOVER wiederherzustellen.

<filename> is cross linked on cluster
(Querverknüpfung bei <Dateiname> in Gruppe)

Erstellen Sie eine Kopie der zu behaltenden Datei und löschen Sie die beiden querverknüpften Dateien.

X lost clusters found in y chains
Convert lost chains to file (Y/N)?
(X verlorene Gruppen in y Ketten gefunden:
Verlorene Ketten in Datei umwandeln (J/N)?

Wird diese Meldung mit Y beantwortet, so erstellt CHKDSK einen Verzeichniseintrag und eine Datei, um dieses Problem zu lösen (die von CHKDSK erstellten Dateien tragen den Namen FILEnnnnnnnn).

CHKDSK zeigt dann

X bytes disk space freed
(X Bytes Plattenspeicherplatz freigegeben)

an.

Beantwortet der Benutzer diese Meldung mit N und hat er den Schalter /F nicht angegeben, so gibt CHKDSK die Gruppen frei und zeigt folgende Meldung an:

X bytes disk space would be freed
(Verzicht auf x Bytes Plattenspeicherplatz)

Probable non-DOS disk
Continue (Y/N)?
(Wahrscheinlich Nicht-DOS-Diskette:
Fortfahren (J/N)?)

Bei der benutzten Diskette handelt es sich nicht um eine DOS-Diskette. Der Benutzer muß nun angeben, ob CHKDSK die Verarbeitung fortsetzen soll oder nicht.

Insufficient room in root directory
Erase files in root and repeat CHKDSK
(Nicht ausreichend Platz in Stammverzeichnis:
Dateien in Stammverzeichnis löschen und CHKDSK wiederholen)

CHKDSK kann die Verarbeitung erst aufnehmen, nachdem der Benutzer Dateien im Stammverzeichnis gelöscht hat.

Unrecoverable error in directory
Convert directory to file (Y/N)?
(Nicht behebbarer Fehler im Inhaltsverzeichnis:
Inhaltsverzeichnis in Datei umwandeln (J/N)?)

Wird diese Meldung mit Y beantwortet, so wandelt CHKDSK das fehlerhafte Inhaltsverzeichnis in eine Datei um. Sie können dann das Inhaltsverzeichnis korrigieren oder löschen.

CIPHER

NAME

CIPHER

TYP

Extern

ZWECK

Verschlüsselt und entschlüsselt Dateien entsprechend einem bestimmten Schlüsselwort.

SYNTAX

CIPHER <Schlüsselwort> [<Dateiname>]

KOMMENTARE

Sie können diesen Befehl benutzen, um eine Datei aus Sicherheitsgründen zu verschlüsseln. Der CIPHER-Befehl benutzt ein Schlüsselwort, das beim Verschlüsseln der Datei angegeben werden muß. Um die Datei NSA.CIA mit dem Schlüsselwort "SECRET" zu verschlüsseln, geben Sie

```
CIPHER SECRET < NSA.CIA
```

ein. Dadurch wird die verschlüsselte Datei (NSA.CIA) am Bildschirm angezeigt. Soll die verschlüsselte Datei zu einer anderen Datei gesendet werden, so wird

```
CIPHER SECRET <NSA.CIA >MYSTERY.NEW
```

eingegeben. MYSTERY.NEW ist der Name der Datei, in die die verschlüsselte Datei gespeichert wird. Sie können die Originaldatei NSA.CIA löschen.

Um die verschlüsselte Datei MYSTERY.NEW zu entschlüsseln, wird das Verfahren einfach umgekehrt:

```
CIPHER SECRET <MYSTERY.NEW
```

Dieser Befehl entschlüsselt die Datei MYSTERY.NEW und zeigt sie am Bildschirm an. Wenn Sie die entschlüsselte Datei zu einer

anderen Datei namens NOSECRET.XXX senden wollen, geben Sie

```
CIPHER SECRET <MYSTERY.NEW >NOSECRET.XXX
```

ein.

HINWEIS: Beim Entschlüsseln der Datei ist dasselbe Schlüsselwort wie beim Verschlüsseln der Datei anzugeben. Ansonsten wird der CIPHER-Befehl nicht ausgeführt. Wenn Sie das <-Zeichen nicht angeben, wird die Eingabe der Tastatur zugeordnet; die Ausgabe erfolgt am Bildschirm. Der Dateiname wird dann nicht beachtet. Wenn das >-Zeichen fehlen sollte, wird die verschlüsselte Datei nicht auf eine andere Datei übertragen. Stattdessen erfolgt die Ausgabe am Bildschirm. Sie können die Verschlüsselung bzw. Entschlüsselung durch Drücken von CONTROL-Z oder CONTROL-C abbrechen.

CLS

NAME

CLS

TYP

Intern

ZWECK

Löscht den Terminalbildschirm.

SYNTAX

CLS

KOMMENTARE

Durch den CLS-Befehl sendet MS-DOS die ANSI-Umschaltfolge ESC [2J (die den Bildschirm löscht) zu der Konsole.

CONFIG

NAME

CONFIG

TYP

Extern

ZWECK

Definiert und ändert Konfigurationsinformationen (vorübergehend oder permanent) für MS-DOS.

SYNTAX

CONFIG

KOMMENTARE

Mit diesem Befehl wird die Verarbeitungsumgebung für MS-DOS definiert: Typ des Druckers und der Diskette, eventuelle programmierbare Funktionstasten und Anzahl der auszuführende Versuche bei Plattenlese- und Plattenschreibvorgängen.

MS-DOS wird ursprünglich mit bestimmten Parametern in Betrieb genommen. In der nachfolgenden Tabelle werden diese Parameter dargestellt. Außerdem werden die Änderungen angeführt, die der Benutzer mit CONFIG durchführen kann.

	Ursprüngliche Definition	Mit CONFIG
Programmierbare Funktionstasten	keine	bis zu 20
Drucker	Paralleldrucker	Serieller Drucker
Serielle Druckerschnittstelle:		
– Stopbits	1	1 1/2 oder 2
– Parität	gerade	deaktiviert oder ungerade
– Zeichenlänge	7 Bits	5, 6, oder 8
– Baudrate	9600	50-19200
Platte	2 Disketten	1 Diskette, 1, 2, oder 3 Festplatten
Rückschreibe- und Wiederholungszähler:		
– Diskette	5,5	1-9, 1-9
– Festplatte	5,5	1-9, 1-9

CONFIG besteht aus einer Reihe von Bedienerführungen am Bildschirm. Nachdem Sie CONFIG eingegeben haben, erscheint das Hauptmenü.

```
CONFIG
      . . .

                CONFIG UTILITY

1)  Modify Function Keys
2)  Select Printer (Serial/Parallel)
3)  Modify Retry/Restore Counter
4)  Modify Serial Printer Interface
5)  Modify Disk Configuration
6)  Exit Program

*   Enter function

1.) Funktionstasten ändern
2.) Drucker wählen (seriell/parallel)
3.) Rückschreibe- und Wiederholungszähler ändern
4.) Serielle Druckerschnittstelle ändern
5.) Plattenkonfiguration ändern
6.) Programm beenden

*   Funktion eingeben
```

Nach der Auswahl der Funktion helfen weitere Bedienerführungen dem Benutzer bei der Definition der Konfiguration. Obwohl sich die Bedienerführungen von selbst erklären, müssen einige Konventionen für die Benutzung berücksichtigt werden.

Keine der 20 programmierbaren Funktionstasten ist im voraus definiert. Eine einzige Definition kann eine Länge von cirka 255 Zeichen aufweisen (die genaue Länge hängt von den benutzten Zeichen ab). Mit der Definition kann eine beliebige Funktion angegeben werden. Sie kann jedoch keine andere Funktionstaste umfassen (keines der Zeichen in dem Bereich von 80-FF wird akzeptiert).

Die Definitionen für die Funktionstasten stehen in einer Tabelle, die bis zu 492 Zeichen aufnehmen kann. Wenn Sie mehr Zeichen eingeben, erscheint eine Meldung am Bildschirm. Wenn Sie mit der Verarbeitung fortfahren, wird die Eingabedefinition der Taste, die

den Überlauf verursacht hat, gelöscht. Der Originalinhalt wird jedoch nicht gelöscht.

Die Funktionstasten sollen die Arbeit vereinfachen. So können Sie beispielsweise eine häufig benutzte Funktion einer Funktionstaste zuordnen. Sie kann dann einfach durch Betätigung der Taste eingeleitet werden.

Angenommen die Verarbeitung beginnt normalerweise mit der Anzeige des Inhaltsverzeichnisses auf der Systemplatte. In diesem Fall könnten Sie den Befehl DIR A: (zuzüglich der <CR> Abschlußfunktion) mit CONFIG der Funktionstaste 1 zuordnen. Zu diesem Zweck geben Sie Funktion 1 aus dem Hauptmenü und Funktion 2 aus dem Funktionstastenmenü an. Anschließend betätigen Sie F1 als Antwort auf die Meldung "Enter Function Key" (Funktionstaste eingeben). Danach wird folgende Bildschirmanzeige ausgegeben:

```
Function 01:
```

Geben Sie nun den Befehl einschließlich der <CR>-Funktion ein:

```
Function 01: DIR A: <CR>
```

HINWEIS: Die Steuerzeichentasten mit den Hexadezimalwerten von 00 bis 1F werden zwischen den Symbolen < > angezeigt.

Sie können die Definition überprüfen und die Funktionstaste erneut betätigen. (Die Definition beginnt und endet stets mit Betätigung der Funktionstaste.) Die Taste ist nun für die Anzeige des Inhaltsverzeichnisses programmiert. Wenn Sie die Zuordnung überprüfen wollen, fordern Sie die Funktion "display definition" (Definition anzeigen).

Wie schon in Kapitel 2 besprochen, muß CONFIG zur Definition des Plattensystems benutzt werden, es sei denn, zwei Disketten werden benutzt. Änderungen an den Parametern für die Plattenkonfiguration müssen als "permanent" angegeben und auf die Betriebssystemplatte geschrieben werden. Anschließend muß ein Neustart des Systems erfolgen. (Ein Neustart bedeutet einfach Aus- und Wiedereinschalten des Computers.) Durch den Neustart werden die Plattenlaufwerke initialisiert.

Die Funktion Exit Programm (Programm beenden) kann nach Ausführung jeder einzelnen Konfigurationsfunktion oder nach Beendigung aller Funktionen benutzt werden. Wenn Sie diese Funktion anfordern, werden 3 Optionen angezeigt.

1) Update O.S. disk in drive A
2) Return to main program
3) Exit CONFIG

ATTENTION: Changes to the disk configuration must be written to disk (permanent) and must be followed by a restart. Update the disk, exit CONFIG, and then turn off and on the computer when the system prompt appears.

* Enter function

1) Betriebssystemplatte in Laufwerk A gemäß Änderung(en) der Konfiguration aktualisieren
2) Zurück zum Hauptprogramm
3) CONFIG beenden

ACHTUNG: Änderungen an der Plattenkonfiguration müssen (permanent) auf die Platte geschrieben werden. Anschließend muß ein Neustart erfolgen. Die Platte aktualisieren, CONFIG beenden und dann den Computer aus- und wieder einschalten, sobald die Systemmeldung angezeigt wird.

* Funktion eingeben

Mit Funktion 1 werden die neuen Konfigurationsparameter auf die Platte geschrieben. Sind die Änderungen nur temporär (gelten sie beispielsweise nur für einen einmaligen Programmablauf), so ist Funktion 3 zu benutzen. Die Änderungen werden nur im Speicher vorgenommen.

Wenn Sie permanente Änderungen durchführen wollen die auch für alle anderen Kopien der Betriebssystemplatte gelten sollen, legen Sie einfach eine weitere Platte in Laufwerk A ein und fordern Sie Funktion 1 an. Dieses Verfahren kann wiederholt werden, bis sämtliche MS-DOS-Betriebssystemplatten aktualisiert sind.

Bei der Benutzung von CONFIG kommt es nur selten zu Fehlern. Es kann jedoch sein, daß Fehler angezeigt werden, wenn sich die Betriebssystemplatte ihrer Kapazitätsgrenze nähert. Während der letzten Phase einer Funktion zur Änderung der Plattenkonfiguration schreibt CONFIG die permanenten Änderungen in eine Datei namens CONFIG.SYS. Ist das Inhaltsverzeichnis oder der Dateibereich selbst voll, so wird eine der beiden folgenden Meldungen angezeigt:

DIRECTORY FULL
DELETE A FILE FROM YOUR O.S. DISK; THEN REPEAT THIS FUNCTION.
(Inhaltsverzeichnis voll:
Eine Datei von der Betriebssystemplatte löschen; danach diese Funktion wiederholen.)

oder

DISK FULL
DELETE OR SHORTEN A FILE FROM YOUR O.S. DISK; THEN REPEAT THIS FUNCTION.
(Platte voll:
Eine Datei von der Betriebssystemplatte löschen oder kürzen; danach diese Funktion wiederholen.)

Um das Problem zu beheben, müssen Sie einfach eine nicht wichtige Datei (beispielsweise eine Arbeitsdatei oder eine Sicherheitsdatei) löschen oder kürzen; danach fordern Sie CONFIG erneut an und wählen die Funktion Exit Program (Programm beenden) aus. Nun können Sie Funktion 1 angeben. Die Konfigurationsänderungen werden auf die Platte geschrieben.

CONFIG.SYS:
↵
DEVICE = TIMDRIVE.SYS C8 04 P5

Erstellen einer Datei von Konsole aus.

COPY CON: TEST
Text ... CTRL Z

COPY

NAME

COPY

TYP

Intern

ZWECK

Kopiert eine oder mehrere Dateien auf eine andere Diskette. Falls gewünscht, können Sie den Kopien andere Namen zuteilen. Mit diesem Befehl können auch Dateien auf der selben Diskette kopiert werden.

SYNTAX

COPY <Dateispez> [Dateispez] [Pfadname] [Pfadname] [/V]

KOMMENTARE

Bevor Sie diesen Befehl benutzen, müssen Sie sich vergewissern, ob auf der Bestimmungsdiskette ausreichend Platz für die Kopie vorhanden ist.

Wird die zweite Option Dateispez nicht angegeben, so wird die Kopie zum Standardlaufwerk gesendet. Sie erhält den selben Namen wie die Originaldatei (erste Option Dateispez). Ist die erste Dateispez im Standardlaufwerk und wird die zweite Dateispez nicht angegeben, so wird COPY abgebrochen (Dateien können nicht in sich selbst kopiert werden). MS-DOS gibt folgende Fehlermeldung an:

File cannot be copied onto itself
0 File(s) copied
(Datei kann nicht in sich selbst kopiert werden,
0 Datei(en) kopiert)

HINWEIS: Eine Datei auf einer Diskette kann nicht mit Hilfe eines einzigen Diskettenlaufwerks auf eine andere Diskette kopiert werden. Um bestimmte Dateien zu kopieren, müssen Sie die Dateien zuerst auf die Festplatte und dann auf eine andere Diskette kopieren.

Die zweite Option kann drei Formen aufweisen:

1. Handelt es sich bei der zweiten Option nur um eine Laufwerkbezeichnung (d:), so wird die Originaldatei mit dem Originaldateinamen in das angegebenen Laufwerk kopiert.
2. Handelt es sich bei der zweiten Option nur um einen Dateinamen, so wird die Originaldatei in eine Datei mit dem angegebenen Dateinamen in dem Standardlaufwerk kopiert.
3. Handelt es sich bei der zweiten Option um eine volle Dateispezifikation, so wird die Originaldatei in eine Datei mit dem angegebenen Dateinamen im Standardlaufwerk kopiert.

Wenn Sie den Schalter /V benutzen, prüft MS-DOS, ob die auf die Bestimmungsdiskette geschriebenen Sektoren richtig aufgezeichnet sind. Obwohl es bei der Ausführung von COPY nur selten zu Aufzeichnungsfehlern kommt, können Sie sich dadurch vergewissern, daß kritische Daten richtig aufgezeichnet wurden. Durch diese Option wird der COPY-Befehl langsamer ausgeführt, da MS-DOS jeden Eintrag auf der Platte überprüfen muß.

Der COPY-Befehl ermöglicht auch eine Dateiverkettung während des Kopierens. Die Verkettung wird einfach durch Auflisten einer beliebigen Anzahl von Dateien in Form von Optionen bei COPY erreicht, wobei die einzelnen Dateien durch "+" voneinander zu trennen sind.

Zum Beispiel

```
COPY A.XYZ + B.COM + B:C.TXT BIGFILE.CRP
```

Dieser Befehl verkettet die Dateien namens A.XYZ, B.COM und B:C.TXT und setzt sie in die Datei namens BIGFILE.CRP im Standardlaufwerk.

Um verschiedene Dateien mit Hilfe von Dateigruppensymbolen in einer Datei zu kombinieren, könnten Sie

```
COPY*.LST COMBIN.PRN
```

eingeben.

Durch diesen Befehl würden sämtliche Dateien mit einer Dateinamenerweiterung von .LST in einer Datei namens COMBIN.PRN kombiniert.

Im folgenden Beispiel wird jede Datei, die mit *.LST übereinstimmt, mit der entsprechenden .REF-Datei kombiniert. Das Ergebnis ist eine Datei mit dem selben Dateinamen, jedoch mit der Erweiterung .PRN. So wird FILE1.LST mit FILE1.REF in FILE1.PRN kombiniert. Danach wird XYZ.LST mit XYZ.REF in XYZ.PRN kombiniert und so fort.

```
COPY*.LST+*.REF*.PRN
```

Im folgenden COPY-Befehl werden sämtliche Dateien, die mit *.LST übereinstimmen, und dann sämtliche Dateien, die mit *.REF übereinstimmen, in einer Datei namens COMBIN.PRN kombiniert:

```
COPY*.LST+*.REF COMBIN.PRN
```

Es darf kein COPY-Befehl für die Verkettung eingegeben werden, wenn einer der Quellendateinamen dieselbe Erweiterung hat, wie der Bestimmungsdateiname. Der folgende Befehl führt beispielsweise zu einem Fehler, wenn ALL.LST schon vorhanden ist:

```
COPY*.LST ALL.LST
```

Der Fehler wird jedoch erst nach der Änderung von ALL.LST entdeckt. An diesem Punkt wäre die Datei möglicherweise schon zerstört.

COPY vergleicht die Dateinamen der Eingabedatei mit dem Dateinamen der Bestimmungsdatei. Sind die Namen identisch, so wird diese eine Eingabedatei übersprungen und die Fehlermeldung "Content of destination lost before copy" (Inhalt der Bestimmungsdatei vor Copy verloren) wird ausgedruckt. Die weitere Verkettung wird normal ausgeführt. Dies ermöglicht ein "Summieren" von Dateien, wie in diesem Beispiel:

```
COPY ALL.LST+*.LST
```

Dieser Befehl hängt sämtliche *.LST-Dateien, mit Ausname von ALL.LST selbst an ALL.LST an. Dieser Befehl führt zu keiner Fehlermeldung und stellt die richtige Methode dar, mit der Sie Dateien mit dem COPY-Befehl verketten können.

Dateien werden gewöhnlich als ASCII-Dateien kopiert bzw. verkettet. Dies hat zur Folge, daß das System das erste CONTROL-Z-Zeichen als Dateiende betrachtet. Es ist ebenfalls möglich, Binär-

dateien miteinander oder Binär- mit ASCII-Dateien zu verketten. Hierzu müssen Sie von den Schaltern /A (=ASCII; wird beim Fehlen eines Schalters ohnehin angenommen) und /B (= binär) Gebrauch machen. Beispiel:

[/A][/B] <Dateispez>[Dateispez][Pfadname][/A][/B][/V]

Wenn Sie Binärdateien verketten oder kopieren wollen, benutzen Sie den /B-Schalter am Anfang der Kommandozeile. Im nachstehenden Beispiel nimmt COPY aufgrund des /B-Schalters an, daß sowohl MASM.ABC als auch MASM.DEF Binärdateien sind. Ein am Anfang der Kommandozeile eingegebener /A- oder /B-Schalter wird nämlich erst durch einen nachfolgenden /A- bzw. /B-Schalter oder am Zeilenende außer Kraft gesetzt.

```
COPY /B MASM.ABC+MASM.DEF MASM.EXE
```

Bei der Verkettung von ASCII- mit Binärdateien muß der /A- bzw. /B-Schalter unmittelbar nach dem jeweiligen Dateienamen erscheinen. Im folgenden Beispiel wird die Datei BIGFILE.CRP durch die Verkettung der ASCII-Datei A.XYZ, der Binärdatei B.BIN und der ASCII-Datei C.TXT erzeugt. Wie im vorherigen Beispiel wird ein /A- bzw. /B-Schalter erst durch einen nachfolgenden Schalter oder am Zeilenende aufgehoben.

```
COPY A.XYZ+B.BIN/B+B:C.TXT/A BIGFILE.CRP
```

Wenn Sie der Zieldatei /A voranstellen, wird dieser Datei ein CONTROL-Z von MS-DOS hinzugefügt. /B erzeugt kein CONTROL-Z.

HINWEIS: Die von Software-Entwicklungshilfen (z.B. einem Macro-Assembler) erzeugten Binärdateien werden als solche von MS-DOS anerkannt, vorausgesetzt, daß die Dateinamenserweiterung ohne Dateigruppensymbole (? oder *) definiert wurde. /B erübrigt sich bei der Verkettung solcher Dateien.

CTTY

NAME

CTTY

TYP

Intern

ZWECK

Mit diesem Befehl kann die Einheit geändert werden, von der aus Sie Befehle ausgeben können (TTY stellt die Konsole dar).

SYNTAX

CTTY \DEV\DEV

KOMMENTARE

DEV steht für "Einheit." Hier handelt es sich um die Einheit, von der aus Befehle an MS-DOS ausgegeben werden. Sie können diesen Befehl benutzen, um die Einheit zu ändern, mit der gerade gearbeitet wird. Durch den Befehl

CTTY \DEV\AUX

können Sie die gesamte Ein- und Ausgabe von Befehlen von der aktuellen Einheit (auf der Konsole) zu dem AUX-Anschluß, wie beispielsweise einem Drucker leiten. Durch den Befehl

CTTY \DEV\CON

wird die E/A wieder zu der Originaleinheit (hier der Konsole) geleitet. Für eine Aufstellung der gültigen Einheitennamen zur Benutzung mit dem CTTY-Befehl sei auf den Abschnitt "Unzulässige Dateinamen" in Kapitel 3 verwiesen.

NAME

DATE

TYP

Intern

ZWECK

Eingabe oder Änderung des dem System bekannten Datums. Dieses Datum wird im Inhaltsverzeichnis für sämtliche vom Benutzer erstellten oder geänderten Dateien festgehalten.

Das Datum läßt sich vom Terminal oder von einer Batch-Datei aus ändern. (MS-DOS zeigt keine Eingabeaufforderung für das Datum an, wenn Sie eine AUTOEXEC.BAT-Datei benutzen. Deshalb wird empfohlen, in diese Datei einen DATE-Befehl aufzunehmen.

SYNTAX

DATE [<mm>-<dd>-<yy>]

KOMMENTARE

Wenn Sie DATE eingeben, antwortet der DATE-Befehl mit folgender Meldung:

Current date is <mm>-<dd>-<yy>
Enter new date: _
(Gegenwärtiges Datum ist <mm>-<dd>-<yy>
Neues Datum eingeben: _)

Wenn Sie das angezeigte Datum nicht ändern wollen, betätigen Sie einfach <CR>.

Nach dem DATE-Befehl kann auch ein bestimmtes Datum eingegeben werden, wie beispielsweise:

DATE 5-9-83

In diesem Fall müssen Sie die Eingabeaufforderung "neues Datum eingeben:" nicht beantworten.

Sie dürfen das neue Datum nur in Form von Zahlen eingeben. Buchstaben sind nicht zulässig. Für das Datum gibt es folgende Auswahlmöglichkeiten:

<mm> = 1-12
<dd> = 1-31
<yy> = 80-99 oder 1980-2099

Die Einträge für Datum, Monat und Jahr können durch Bindestriche (-) oder durch Schrägstriche (/) voneinander getrennt werden.

Sind die gewählten Möglichkeiten oder Trennzeichen nicht gültig, so zeigt DATE folgende Meldung an:

Invalid date
Enter new date:_
(Ungültiges Datum
Neues Datum eingeben:_)

DATE wartet dann, bis der Benutzer ein gültiges Datum eingibt.

DEL

NAME

DEL (DELETE)

TYP

Intern

SYNONYM

ERASE

ZWECK

Löscht sämtliche Dateien mit der angegebenen Dateispezifikation.

SYNTAX

DEL [Dateispez] [Pfadname]

KOMMENTARE

Lautet die Dateispezifikation *.*, so wird die Systemmeldung "Are you sure?" (Sind Sie sicher?) angezeigt. Wird als Antwort Y oder y eingegeben, so werden sämtliche Dateien wie gefordert gelöscht. Für den DELETE-Befehl können Sie auch ERASE eingeben.

DIR

NAME

DIR A: > B:KATALOG

DIR (DIRECTORY)

TYP

Intern

SYNTAX

DIR [Dateispez] [Pfadname] [/P] [/W]

ZWECK

Listet die Dateien in einem Verzeichnis auf.

KOMMENTARE

Wenn Sie nur DIR eingeben, werden sämtliche Verzeichniseinträge im Standardlaufwerk aufgelistet. Wenn Sie nur die Laufwerkspezifikation angeben (DIR d:), so werden sämtliche Einträge auf der Platte im angegebenen Laufwerk aufgelistet. Wenn Sie nur einen Dateinamen ohne Erweiterung (DIR Dateiname) eingeben, werden sämtliche Dateien mit dem angegebenen Dateinamen auf der Platte im Standardlaufwerk aufgelistet. Bei einer Dateispezifikation (zum Beispiel DIR d:Dateiname.erw) werden sämtliche Dateien mit dem angegebenen Dateinamen auf der Platte im angegebenen Laufwerk aufgelistet. In allen Fällen werden die Dateien mit ihrer Größe in Bytes und mit Zeitpunkt und Datum der letzten Änderung aufgelistet.

Sie dürfen die Dateigruppensymbole ? und * im Dateinamen benutzen. Als Beispiel werden in der nachfolgenden Tabelle äquivalente Befehlsbezeichnungen angegeben.

BEFEHL	ÄQUIVALENT
DIR	DIR *.*
DIR FILENAME	DIR FILENAME.*
DIR.EXT	DIR*.EXT
DIR.	DIR*.

Mit DIR können zwei Schalter angegeben werden. Der Schalter /P wählt den Seitenmodus an. Wenn der Schalter /P gesetzt ist, er-

folgt eine Unterbrechung der Anzeige des Inhaltsverzeichnisses, nachdem der Bildschirm gefüllt ist. Um die Anzeige der Ausgabe wieder fortzusetzen, betätigen Sie eine beliebige Taste.

Mit dem Schalter /W können Sie eine breite Anzeige wählen. Ist /W gesetzt, so werden nur Dateinamen (fünf Dateinamen pro Zeile) ohne weitere Dateiinformationen angezeigt.

DISKCOPY

NAME

DISKCOPY

TYP

Extern

ZWECK

Kopiert den Inhalt der Platte im Quellenlaufwerk auf die Platte im Bestimmungslaufwerk.

SYNTAX

DISKCOPY [d:] [d:]

KOMMENTARE

Als erste Option wird das Quellenlaufwerk angegeben; die zweite Option stellt das Bestimmungslaufwerk dar.

Die Platte in dem Bestimmungslaufwerk muß (mit dem selben Betriebssytem und in demselben Format wie die Quellenplatte) formatiert sein, bevor Sie DISKCOPY benutzen können.

Sie können dieselben Laufwerke oder unterschiedliche Laufwerke angeben. Sind die angegebenen Laufwerke identisch, so wird die Kopieroperation mit einem einzigen Laufwerk ausgeführt. Der Benutzer wird zu gegebener Zeit aufgefordert, die Platten einzulegen. DISKCOPY wartet, bis der Benutzer eine beliebige Taste betätigt, bevor fortgefahren wird.

Nach dem Kopieren gibt DISKCOPY folgende Eingabeaufforderung aus:

```
Copy another (Y/N)?_
(Weitere kopieren (J/N)?_
```

Wird Y betätigt, so benutzt der Kopiervorgang die ursprünglich angegebenen Laufwerke, nachdem der Benutzer gemäß Aufforderung die entsprechende Platte eingelegt hat.

Soll das Kopierverfahren beendet werden, so müssen Sie N betätigen.

Bevor Sie DISKCOPY benutzen, sollten Sie außerdem die folgenden Befehlseigenschaften berücksichtigen:

- Werden beide Optionen weggelassen, so erfolgt eine Kopieroperation mit einem einzigen Laufwerk im Standardlaufwerk
- Wenn Sie die zweite Option weglassen, wird das Standardlaufwerk als Bestimmungslaufwerk benutzt.
- Beide Platten müssen über dieselbe Anzahl von physischen Sektoren verfügen. Diese Sektoren müssen außerdem dieselbe Größe aufweisen.
- Platten, auf denen viele Dateien erstellt und gelöscht wurden, werden fragmentiert, da der Speicherplatz auf der Platte nicht sequentiell zugeordnet wird. Der zuerst gefundene freie Sektor wird als nächster zugeordnet, unabhängig von seiner Position auf der Platte.

  Eine fragmentierte Platte kann zu schlechten Leistungen führen, da es lange Zeit dauert, bis eine Datei gefunden, gelesen oder geschrieben wird. In diesem Fall sollten Sie DISKCOPY benutzten, um die Platte zu kopieren und dabei ein Fragmentieren zu vermeiden.

  Zum Beispiel:

  COPY A:*.* B:

  kopiert sämtliche Dateien von der Platte in Laufwerk A auf die Platte in Laufwerk B.
- DISKCOPY bestimmt automatisch die Anzahl der zu kopierenden Seiten, je nach Quellenlaufwerk und Quellenplatte.
- Werden während der Ausführung von DISKCOPY, Plattenfehler gefunden, so zeigt MS-DOS folgende Meldung an:

  DISK error while reading drive A
  Abort, Ignore, Retry?
  (Plattenfehler beim Lesen von Laufwerk A
  Abbrechen, ignorieren, erneut versuchen?)

Für Informationen über diese Fehlermeldung wird auf Anhang B, Plattenfehler, verwiesen.

ECHO

NAME

ECHO

TYP

Intern; Batch-Verarbeitung

ZWECK

Schaltet die Echoeinrichtung für die Batch-Verarbeitung an und aus.

SYNTAX

ECHO [ON-Meldung]
ECHO [OFF-Meldung]

KOMMENTARE

Normalerweise werden die Befehle in einer Batch-Datei auf der Konsole angezeigt ("wiederholt"), wenn sie vom Befehlsprozessor entdeckt werden. Mit ECHO OFF können Sie diese Einrichtung ausschalten. Mit ECHO ON wird sie wieder eingeschaltet.

Sind weder ON noch OFF angegeben, so wird die aktuelle Einstellung angezeigt.

EXIT

NAME

EXIT

TYP

Intern

ZWECK

Beendet das Programm COMMAND.COM (den Befehlsprozessor) und kehrt gegebenenfalls zu einer vorhergehenden Ebene zurück.

SYNTAX

EXIT

KOMMENTARE

Sie können diesen Befehl während des Ablaufs eines Anwenderprogramms benutzen, wenn der MS-DOS-Befehlsprozessor gestartet und danach wieder zum Benutzerprogramm zurückgekehrt werden soll. Um zum Beispiel ein Inhaltsverzeichnis in Laufwerk B während der Ausführung eines Anwenderprogramms zu überprüfen, muß der Befehlsprozessor durch Eingabe von COMMAND als Antwort auf die Systemmeldung in Form des Standardlaufwerkbuchstabens gestartet werden:

```
A>COMMAND
```

Nun können Sie den DIR-Befehl eingeben. MS-DOS zeigt dann das Inhaltsverzeichnis für das Standardlaufwerk an. Nach der Eingabe von EXIT kehrt der Benutzer zur vorhergehenden Ebene (dem Anwenderprogramm) zurück.

FIND

NAME

FIND

Find "INV-1" B:UNIVERS3.BAS > C:TEST

TYP

Extern

ZWECK

Sucht nach einer bestimmten Textfolge in einer Datei oder in mehreren Dateien.

SYNTAX

FIND [/V /C /N] <Zeichenfolge> [<Dateiname...>]

KOMMENTARE

FIND ist ein Filter, der eine Zeichenfolge und eine Serie von Dateinamen als Optionen nimmt. Er zeigt sämtliche Zeilen, die eine angegebene Zeichenfolge enthalten, aus den in der Befehlszeile angegebenen Dateien an.

Werden keine Dateien angegeben, so bearbeitet FIND den Bildschirminhalt und zeigt sämtliche Zeilen an, die die angegebene Zeichenfolge enthalten.

Sie können folgende Schalter mit FIND benutzen:

/V
Wird dieser Schalter angegeben, so zeigt FIND sämtliche Zeilen an, die die angegebene Zeichenfolge nicht enthalten.

/C
Wird dieser Schalter angegeben, so zeigt FIND für jede Datei nur die Anzahl von Zeilen an, die eine Übereinstimmung aufweisen.

/N
Wird dieser Schalter angegeben, so steht vor jeder Zeile ihre relative Zeilennummer in der Datei.

Die Zeichenfolge muß in Anführungszeichen stehen. Durch

FIND "Paradies" BOOK1.TXT BOOK2.TXT

werden beispielsweise sämtliche Zeilen aus BOOK1.TXT und BOOK2.TXT (in dieser Reihenfolge) angezeigt, die die Zeichenfolge "Paradies" enthalten. Aufgrund des Befehls:

```
DIR B: | FIND /V "DAT"
```

zeigt MS-DOS sämtliche Dateinamen auf der Diskette in Laufwerk B an, die die Zeichenfolge DAT nicht enthalten. Sie müssen eine Zeichenfolge, die bereits Anführungszeichen enthält, in doppelte Anführungszeichen setzen.

Wird ein Fehler entdeckt, so antwortet FIND mit einer der folgenden Fehlermeldungen:

Incorrect DOS version
(Falsche DOS-Version)
FIND kann nur mit der MS-DOS-Version 2.0 oder höheren Versionen ausgeführt werden.

FIND: Invalid number of parameters
(FIND: Ungültige Anzahl von Parametern)
Beim FIND-Befehl wurde keine Zeichenfolge angegeben.

FIND: Syntax error
(FIND: Syntaxfehler)
Beim Erteilen des FIND-Befehls wurde eine unzulässige Zeichenfolge eingegeben.

FIND: File not found <filename>
(FIND: Datei <Dateiname> nicht gefunden
Der angegebene Dateiname ist nicht vorhanden oder FIND kann ihn nicht finden.

FIND: Read error in <filename>
(FIND: Lesefehler in <Dateiname>)
Beim Lesen der angegebenen Datei entstand ein Fehler.

FIND: Invalid parameter <option-name>
(FIND: Ungültiger Parameter <Optionsname>
Es wurde eine nicht bestehende Option angegeben.

FOR

NAME

FOR

TYP

Intern; Batch-Verarbeitung

ZWECK

Befehlserweiterung, die bei der Verarbeitung von Batch-Dateien und bei der interaktiven Dateiverarbeitung benutzt wird.

SYNTAX

- Bei Batch-Verarbeitung:

  FOR %% <c> IN <Gruppe> DO <Befehl>

- Bei der interaktiven Verarbeitung:

  FOR % <c> IN <Gruppe> DO <Befehl>

KOMMENTARE

<c> kann ein beliebiges Zeichen mit Ausnahme von 0, 1, 2, 3. . .9 sein, um eine Verwechslung mit den Batch-Parametern %0 bis %9 zu vermeiden.

<Gruppe> ist (<Datenfeld>. . .)

Die Variable %% <c> wird sequentiell auf jedes Element von <Gruppe> gesetzt. Danach wird <Befehl> ausgewertet.

Ist ein Element von <Gruppe> ein Ausdruck, der * und/oder @ enthält, so wird die Variable auf jedes vergleichbare Zeichenmuster der Platte gesetzt. In diesem Fall kann nur ein derartiges <Datenfeld> in der Gruppe vorhanden sein. Jedes zusätzlich angegebene <Datenfeld> wird ignoriert.

HINWEIS: Die Wörter IN, FOR, und DO sind in Großbuchstaben einzugeben.

Beispiele:

```
FOR %%f IN (*.ASM) DO MASM %%f;
FOR %%f IN (FOO BAR BLECH) DO REM %%f
```

Die '%%' sind erforderlich, damit nach der Verarbeitung der Batch-Parameter (%0-%9) noch ein '%' vorhanden ist. Wäre nur '%f' vorhanden, so würde der Prozessor des Batch-Parameters das '%' erkennen, das 'f' sehen, und zu dem Schluß kommen, daß '%f' ein Fehler ist (falsche Parameterreferenz). Daraufhin würde er '%f' löschen, sodaß es niemals im FOR-Befehl erscheinen würde. Steht FOR nicht in einer Batch-Datei, so wird nur ein '%' benutzt.

FORMAT

NAME

FORMAT

TYP

Extern

ZWECK

Formatiert die Diskette im angegebenen Laufwerk für die Aufnahme von MS-DOS Dateien.

SYNTAX

FORMAT [d:] [/V /J /D /I /O /S] (Disketten)
FORMAT [d:] [/V] (Festplatten)

KOMMENTARE

Dieser Befehl formatiert die Platte und initialisiert das Inhaltsverzeichnis sowie die Dateizuordnungstabellen (FAT). Wenn Sie kein Laufwerk angeben, wird die Platte im Standardlaufwerk formatiert. Sämtliche Platten werden mit doppelter Schreibdichte, doppelseitig und entweder mit 9 Sektoren pro Spur bei einer Diskette oder mit 17 Sektoren pro Spur bei einer Festplatte formatiert.

Beim Formatieren einer Festplatte zeigt FORMAT eine Meldung an, mit der der Benutzer um Angabe der Anzahl von Überprüfungen gebeten wird (Leseprüfungen bei jeder Spur). Der Standardwert ist 5. Wenn Sie diesen Wert erhöhen, wird die für das Formatieren benötigte Zeit wesentlich verlängert.

Sechs Schalteroptionen können beim Formatieren einer Diskette angefordert werden. Beim Formatieren einer Festplatte ist nur der Schalter /V gültig. Zwei dieser Schalter werden häufiger benutzt als die anderen.

/V

Wird dieser Schalter angegeben, so unterbricht FORMAT das Formatieren und zeigt eine Meldung an, mit der nach einem Datenträgerkennsatz gefragt wird (er ist für die Kennzeichnung der Platte besonders nützlich).

/S

Wenn Sie diesen Schalter angeben, so kopiert FORMAT die Betriebssystemdateien von der Diskette im Standardlaufwerk auf die neu formatierte Diskette. Die Dateien werden in folgender Reihenfolge kopiert: IO.SYS, MSDOS.SYS, COMMAND.COM. (Andere Dateien können dann einzeln mit dem COPY-Befehl kopiert werden.) Wenn Sie diesen Schalter mit anderen Schaltern benutzen, muß /S als letzter eingegeben werden.

HINWEIS: Um eine gesamte Systemplatte zu kopieren, müssen Sie das Formatieren *ohne Schalter* durchführen und danach DISKCOPY benutzen. DISKCOPY erstellt ein "Spiegelbild" und überschreibt eventuell vorhandene Kennsätze.

Mit der nächsten Schaltergruppe kann ein anderes Format auf der der Diskette erstellt werden. (Diese Formate können erforderlich sein, wenn Informationen von einer nicht im NCR-Format erstellten Diskette kopiert und benutzt werden sollen.) Wenn Sie keinen Schalter angeben, so wird vom Standardformat ausgegeangen: 9 Sektoren pro Spur; doppelseitig, doppelte Schreibdichte (360 KB Diskettenkapazität).

/J

Formatiert mit 9 Sektoren pro Spur; einseitig, doppelte Schreibdichte (180 KB Diskettenkapazität).

/D

Formatiert mit 8 Sektoren pro Spur; doppelseitig, doppelte Schreibdichte (320 KB Diskettenkapazität).

/I

Formatiert mit 8 Sektoren pro Spur; einseitig, doppelte Schreibdichte (160 KB Diskettenkapazität).

Der Schalter /O generiert ein E5-Zeichen in der ersten Position eines leeren (verfügbaren) Verzeichniseintrags. Dieser Schalter darf nur benutzt werden, wenn die Kompatibilität mit älteren Versionen von MS-DOS aufrechterhalten werden muß. Der gegenwärtige Standardeintrag für die Angabe von leeren Verzeichniseinträgen ist 00.

GOTO

NAME

GOTO

TYP

Intern; Batch-Verarbeitung

ZWECK

Befehlserweiterung, die bei der Verarbeitung von Batch-Dateien benutzt wird.

SYNTAX

GOTO<Marke>

KOMMENTARE

Durch GOTO werden Befehle aus der Batch-Datei gelesen, wobei mit der Zeile hinter der <Marke> begonnen wird. Wenn Sie keine Marke definiert haben, wird die gegenwärtige Batch-Datei beendet.

Zum Beispiel erzeugt:

```
:foo
REM looping . . .
GOTO foo ↵
^Z
```

eine unendliche Folge von Meldungen: REM looping . . . .

Wird eine Zeile in einer Batch-Datei mit ':' begonnen, so wird die Zeile von der Batch-Verarbeitung ignoriert. Die auf GOTO folgenden Zeichen definieren eine Marke. Dieses Verfahren läßt sich jedoch auch für die Eingabe von Kommentarzeilen benutzen.

NAME

IF

TYP

Intern; Batch-Verarbeitung

ZWECK

Befehlserweiterung, die bei der Verarbeitung von Batch-Dateien benutzt wird.

SYNTAX

IF <Bedingung> <Befehl>

KOMMENTARE

Bei <Bedingung> handelt es sich um einen der folgenden Parameter:

ERRORLEVEL <Zahl>
Ist nur wahr, wenn das vorhergehende von COMMAND ausgeführte Programm einen Beendigungscode von <Zahl> oder höher hatte.

<Zeichenfolge1> == <Zeichenfolge2>
Ist nur wahr, wenn <Zeichenfolge1> und <Zeichenfolge2> nach der Parameterersetzung identisch sind. Zeichenfolgen dürfen keine eingefügten Trennzeichen aufweisen.

EXIST <Dateiname>
Ist nur wahr, wenn <Dateiname> vorhanden ist.

NOT <Bedingung>
Ist nur wahr, wenn <Bedingung> falsch ist.

Die IF-Anweisung ermöglicht die bedingte Ausführung von Befehlen. Ist die <Bedingung> wahr, so wird der <Befehl> ausgeführt. Ansonsten wird der <Befehl> ignoriert.

HINWEIS: Sie müssen die Wörter ERRORLEVEL, EXIST und NOT in Großbuchstaben angeben.

Beispiele:

```
FOR NOT EXIST \TMP\FOO ECHO Can't find file
IF NOT ERRORLEVEL 3 LINK $1,,,;
```

LOCATE

NAME

LOCATE

TYP

Extern

ZWECK

Wandelt .EXE-Dateien (ausführbare Dateien) in das Binärformat um. Dadurch wird Platz auf der Diskette gespart und das Programmladen schneller ausgeführt.

SYNTAX

LOCATE <Dateispez> [d:] [<Dateiname> [<.erw>]]

KOMMENTARE

Dieser Befehl ist nur nützlich, wenn .EXE-Dateien in das Binärformat umgewandelt werden sollen. Die von Dateispez. angegebene Datei ist die Eingabedatei. Wenn Sie keine Erweiterung angeben, wird standardmäßig die Erweiterung .EXE genommen. Die Eingabedatei wird in das .COM Dateiformat (Speicherabbild des Programms) umgewandelt und in die Ausgabedatei gesetzt. Wenn Sie kein Laufwerk angeben, so wird das Laufwerk der Eingabedatei benutzt. Wenn Sie keinen Ausgabedateinamen angeben, so wird der Name der Eingabedatei benutzt. Wenn Sie keine Dateinamenerweiterung in dem Namen der Ausgabedatei angeben, so erhält die neue Datei eine Erweiterung .BIN.

Die Eingabedatei muß ein gültiges .EXE Format aufweisen, das von dem Linkprogramm erstellt wurde. Der residente oder aktuelle Code und der Datenteil der Datei müssen kleiner als 64 K sein. Ein Stapelsegment darf nicht vorhanden sein.

Zwei Arten der Umwandlung sind möglich, je nachdem, ob der ursprüngliche CS:IP (Codesegment:Instruktionszeiger) in der .EXE-Datei angegeben wird.

1. Wird CS:IP in der .EXE-Datei nicht angegeben, so geht LOCATE von einer reinen Binärumwandlung aus. Sind Segmentfixierungen erforderlich (d.h., das Programm enthält Instruk-

tionen, die eine Segmentverschiebung erforderlich machen), so fordert LOCATE den Benutzer auf, einen Anfangswert einzugeben. Bei diesem Wert handelt es sich um das absolute Segment, bei dem das Programm geladen werden muß. Das daraus entstehende Programm kann nur benutzt werden, wenn es zu der absoluten von einer Benutzeranwendung angegebenen Speicheradresse geladen wird. Der Befehlsprozessor allein kann das Programm nicht einwandfrei laden.

2. Wird CS:IP als 0000:100H angegeben, so wird davon ausgegangen, daß die Datei als .COM-Datei ausgeführt werden soll, wobei der Adressenzeiger von der Assembleranweisung ORG auf 100H gesetzt wird. Die ersten 100H Bytes der Datei werden gelöscht. Segmentfixierungen sind nicht zulässig, da die .COM-Dateien segmentverschieblich sein müssen. Für sie müssen also die in dem Programmierhandbuch (Programmer's Manual) erläuterten Eingabebedingungen erfüllt sein. Nachdem die Umwandlung beendet ist, kann die neu erstellte Datei mit einer .COM-Erweiterung umbenannt werden. Der Befehlsprozessor kann das Programm dann auf dieselbe Art und Weise laden und ausführen, wie die auf der MS-DOS-Diskette befindlichen .COM-Programme.

Wird CS:IP keinem dieser Kriterien gerecht oder wird es dem .COM-Dateikriterium gerecht, verfügt jedoch über Segmentfixierungen, so wird folgende Meldung angezeigt:

File cannot be converted
(Datei kann nicht umgewandelt werden)

Diese Meldung wird auch angezeigt, wenn es sich bei der Datei nicht um eine gültige ausführbare Datei handelt.

Findet LOCATE einen Fehler, so werden eine oder mehrere der folgenden Fehlermeldungen angezeigt:

File not found
(Datei nicht gefunden)
Die Datei steht nicht auf der angegebenen Diskette.

Insufficient memory
(Speicherplatz nicht ausreichend)
Für die Ausführung von LOCATE ist nicht ausreichend Speicherplatz vorhanden.

File creation error
(Fehler bei der Dateierstellung)
LOCATE kann die Ausgabedatei nicht erstellen. Mit CHKDSK können Sie feststellen, ob das Inhaltsverzeichnis voll ist, oder ob eine andere Bedingung den Fehler verursacht hat.

Insufficient disk space
(Nicht ausreichend Platz auf der Diskette)
Für das Erstellen einer neuen Datei ist nicht ausreichend Platz auf der Diskette vorhanden.

Fixups needed — base segment (hex):
(Anfangswerte erforderlich — Basissegment (hex):)
Die Quellendatei (.EXE) erfordert ein Ladesegment. Die absolute Segmentadresse, an der der fertige Modul stehen soll, ist anzugeben.

File cannot be converted
(Datei kann nicht umgewandelt werden)
Die Eingabedatei weist nicht das korrekte Format auf.

WARNING — Read error on .EXE file.
Amount read less than size in header
(WARNUNG — Lesefahler bei .EXE-Datei.
Gelesene Datenmenge kleiner als Größe in Kopfsatz)
Hier handelt es sich nur um eine Warnmeldung.

MKDIR

NAME

MKDIR

TYP

Intern

SYNONYM

MD

ZWECK

Erstellt ein neues Inhaltsverzeichnis.

SYNTAX

MKDIR <Pfadname>

KOMMENTARE

Mit diesem Befehl wird eine hierarchische Verzeichnisstruktur erstellt. Arbeitet der Benutzer in seinem Stammverzeichnis, so kann er mit dem MKDIR-Befehl Unterverzeichnisse erstellen. Mit dem Befehl

MKDIR \USER

wird ein Unterverzeichnis \USER im Stammverzeichnis erstellt. Um ein Verzeichnis namens JOE unter \USER zu erstellen, wird

MKDIR \USER\JOE

eingegeben.

NAME

MORE

TYP

Extern

ZWECK

Sendet die Ausgabe in Form eines Bildschirminhalts nach dem anderen an die Konsole.

SYNTAX

MORE

KOMMENTARE

MORE ist ein Filter, der von der Standardeingabe-Einheit (wie beispielsweise einem Befehl vom Terminal) liest und jeweils einen Bildschirminhalt mit Informationen anzeigt. Danach unterbricht der MORE-Befehl die Bildschirmausgabe und zeigt die Meldung - - MORE - - am unteren Rande des Bildschirms an.

Durch Betätigung der <CR> Taste wird ein weiterer Bildschirminhalt mit Informationen angezeigt. Dieses Verfahren wird fortgesetzt, bis sämtliche Eingabedaten gelesen wurden.

Der MORE-Befehl ist für die Anzeige einer langen Datei nützlich. Wenn Sie

```
TYPE MYFILES.COM MORE
```

eingeben, zeigt MS-DOS die (in dem Standardlaufwerk befindliche) Datei MYFILES.COM Seite für Seite am Bildschirm an.

PATH

NAME

PATH

TYP

Intern

ZWECK

Setzt einen Befehlspfad.

SYNTAX

PATH [<Pfadname>[;<Pfadname>] . . .]

KOMMENTARE

Mit diesem Befehl kann MS-DOS mitgeteilt werden, welche Inhaltsverzeichnisse nach externen Befehlen abgesucht werden sollen, nachdem MS-DOS das Arbeitsverzeichnis abgesucht hat. Der Standardwert ist \BIN, wobei \BIN der Name des Inhaltsverzeichnisses ist, in dem sämtliche externen Befehle von MS-DOS stehen.

Um MS-DOS anzuweisen, das Inhaltsverzeichnis \BIN\USER\JOE (zusätzlich zu dem \BIN Verzeichnis) nach externen Befehlen abzusuchen, geben Sie

```
PATH \BIN\USER\JOE
```

ein. MS-DOS sucht nun das Inhaltsverzeichnis \BIN\USER\JOE nach externen Befehlen ab, bis Sie einen anderen Pfad festlegen oder MS-DOS ausschalten.

MS-DOS kann auch angewiesen werden, mehr als einen Pfad abzusuchen. Hierzu können Sie verschiedene Pfadnamen angeben, die durch Semikolon voneinander getrennt sind. Mit

```
PATH \BIN\USER\JOE; \BIN\USER\SUE; \BIN\DEV
```

wird MS-DOS beispielsweise angewiesen, die von den obigen Pfadnamen angegebenen Inhaltsverzeichnisse nach externen Befehlen

abzusuchen. MS-DOS sucht die Pfadnamen in der in dem PATH-Befehl angegebenen Reihenfolge ab.

Wenn Sie den PATH-Befehl ohne Optionen angeben, wird der gegenwärtige Pfad ausgedruckt. Wenn Sie PATH; angeben, setzt MS-DOS den NULL-Pfad. Dies bedeutet, daß nur das Arbeitsverzeichnis nach externen Befehlen abgesucht wird.

PAUSE

NAME

PAUSE

TYP

Intern

ZWECK

Suspendiert die Ausführung der Batch-Datei.

SYNTAX

PAUSE [Kommentar]

KOMMENTARE

Während der Ausführung einer Batch-Datei kann es beispielsweise erforderlich sein, die Disketten auszuwechseln. Mit PAUSE wird die Ausführung suspendiert, bis der Benutzer eine beliebige Taste betätigt, mit Ausnahme der Taste <CONTROL-C>.

Erkennt der Befehlsprozessor PAUSE, so druckt er:

Strike a key when ready . . .
(Beliebige Taste betätigen, wenn bereit . . .)

Wenn Sie <CONTROL-C> betätigen, wird eine weitere Systemmeldung angezeigt:

Abort batch job (Y/N)?
(Batch-Betrieb abbrechen (J/N)?)

Wenn Sie Y als Antwort auf diese Systemmeldung eingeben, wird die Ausführung der restlichen Batch-Befehlsdatei abgebrochen. Die Steuerung wird wieder der Ebene der Betriebssystembefehle zurückgegeben. Deshalb wird PAUSE für die Aufteilung einer Batch-Datei benutzt, so daß der Benutzer die Batch-Befehlsdatei zwischendurch beenden kann.

Der Kommentar ist wahlweise und wird auf derselben Zeile wie PAUSE eingegeben. Möglicherweise soll der Benutzer der Batch-Datei durch eine Meldung informiert werden, daß die Verarbeitung

der Batch-Datei vorübergehend eingestellt wurde. Beispielsweise sollen die Disketten in einem der Laufwerke ausgewechselt werden. In solchen Fällen kann wahlweise eine Systemmeldung ausgegeben werden. Die Kommentarmeldung wird vor der Meldung "Strike a key" angezeigt.

PRINT

NAME

PRINT

TYP

Extern

ZWECK

Druckt eine Textdatei auf einem Zeilendrucker aus, während andere MS-DOS Befehle verarbeitet werden. (Dies wird normalerweise als "Drucken im Hintergrund" bezeichnet.)

SYNTAX

PRINT [[Dateispez] [/T] [/C] [/P]] . . .

KOMMENTARE

Der PRINT-Befehl ist nur dann zu benutzen, wenn ein Zeilendrucker an den Computer angeschlossen ist. Für diesen Befehl stehen folgende Schalter zur Verfügung:

/T — TERMINATE
Dieser Schalter löscht sämtliche Dateien in der Druckerwarteschlange (die auf das Ausdrucken warten). Eine entsprechende Meldung wird ausgedruckt.

/C — CANCEL
Mit diesem Schalter wird der Löschmodus aktiviert. Die vorhergehende Dateispezifikation und alle nachfolgenden Dateispezifikationen werden in der Druckerwarteschlange suspendiert, bis der Benutzer einen Schalter /P eingibt.

/P — PRINT
Mit diesem Schalter wird der Druckmodus aktiviert. Die vorhergehende Dateispezifikation und alle nachfolgenden Dateispezifikationen werden der Druckerwarteschlange hinzugefügt, bis der Benutzer einen Schalter /C eingibt.

Bei PRINT ohne Optionen wird der Inhalt der Druckerwarteschlange auf dem Bildschirm angezeigt, ohne daß die Warteschlange dadurch beeinflußt wird.

Beispiele:

```
PRINT /T
```

löscht die Druckerwarteschlange.

```
PRINT /T*.ASM
```

löscht die Druckerwarteschlange und setzt sämtliche im Standardlaufwerk befindliche .ASM Dateien in eine Warteschlange.

```
PRINT A:TEMP1.TST/C A:TEMP2.TST A:TEMP3.TST
```

entfernt die drei angegebenen Dateien aus der Druckerwarteschlange.

```
PRINT TEMP1.TST /C TEMP2.TST /P TEMP3.TST
```

entfernt TEMP1.TST aus der Warteschlange und fügt TEMP2.TST sowie TEMP3.TST der Warteschlange hinzu.

Wird ein Fehler entdeckt, so zeigt PRINT eine der folgenden Fehlermeldungen an:

Name of list device [PRN:]
(Name des Druckgerätes [PRN:])
: Diese Meldung wird angezeigt, wenn Sie PRINT zum erstenmal ausführen. Jede Ausgabe-Einheit kann angegeben werden. Dieses Gerät wird dann zum Ausgabegerät für PRINT. Wie in [ ] angegeben, wird durch Betätigung von <CR> die Druckeinheit PRN benutzt.

List output is not assigned to a device
(Listenausgabe ist keinem Gerät zugeordnet)
: Diese Meldung erscheint, wenn der auf die vorhergehende Meldung eingegebene "Name des Druckgeräts" ungültig ist. Nachfolgende Versuche führen zur Anzeige derselben Meldung, bis ein gültiges Gerät angegeben wird.

PRINT queue is full
(Druckerwarteschlange ist voll)
: In der Warteschlange ist Platz für 10 Dateien. Wird versucht, mehr als 10 Dateien in die Warteschlange zu setzen, so wird diese Meldung auf der Konsole angezeigt.

PRINT queue is empty
(Druckerwarteschlange ist leer)
In der Druckerwarteschlange stehen keine Dateien.

No files match d:XXXXXXXX.XXX
(Keine Dateien stimmen mit d:XXXXXXXX.XXX überein)
Die Dateispezifikation enthält Dateien, die der Warteschlange hinzugefügt werden sollen. Mit der Spezifikation stimmen jedoch keine Daten überein. (Stehen in der Warteschlange keine Dateien, die mit einer gelöschten Dateispezifikation übereinstimmen, so wird keine Fehlermeldung angezeigt.)

Drive not ready
(Laufwerk nicht bereit)
Diese Meldung kann erscheinen, wenn PRINT auf eine Platte zugreifen möchte. PRINT wiederholt den Versuch, bis das Laufwerk bereit ist. Jeder andere Fehler führt dazu, daß die aktuelle Datei aus der Warteschlange entfernt wird. In diesem Fall wird eine Fehlermeldung am Drucker ausgegeben.

All files cancelled
(Sämtliche Dateien aus der Warteschlange entfernt)
Wenn Sie den /T (TERMINATE)-Schalter angeben, wird die Meldung "All files cancelled by operator" (Alle Dateien vom Bediener entfernt) am Drucker ausgegeben. Wird die gerade ausgedruckte Datei durch /C entfernt, so wird die Meldung "File cancelled by operator" (Datei vom Bediener entfernt) ausgedruckt.

PROMPT

NAME

PROMPT

TYP

Intern

ZWECK

Ändert die Zeichenfolge für die MS-DOS Eingabeaufforderung.

SYNTAX

PROMPT [<Text der Eingabeaufforderung>]

KOMMENTARE

Mit diesem Befehl kann die MS-DOS Eingabeaufforderung geändert werden. Wenn Sie keinen Text eingeben, wird die Standardeingabeaufforderung benutzt, bei der es sich um die Standardlaufwerkbezeichnung handelt. Für die Eingabeaufforderung kann jedoch auch eines der nachfolgend angegebenen Zeichen benutzt werden.

Sie dürfen die nachfolgenden Zeichen im PROMPT-Befehl für die Angabe von Spezialeingabeaufforderungen benutzen. Im PROMPT-Befehl muß vor dem gewünschten Zeichen ein Dollarzeichen ($) stehen:

Dieses Zeichen		ergibt folgende Eingabeaufforderung
$	—	'$'-Zeichen
t	—	Gegenwärtige Zeit
d	—	Gegenwärtiges Datum
p	—	Gegenwärtiges Inhaltsverzeichnis des Standardlaufwerks
v	—	Versionsnummer
n	—	Standardlaufwerk
g	—	'>' Zeichen
l	—	'<' Zeichen
b	—	'|' Zeichen
_	—	CR LF-Folge
s	—	Leerzeichen

h	—	Rückschritt
e	—	ASCII Code X'1B' (Umschaltfolge)

Beispiel:

```
PROMPT $n:
```

Mit diesem Befehl wird die Eingabeaufforderung in Form des Standardlaufwerks, gefolgt von einem Doppelpunkt, festgelegt.

In den Eingabeaufforderungen können auch Umschaltfolgen benutzt werden. Zum Beispiel:

```
PROMPT $e [7m$n$g$e[m
```

legt die Anzeige der Eingabeaufforderung im Bildumkehrmodus fest und verwendet für sonstigen Text den normalen Anzeigemodus.

NAME

RDCPM

TYP

Extern

ZWECK

Überträgt NCR CP/M® Dateien auf eine für MS-DOS formatierte Diskette.

SYNTAX

RDCPM DIR d:	(Zeigt das Inhaltsverzeichnis auf der CP/M Diskette an)
RDCPM d:Dateiname [d:]	(Überträgt die CP/M Datei auf die MS-DOS Diskette)

KOMMENTARE

RDCPM liest die Datei von einer NCR CP/M Diskette und überträgt sie auf eine für MS-DOS formatierte Diskette. Nach der Übertragung ist die Datei eine MS-DOS Datei.

Mit der DIR Variante von RDCPM wird das Inhaltsverzeichnis einer CP/M Diskette angezeigt. Auf diese Weise können Sie die Namen der Dateien lesen. Die Laufwerkbezeichnung für die CP/M Diskette ist unbedingt anzugeben.

Für die Übertragung einer Datei müssen Sie die Laufwerkbezeichnung der CP/M Diskette und den Dateinamen angeben.

Für die Übertragung mehrerer Dateien können die Dateigruppensymbole (* und ?) benutzt werden. (Für eine Beschreibung der Benennung von Dateien mit Dateigruppensymbolen sei auf Kapitel 3 verwiesen.) Die Bezeichnung des Bestimmungslaufwerks erfolgt wahlweise. Wenn Sie kein Laufwerk angeben, wird davon ausgegangen, daß die Diskette im Standardlaufwerk die Bestimmungsdiskette ist.

WORDSTAR TEXT FILE

A> RDCPM DIR B:

RDCPM B:

Beispiele:

```
A>RDCPM C: MYFILE.TXT B:
A>RDCPM C: MYFILE.TXT
```

Der erste Befehl überträgt MYFILE.TXT von der Diskette in Laufwerk C auf die Diskette in Laufwerk B. Der zweite Befehl überträgt dieselbe Datei auf die Diskette in Laufwerk A (Standardlaufwerk).

HINWEIS: Die Bezeichnungen für Quellen- und Bestimmungslaufwerk müssen unterschiedlich sein. Verfügt der Benutzer nur über ein einziges Diskettenlaufwerk und möchte er eine CP/M Datei von einer Diskette übertragen, so muß er die Datei zuerst (mit dem CP/M Betriebssystem) auf ein anderes Plattenlaufwerk kopieren. Erst dann kann er den RDCPM-Befehl benutzen.

Die folgenden Meldungen können erscheinen. Die meisten dieser Meldungen erklären sich von selbst.

Hard disk error on CP/M drive
(Hardwareplattenfehler beim CP/M Laufwerk)
Die von der Bezeichnung für das Quellenlaufwerk angegebene Diskette darf keine CP/M Diskette sein.

Source and destination drives must not be the same
(Quellen- und Bestimmungslaufwerke dürfen nicht identisch sein)
Die CP/M und MS-DOS Disketten müssen in verschiedenen Laufwerken eingelegt sein.

Drive not available for CP/M reading
(Laufwerk für das Lesen von CP/M nicht verfügbar)
MS-DOS kann nicht auf das angegebene Laufwerk zugreifen. Zu diesem Fehler kommt es, wenn die MS-DOS Plattenkonfiguration nicht korrekt ist. So wurde beispielsweise:

```
RDCPM C: YOURFILE B:
```

angegeben, das System ist jedoch für ein 2-Diskettensystem mit den Laufwerken A und B konfiguriert. (Es wurde keine Festplatte definiert.) Ist der Fehler auf die Definition der Konfiguration zurückzuführen, so sollten Sie das CONFIG-Dienstprogramm aufrufen, um die Definition zu ändern. Danach können Sie RDCPM erneut ausführen.

Insufficient disk space
(Nicht genügend Platz auf der Diskette)

Auf der MS-DOS Bestimmungsdiskette ist nicht genügend Platz für die CP/M Datei(en) vorhanden.

No room in directory to create file
(Kein Platz im Inhaltsverzeichnis für das Erstellen einer Datei)

In dem Inhaltsverzeichnis auf der MS-DOS Diskette ist kein Platz für das Erstellen eines Eintrags für die CP/M Datei(en) vorhanden.

Source file name missing
(Name der Quellendatei fehlt)

Bei der angegebenen Datei handelt es sich nicht um die NCR CP/M Diskette. Prüfen Sie, ob der Dateiname richtig eingegeben wurde.

Source file not found
(Quellendatei nicht gefunden)

Die angegebene Datei befindet sich nicht auf der NCR CP/M Diskette. Prüfen Sie, ob der Dateiname richtig eingegeben wurde.

File transfer complete
(Dateiübertragung beendet)

Die angegebene(n) Datei(en) wurde(n) erfolgreich übertragen.

RECOVER

NAME

RECOVER

TYP

Extern

ZWECK

Stellt eine Datei oder eine gesamte Diskette mit fehlerhaften Sektoren wieder her.

SYNTAX

RECOVER <Dateiname>
RECOVER <d:>

KOMMENTARE

Ist ein Sektor auf einer Diskette fehlerhaft, so kann entweder die Datei, die diesen Sektor enthält, (ohne den fehlerhaften Sektor) oder die gesamte Diskette wiederhergestellt werden (wenn sich der fehlerhafte Sektor im Inhaltsverzeichnis befindet.)

Für das Wiederherstellen einer bestimmten Datei geben Sie

RECOVER <Dateiname>

ein. MS-DOS liest die Datei Sektor für Sektor und überspringt die fehlerhaften Sektoren. Findet MS-DOS den bzw. die fehlerhaften Sektoren, so werden sie markiert. MS-DOS weist dann diesem Sektor keine Daten mehr zu.

Für das Wiederherstellen einer Diskette geben Sie

RECOVER <d:>

ein, wobei d: der Buchstabe des Laufwerks ist, das die wiederherzustellende Diskette enthält.

Ist im Stammverzeichnis nicht ausreichend Platz vorhanden, so druckt RECOVER eine Meldung aus und speichert Informationen über die zusätzlichen Dateien in der Dateizuordnungs-

tabelle. RECOVER kann dann erneut ausgeführt werden, um diese Dateien wieder herzustellen, wenn im Stammverzeichnis wieder mehr Platz vorhanden ist.

REM

NAME

REM (REMARK)

TYP

Intern

ZWECK

Zeigt während der Ausführung einer Batch-Datei Bemerkungen an, die auf derselben Zeile wie der REM-Befehl in der Batch-Datei stehen.

SYNTAX

REM [Kommentar]

KOMMENTARE

Die einzigen zulässigen Trennzeichen im Kommentar sind das Leerzeichen, der Tabulatorstopp und das Komma. Zum Beispiel:

```
1: REM Diese Datei prüft neue Disketten
2: REM Sie trägt den Namen NEWDISK.BAT
3: PAUSE Neue Diskette in Laufwerk B einlegen
4: FORMAT B:/S
5: DIR B:
6: CHKDSK B:
```

NAME

REN (RENAME)

TYP

Intern

SYNONYM

RENAME

ZWECK

Ändert den Namen der ersten Option (Dateispez) in die zweite Option (Dateiname).

SYNTAX

REN <Dateispez> <Dateiname>

KOMMENTARE

Die erste Option (Dateispez) muß eine Laufwerkbezeichnung darstellen, wenn die Diskette sich nicht im Standardlaufwerk befindet. Eine Laufwerkbezeichnung für die zweite Option (Dateiname) wird ignoriert. Die Datei bleibt auf der Diskette, auf der sie sich gerade befindet.

Sie können in beiden Optionen Dateigruppensymbole benutzen. Sämtliche Dateien, die mit der ersten Dateispezifikation übereinstimmen, werden umbenannt. Werden in dem zweiten Dateinamen Dateigruppensymbole angegeben, so erfolgt keine Änderung der entsprechenden Zeichenpositionen.

Mit dem folgenden Befehl werden beispielsweise die Namen sämtlicher Dateien mit der Erweiterung .LST in gleiche Namen mit der Erweiterung .PRN geändert:

```
REN *.LST *.PRN
```

Im nächsten Beispiel benennt REN die Datei ABODE in Laufwerk B in ADOBE um:

```
REN B:ABODE ?D?B?
```

Die Datei bleibt in Laufwerk B.

Wird versucht, eine Datei mit einem schon im Inhaltsverzeichnis vorhandenen Namen umzubenennen, so wird die Fehlermeldung "File not found" (Datei nicht gefunden) angezeigt.

NAME

RMDIR (REMOVE DIRECTORY)

TYP

Intern

SYNONYM

RD

ZWECK

Löscht ein Inhaltsverzeichnis aus einer hierarchischen Verzeichnisstruktur.

SYNTAX

RMDIR [d:] <Pfadname>

KOMMENTARE

Mit diesem Befehl wird ein leeres Inhaltsverzeichnis oder ein Inhaltsverzeichnis mit den Einträgen . und . . gelöscht.

Um das Inhaltsverzeichnis \BIN\USER\JOE zu löschen, sollten Sie zuerst einen DIR-Befehl für diesen Pfad erteilen, um zu gewährleisten, daß das Inhaltsverzeichnis keine wichtigen Dateien enthält, die nicht gelöscht werden sollen. Danach geben Sie

```
RMDIR \BIN\USER\JOE
```

ein. Das Inhaltsverzeichnis wird aus der Verzeichnisstruktur gelöscht.

SET

NAME

SET

TYP

Intern

ZWECK

Setzt einen Zeichenfolgewert auf den Wert einer anderen Zeichenfolge zur Benutzung in nachfolgenden Programmen.

SYNTAX

SET [<Zeichenfolge=Zeichenfolge>]

KOMMENTARE

Dieser Befehl ist nur von Bedeutung, wenn Werte gesetzt werden sollen, die von den vom Benutzer geschriebenen Programmen benutzt werden. Ein Anwenderprogramm kann sämtliche Werte überprüfen, die mit dem SET-Befehl gesetzt wurden, indem es SET ohne Optionen erteilt. Durch SET TTY = VT52 wird beispielsweise der TTY-Wert auf VT52 gesetzt, bis dieser Wert mit einem weiteren SET-Befehl geändert wird.

Sie können den SET-Befehl auch in der Batch-Verarbeitung benutzen. Auf diese Weise können die austauschbaren Parameter mit Namen anstatt mit Zahlen definiert werden. Enthält die Batch-Datei die Anweisung "LINK %FILE%", so können Sie den Namen, den MS-DOS für diese Variable benutzt, mit dem SET-Befehl setzen. Durch den Befehl SET FILE = DOMORE wird der Parameter %FILE% durch den Dateinamen DOMORE ersetzt. Deshalb ist es nicht erforderlich, jede Batch-Datei einzeln aufzubereiten, um die austauschbaren Parameternamen zu ändern. Vergessen Sie nicht, bei der Benutzung von Texten (anstelle von Zahlen) als austauschbare Parameter, den Namen jeweils mit einem Prozentzeichen abzuschließen.

NAME

SHIFT

TYP

Intern; Batch-Verarbeitung

ZWECK

Ermöglicht den Zugriff zu mehr als 10 austauschbaren Parametern bei der Verarbeitung von Batch-Dateien.

SYNTAX

SHIFT

KOMMENTARE

Normalerweise sind die Befehlsdateien auf die Abarbeitung von 10 Parametern, %0 bis %9, beschränkt. Um den Zugriff zu mehr als 10 Parametern zu ermöglichen, wird SHIFT zur Änderung der Parameter der Befehlszeile benutzt.

Bei:

%0 = "foo"
%1 = "bar"
%2 = "name"
%3 . . . %9 sind leer

führt SHIFT zu:

%0 = "bar"
%1 = "name"
%2 . . . %9 sind leer

Stehen mehr als 10 Parameter auf einer Befehlszeile, so werden die hinter dem zehnten Parameter (%9) stehenden Parameter durch aufeinanderfolgende Verschiebungen nacheinander in %9 verschoben.

SORT

NAME

SORT

TYP

Extern

ZWECK

SORT liest die Eingabe von dem Benutzerterminal, sortiert die Daten und schreibt sie dann auf den Terminalbildschirm oder in Dateien.

SYNTAX

SORT [/R] [/+n]

KOMMENTARE

SORT kann beispielsweise dazu benutzt werden, eine Datei unter Verwendung einer bestimmten Anfangsspalte alphabetisch zu ordnen. Mit zwei Schaltern kann der Benutzer Optionen auswählen:

/R
: Kehrt den Sortiervorgang um, d.h., die Sortierreihenfolge ist von Z bis A.

/+n
: Sortiert ab Spalte n, wobei n eine beliebige Zahl darstellt. Wenn Sie diesen Schalter nicht angeben, beginnt das Sortieren ab Spalte 1.

Im nachstehenden Beispiel liest der Befehl die Datei UNSORT .TXT, kehrt den Sortiervorgang um und schreibt die Ausgabe dann in eine Datei namens SORT.TXT:

```
SORT /R<UNSORT.TXT>SORT.TXT
```

Durch den nächsten Befehl wird die Ausgabe des Verzeichnisbefehls an den SORT-Filter übergeben. Der SORT-Filter sortiert die Verzeichnisauflistung ab Spalte 14 (in dieser Spalte in der Verzeichnisauflistung ist die Dateigröße enthalten) und sendet die

Ausgabe dann an die Konsole. Das Ergebnis dieses Befehls ist also ein nach Dateigröße sortiertes Inhaltsverzeichnis:

```
DIR \ SORT /+14
```

Der Befehl

```
DIR \ SORT /+14 \ MORE
```

bewirkt dasselbe wie der Befehl in dem vorhergehenden Beispiel. Hier gibt der MORE-Filter dem Benutzer jedoch die Möglichkeit, das sortierte Inhaltsverzeichnis mit geeigneten Ausgabe-Unterbrechungen am Bildschirm zu lesen.

HINWEIS: Nach A>SORT muß eine Tastatur-Eingabe erfolgen, die nur von CONTROL-Z beendet werden kann.

SYS

NAME

SYS (SYSTEM)

TYP

Extern

ZWECK

Überträgt die MS-DOS Systemdateien von der Diskette im Standardlaufwerk auf die Diskette in dem von d: angegebenen Laufwerk.

SYNTAX

SYS <d>:

KOMMENTARE

SYS wird normalerweise dazu benutzt, die Systemdateien auf den neuesten Stand zu bringen oder sie auf eine formatierte Diskette unter Ausschluß anderer Dateien zu setzen. Für d: ist ein Eintrag erforderlich.

Stehen IO.SYS und MSDOS.SYS auf der Bestimmungsdiskette, so müssen sie dieselbe Menge Platz auf der Diskette belegen, wie von der neuen Systemdatei benötigt wird. Dies bedeutet, daß keine Systemdateien von einer MS-DOS 2.0 Diskette auf eine MS-DOS 1.1 Diskette übertragen werden können. Sie müssen die MS-DOS 1.1 Diskette mit dem MS-DOS Befehl FORMAT neu formatieren, bevor Sie den SYS-Befehl einsetzen.

Die Bestimmungsdiskette muß vollständig leer sein oder die Systemdateien IO.SYS und MSDOS.SYS bereits enthalten. Die übertragenen Dateien werden in der folgenden Reihenfolge kopiert:

IO.SYS
MSDOS.SYS

IO.SYS und MSDOS.SYS sind beides versteckte Dateien, die bei der Ausführung des DIR-Befehls nicht angezeigt werden. COMMAND.COM (der Befehlsprozessor) wird nicht übertragen. Um COMMAND.COM zu übertragen, müssen Sie den COPY-Befehl benutzen.

Entdeckt SYS einen Fehler, so wird eine der folgenden Meldungen angezeigt:

No room for system on destination disk
(Kein Platz für Systemdatei auf der Bestimmungsdiskette)

Auf der Bestimmungsdiskette ist nicht ausreichend Platz für die IO.SYS und MSDOS.SYS Dateien vorhanden.

Incompatible system size
(Nicht kompatible Systemdateigröße)

Die Systemdateien IO.SYS und MSDOS.SYS belegen nicht eine der neuen Systemdatei entsprechende Menge an Speicherplatz auf der Bestimmungsdiskette.

TIME

NAME

TIME

TYP

Intern

ZWECK

Zeigt die Uhrzeit an und legt sie fest.

SYNTAX

TIME [<hh>[:<mm>]]

KOMMENTARE

Wird der TIME-Befehl ohne Parameter eingegeben, so wird folgende Meldung angezeigt:

Current time is <hh>:<mm>:<ss>.<cc>
Enter new time: _

(Gegenwärtige Uhrzeit ist <hh>:<mm>:<ss>.<cc>
Neue Uhrzeit eingeben:_)

Soll die angezeigte Uhrzeit nicht geändert werden, betätigen Sie die <CR> Taste. Sie können eine geänderte Uhrzeit als Option für den TIME-Befehl wie folgt eingeben:

TIME 8:20

Die neue Uhrzeit darf nur in Form von Zahlen eingegeben werden. Buchstaben dürfen nicht benutzt werden. Folgende Optionen sind zulässig:

<hh> = 00-24
<mm> = 00-59

Die Einträge für Stunden und Minuten müssen durch Doppelpunkte voneinander getrennt sein. Die <ss> (Sekunden) oder <cc> (Hundertstel Sekunden) müssen nicht eingegeben werden.

MS-DOS benutzt Ihre Eingabe als die nun gültige Uhrzeit. Sind Optionen oder Trennzeichen ungültig, so zeigt MS-DOS folgende Meldung an:

```
Invalid time
Enter new time:_
```

```
(Ungültige Uhrzeit
Neue Uhrzeit eingeben:_)
```

MS-DOS wartet auf gültige Eingaben (oder <CR>).

TYPE

NAME

TYPE

TYP

Intern

ZWECK

Zeigt den Inhalt der Datei am Konsolbildschirm an.

SYNTAX

TYPE <Dateispez>

KOMMENTARE

Mit diesem Befehl können Sie eine Datei einsehen, ohne sie zu ändern. Um eine Datei zu ändern, müssen Sie sich zuerst mit Hilfe des DIR-Befehls des Namens der Datei vergewissern. Anschließend können Sie mit EDLIN weiterverfahren.

Die einzige von TYPE ausgeführte Formatierung besteht darin, daß Tabulatorzeichen in eine entsprechende Anzahl von Leerzeichen umgesetzt werden, damit eine spaltengerechte Ausgabe (Tabulator = jede achte Spalte) gewährleistet ist.

Hier muß beachtet werden, daß bei einer Anzeige von Binärdateien Steuerzeichen (wie beispielsweise CONTROL-Z) an den Computer gesendet werden. Dies gilt auch für akustische Signale, Seitenvorschübe und Umschaltfolgen.

VER

NAME

VER

TYP

Intern

ZWECK

Druckt die MS-DOS Versionsnummer aus.

SYNTAX

VER

KOMMENTARE

Möchte der Benutzer wissen, welche Version von MS-DOS er benutzt, so gibt er VER ein. Die Versionsnummer wird dann am Bildschirm angezeigt.

VERIFY

NAME

VERIFY

TYP

Intern

ZWECK

Schaltet den Prüfschalter beim Schreiben auf Diskette ein oder aus.

SYNTAX

VERIFY [ON]
VERIFY [OFF]

KOMMENTARE

Dieser Befehl hat denselben Zweck wie der Schalter /V im COPY-Befehl. Soll geprüft werden, ob sämtliche Dateien richtig auf die Diskette geschrieben wurden, so kann der VERIFY-Befehl benutzt werden. Mit ihm wird MS-DOS angwiesen, zu überprüfen, ob sämtliche Dateien intakt sind (z.B., ob fehlerhafte Sektoren enthalten sind). MS-DOS führt dann jedesmal, wenn Daten auf eine Diskette geschrieben werden, einen VERIFY-Befehl aus. Der Benutzter erhält nur dann eine Fehlermeldung, wenn MS-DOS die Datei nicht erfolgreich auf Diskette schreiben konnte.

VERIFY ON bleibt in Kraft, bis Sie dies in einem Programm (durch einen SET VERIFY Systemaufruf — siehe "Programmer's Manual") oder durch einen VERIFY OFF Befehl ändern.

Wenn Sie die aktuelle Einstellung von VERIFY ermitteln wollen, geben Sie VERIFY ohne Optionen ein.

VOL

NAME

VOL (VOLUME)

TYP

Intern

ZWECK

Zeigt den Datenträgerkennsatz der Diskette an, sofern dieser vorhanden ist.

SYNTAX

VOL [d:]

KOMMENTARE

Dieser Befehl druckt den Datenträgerkennsatz der Diskette in Laufwerk d: aus. Wenn Sie kein Laufwerk angeben, druckt MS-DOS den Datenträgerkennsatz der Diskette im Standardlaufwerk aus.

MS-DOS EDITIER- UND FUNKTIONSTASTEN

BESONDERE EDITIERTASTEN

Die besonderen Editiertasten sind von besonderem Interesse, da die MS-DOS Befehlseingabe sich deutlich von der Befehlseingabe in anderen Betriebssystemen unterscheidet. Eine Tastenfolge muß nicht ständig neu eingegeben werden, da die letzte Befehlszeile automatisch in einem Spezialspeicherbereich gespeichert wird, der als Schablone bezeichnet wird.

Mit Hilfe der Schablone und der besonderen Editiertasten können Sie von den folgenden MS-DOS Einrichtungen Gebrauch machen:

- Eine Befehlszeile kann sofort wiederholt werden, indem Sie zwei Tasten betätigen.
- Wird ein Tastendruck vergessen oder versehentlich betätigt, so können Sie die Befehlszeile editieren, ohne die gesamte Befehlszeile neu einzugeben.
- Eine Befehlszeile, die einer vorhergehenden Befehlszeile in etwa entspricht, kann mit einem Mindestaufwand an Tasteneingabe aufbereitet und ausgeführt werden, indem eine besondere Editiertaste betätigt wird.

Abbildung 6.1 zeigt die Verbindung zwischen der Befehlszeile und der Schablone.

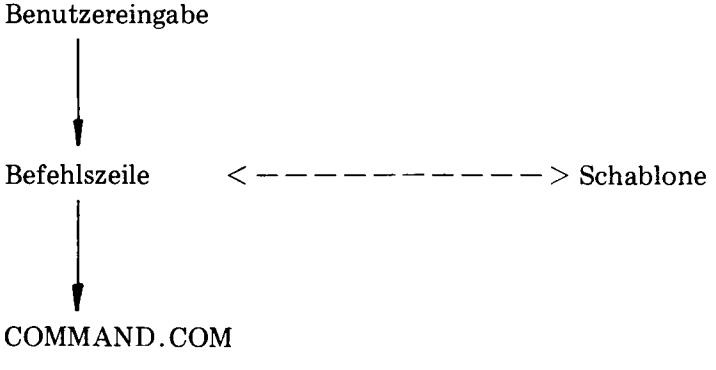


Abb. 6.1 Befehlszeile und Schablone

Der Benutzer gibt einen Befehl an MS-DOS mittels der Befehlszeile ein. Betätigt er die <CR> Taste, so wird der Befehl automatisch an den Befehlsprozessor (COMMAND.COM) zur Ausführung gesendet. Gleichzeitg wird eine Kopie dieses Befehls an die Schablone gesendet. Der Benutzer kann nun den Befehl erneut aufrufen oder ihn mit den besonderen MS-DOS Editiertasten ändern.

Abb. 6.2 enthält eine vollständige Liste der besonderen Editiertasten. Jede dieser Tasten wird im einzelnen in Kapitel 7 beschrieben. Dort werden sie zur Aufbereitung der Textdateien benutzt.

Funktion	Taste(n)*	Beschreibung
Kopieren eines Zeichens	<COPY1> ESC S	Kopiert ein Zeichen aus der Schablone in die Befehlszeile.
Kopieren bis zu Zeichen	<COPYUP> ESC T	Kopiert sämtliche Zeichen bis zu dem angegebenen Zeichen aus der Schablone und setzt diese Zeichen in die Befehlszeile.
Kopieren der Schablone	<COPYALL> ESC U	Kopiert alle verbleibenden Zeichen in der Schablone in die Befehlszeile
Überspringen eines Zeichens	<SKIP1> ESC V	Überspringt ein Zeichen in der Schablone (d.h. kopiert es nicht).
Überspringen bis zu Zeichen	<SKIPUP> ESC W	Überspringt die Zeichen in der Schablone bis zu dem angegebenen Zeichen (d.h. kopiert sie nicht).
Beenden der Eingabe	<VOID> ESC E	Beendet die aktuelle Eingabe; beläßt die Schablone unverändert.
Löschen einer Zeile	<KILL> ESCJ	Ersetzt den Inhalt der Schablone durch die neue Eingabe.
Einfügemodus	<INSERT> ESC P	Leitet den Einfügemodus ein.
Ersatzmodus	<EXIT> ESC Q	Schaltet den Einfügemodus ab. Dies ist die Standardvorgabe.
Neue Schablone	<NEWLINE> ↵	Macht die neue Zeile zur neuen Schablone

*Die meisten Funktionen erfordern eine Eingabe über zwei Tasten. Die Tasten dürfen nicht gleichzeitig betätigt werden.

Abb. 6.2 Besondere Editierfunktionen

Aus dieser Abbildung ist ersichtlich, daß eine Editierfunktion (mit Ausnahme von <NEWLINE>) mit zwei Tasten eingeleitet wird. Zuerst wird die ESC-Taste betätigt und danach die Editiertaste. Die beiden Tasten dürfen nicht gleichzeitig betätigt werden.

In den folgenden Beispielen wird jeweils der Name der Funktionstaste und nicht die eigentliche Taste benutzt.

Wenn Sie

DIR PROG.COM

eingeben, zeigt MS-DOS Informationen über die Datei PROG.COM am Bildschirm an. Die Befehlszeile wird gleichzeitig in der Schablone gesichert. Zur Wiederholung des Befehls müssen Sie nur die Editiertasten <COPYALL> und <CR> benutzen.

Der wiederholte Befehl wird während der Eingabe am Bildschirm angezeigt:

<COPYALL>DIR PROG.COM <CR>

Durch Betätigung der <COPYALL> Taste wird der Inhalt der Schablone in die Befehlszeile kopiert. Durch Betätigung von <CR> wird die Befehlszeile zur Ausführung an den Befehlsprozessor gesendet.

Möchte der Benutzer Informationen über eine Datei namens PROG.ASM, so kann er den Inhalt der Schablone benutzen und:

<COPYALL>C

eingeben. Durch Eingabe von <COPYALL>C werden sämtliche Zeichen aus der Schablone bis zu, jedoch nicht einschließlich "C" in die Befehlszeile kopiert. MS-DOS zeigt

DIR PROG._

an.

Das Unterstreichungszeichen stellt die Schreibmarke dar. Nun geben Sie

.ASM

ein. Das Ergebnis lautet:

DIR PROG.ASM_

Die Befehlszeile "DIR PROG.ASM" steht nun in der Schablone und kann zur Ausführung an den Befehlsprozessor gesendet werden. Zu diesem Zweck betätigen Sie <CR>.

Nun soll folgender Befehl ausgeführt werden:

TYPE PROG.ASM

Zu diesem Zweck geben Sie

TYPE <INSERT> <COPYALL> <RETURN>

ein. Die Zeichen werden von der Tastatur direkt in die Befehlszeile eingegeben, wobei Zeichen in der Schablone überschrieben werden. Diese automatische Ersatzeinrichtung wird ausgeschaltet, wenn die Einfügetaste betätigt wird. So ersetzen die Zeichen "TYPE" die Zeichen "DIR" in der Schablone. Um zwischen "TYPE" und "PROG.ASM" ein Leerzeichen einzufügen, müssen Sie <INSERT> und danach die Leertaste betätigen. Um schließlich den Rest der Schablone in die Befehlszeile zu kopieren, betätigen Sie <COPYALL> und danach <CR>. Der Befehl TYPE PROG.ASM wird von MS-DOS verarbeitet. Die Schablone besteht nun aus "TYPE PROG.ASM".

Wenn Sie "TYPE" versehentlich als "BYTE" eingeben, so kommt es zu einem Befehlsfehler. Bevor Sie jedoch den ganzen Befehl löschen, sollten Sie die falsch buchstabierte Zeile vor Betätigung von <CR> sichern, indem Sie mit der <NEWLINE> Taste eine neue Schablone erstellen:

BYTE PROG.ASM<NEWLINE>

Sie können dann den fehlerhaften Befehl durch folgende Eingabe aufbereiten:

T<COPY1>P<COPYALL>

Die <COPY1> Taste kopiert ein einziges Zeichen aus der Schablone in die Befehlszeile. Die sich ergebende Befehlszeile entspricht dann dem gewünschten Befehl:

TYPE PROG.ASM

Eine andere Möglichkeit wäre, dieselbe Schablone mit BYTE PROG.ASM zu benutzen. Danach werden die <SKIP1> und <INSERT> Tasten betätigt, um dasselbe Ergebnis zu erzielen:

<SKIP1> <SKIP1> <COPY1> <INSERT>YP<COPYALL>

Die folgende Aufstellung verdeutlicht, wie sich diese Eingabe auf die Befehlszeile auswirkt:

<SKIP1>	—	Überspringt das erste Zeichen in der Schablone.
<SKIP1>	—	Überspringt das zweite Zeichen in der Schablone.
<COPY1>	T	Kopiert das dritte Zeichen in der Schablone.
<INSERT>YP	TYP	Fügt zwei Zeichen ein.
<COPYALL>	TYPE PROG.ASM	Kopiert den Rest der Schablone.

Hier sollte beachtet werden, daß <SKIP1> die Befehlszeile nicht beeinflußt. Sie wirkt sich auf die Schablone aus, indem das erste Zeichen gelöscht wird. Gleichermaßen löscht <SKIPUP> Zeichen in der Schablone bis zu einem bestimmten Zeichen, dieses jedoch nicht eingeschlossen.

Diese besonderen Editiertasten gestalten die Eingabe über die Tastatur sehr viel wirkungsvoller. Im nächsten Abschnitt werden die Funktionen der Steuerzeichen beschrieben, die bei der Eingabe von Befehlen ebenfalls hilfreich sind.

FUNKTIONEN DER STEUERZEICHEN

Eine Steuerzeichenfunktion wirkt sich auf die Befehlszeile aus. <CONTROL-C> und <CONTROL-S> haben Sie bereits kennengelernt. Die anderen Funktionen der Steuerzeichen werden in der nachfolgenden Tabelle zusammengefaßt.

Hier sei nochmals daran erinnert, daß der Benutzer bei der Eingabe eines Steuerzeichens wie beispielsweise <CONTROL-C> die Kontrolltaste gedrückt halten und dann die "C"-Taste betätigen muß.

Steuerzeichen	Funktion
<CONTROL-C>	Beendet den gegenwärtigen Befehl.
<CONTROL-H>	Löscht das letzte Zeichen aus der Befehlszeile und das entsprechende Zeichen am Terminalbildschirm (wie Rücktaste).
<CONTROL-J> oder <CONTROL-N>	Fügt ein physisches Zeilenende ein, löscht die Befehlszeile jedoch nicht. Um die aktuelle logische Zeile über die phyischen Grenzen eines Terminalbildschirms hinaus zu erweitern, wird die <ZEILENVORSCHUB> Taste benutzt.
<CONTROL-P>	Leitet die Terminalausgabe an den Zeilendrucker weiter. Wird diese Taste erneut betätigt, so wird dieses Echo gelöscht.
<CONTROL-S>	Unterbricht die fortlaufende Ausgabe am Terminalbildschirm. Um die Ausgabe wieder fortzusetzen, wird eine beliebige Taste betätigt.
<CONTROL-X>	Löscht die aktuelle Zeile, löscht die Befehlszeile, gibt dann einen umgekehrten Schrägstrich (\), eine Zeilenschaltung und einen Zeilenvorschub aus. Die von diesen besonderen Editierbefehlen benutzte Schablone ist nicht betroffen.

WICHTIG: Wenn Sie während eines Druckvorgangs CONTROL-P oder CONTROL-N eingeben, ist eine einwandfreie Ausgabe der Druckdatei nicht mehr gewährleistet. Die Druckausgabe besteht dann aus dem Inhalt der Druckdatei und den von Ihnen an der Tastatur eingegebenen Zeichen. Aus diesem Grunde sollten Sie während des Druckvorgangs von CONTROL-P und CONTROL-N absehen.

Abb. 6.3 Funktionen der Steuerzeichen

DER ZEILEN-EDITOR (EDLIN)

ALLGEMEINE INFORMATION

In diesem Kapitel wird die Benutzung des Zeilen-Editors (EDLIN) beschrieben. Sie können EDLIN für das Erstellen, Ändern und Anzeigen von Dateien benutzen. Hierbei spielt es keine Rolle, ob es sich um Quellenprogramm- oder Textdateien handelt. Insbesondere kann EDLIN zur Ausführung der folgenden Funktionen benutzt werden:

- Erstellen neuer Quellendateien und Sichern dieser Quellendateien.
- Ändern der bestehenden Dateien und Sichern sowohl in der neuen Form als auch als Originaldatei.
- Löschen, Editieren, Einfügen und Anzeigen von Zeilen.
  Suche nach Text, sowie Löschen oder Ersetzen von Text innerhalb einer oder mehrerer Zeilen.

Der Text in Dateien, die mit EDLIN erstellt oder editiert werden, ist in Zeilen unterteilt. Jede Zeile kann eine Länge von bis zu 253 Zeichen haben. Von EDLIN werden während des Editiervorgangs Zeilennummern generiert und angezeigt. Diese Zeilennummern sind jedoch in der gesicherten Datei nicht wirklich vorhanden.

Wenn Sie Zeilen einfügen, werden sämtliche Zeilennummern im Anschluß an den eingefügten Text automatisch um die Anzahl der eingefügten Zeilen erhöht. Wenn Sie Zeilen in einer Datei löschen, werden sämtliche Zeilennummern im Anschluß an den gelöschten Text automatisch um die Anzahl der gelöschten Zeilen vermindert. Die Zeilen in der Datei sind also immer fortlaufend numeriert.

STARTEN VON EDLIN

Um EDLIN zu starten, wird:

EDLIN <Dateispez>

eingegeben. Wird eine neue Datei erstellt, so muß die <Dateispez> den Namen der zu erstellenden Datei darstellen. Findet EDLIN diese Datei nicht in einem Laufwerk, so erstellt er eine neue Datei

mit dem vom Benutzer angegebenen Namen. Folgende Meldung und Eingabeaufforderung werden angezeigt:

```
New file
(Neue Datei)
*
_
```

Die Eingabeaufforderung bei EDLIN ist ein Sternchen (*).

Nun können Textzeilen in die neue Datei eingegeben werden. Um Text eingeben zu können, müssen Sie einen I-Befehl (Insert) für das Einfügen von Zeilen eingeben. Der I-Befehl wird später in diesem Kapitel besprochen.

Soll eine bestehende Datei editiert werden, so muß <Dateispez> den Namen der zu editierenden Datei darstellen. Wenn EDLIN die angegebene Datei in dem bezeichneten Laufwerk oder dem Standardlaufwerk gefunden hat, wird die Datei in den Speicher geladen. Kann die gesamte Datei geladen werden, so zeigt EDLIN die folgende Meldung auf dem Bildschirm an:

```
End of input file
(Ende der Eingabedatei)
*
```

Die Datei kann mit den EDLIN Editierbefehlen aufbereitet werden.

Kann die Datei nicht ganz in den Speicher geladen werden, so lädt EDLIN Zeilen, bis der Speicher zu 3/4 gefüllt ist. Dann wird die Eingabeaufforderung (*) angezeigt. Nun können Sie den im Speicher befindlichen Teil der Datei aufbereiten.

Um den Rest der Datei aufbereiten zu können, müssen einige der editierten Zeilen auf Platte gesichert werden, damit Speicherplatz freigegeben wird. Danach kann EDLIN die nicht aufbereiteten Zeilen von der Platte in den Speicher laden. Eine Beschreibung dieses Verfahrens befindet sich in diesem Kapitel (Write- und Append-Befehle).

Nach Abschluß der Editor-Arbeiten können Sie die Originaldateien und die aktualisierten (neuen) Dateien mit Hilfe des End-Befehls sichern. Der End-Befehl wird in diesem Kapitel in dem Abschnitt "EDLIN-Befehle" beschrieben. Die Originaldatei wird

mit der Erweiterung .BAK umbenannt. Die neue Datei hat den in dem EDLIN-Befehl angegebenen Dateinamen und die angegebene Erweiterung. Die Originaldaei .BAK wird erst nach Abschluß der Editor-Arbeiten oder wenn der Editor (EDLIN) Speicherplatz auf der Platte benötigt gelöscht.

Eine Datei mit einer Dateinamenerweiterung von .BAK darf nicht editiert werden, da EDLIN davon ausgeht, daß jede .BAK-Datei eine Sicherungsdatei ist. Muß eine derartige Datei aufbereitet werden, so muß sie mit einer anderen Erweiterung umbenannt werden (wobei der in Kapitel 5 beschriebene MS-DOS Befehl RENAME zu benutzen ist). Danach können Sie EDLIN starten und die neue <Dateispez> angeben.

BESONDERE EDITIERTASTEN

Die in Kapitel 6 beschriebenen besonderen Editiertasten und die Schablone können für die Editierung der Textdateien benutzt werden. Diese Tasten werden im einzelnen in diesem Abschnitt beschrieben.

In Abb. 7.1 werden die Befehle, Codes und Funktionen zusammengefaßt. Im Anschluß an die Abbildung werden die besonderen Editiertasten beschrieben.

Funktion	Taste(n)*	Beschreibung
Kopieren eines Zeichens	<COPY1> ESC S	Kopiert ein Zeichen aus der Schablone in die neue Zeile.
Kopieren bis zu Zeichen	<COPYUP> ESC T	Kopiert sämtliche Zeichen aus der Schablone bis zu dem angegebenen Zeichen in die neue Zeile.
Kopieren der Schablone	<COPYALL> ESC U	Kopiert alle verbleibenden Zeichen in der Schablone auf den Bildschirm.
Überspringen eines Zeichens	<SKIP1> ESC V	Überspringt ein Zeichen, ohne es zu kopieren.
Überspringen bis zu Zeichen	<SKIPUP> ESC W	Überspringt die Zeichen in der Schablone bis zu dem angegeben Zeichen.
Beenden der Eingabe	<VOID> ESC E	Beendet die aktuelle Eingabe; beläßt die Schablone unverändert.
Löschen einer Zeile	<KILL> ESC J	Ersetzt den Inhalt der Schablone durch die neue Eingabe.
Einfüge-Modus	<INSERT> ESC P	Leitet den Einfüge-Modus ein.
Ersatz-Modus	<REPLACE> ESC Q	Schaltet den Einfüge-Modus ab. Dies ist die Standardvorgabe.
Neue Schablone	<NEWLINE> ↵	Macht die neue Zeile zur neuen Schablone.

*Die meisten Funktionen erfordern die Betätigung von zwei Tasten, die aber nicht gleichzeitig gedrückt werden dürfen.

Abb. 7.1 Besondere Editiertasten

<COPY1>

TASTE

ESC S

ZWECK

Kopiert ein Zeichen aus der Schablone in die Befehlszeile.

KOMMENTARE

Durch Betätigung der <COPY1> Taste wird ein Zeichen aus der Schablone in die Befehlszeile kopiert. Wird die <COPY1> Taste betätigt, so wird ein Zeichen in die Befehlszeile eingefügt. Der Einfügemodus wird automatisch ausgeschaltet.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei.
1:*_
```

Am Anfang der Editor-Arbeiten wird die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) an den Anfang der Zeile gesetzt. Nach Betätigung der <COPY1> Taste wird das erste Zeichen (D) in die zweite der beiden angezeigten Zeilen kopiert:

```
                1:*Dies ist eine Beispieldatei.
<COPY1>         1:*D_
```

Nach jeder Betätigung der <COPY1> Taste, wird ein weiteres Zeichen angezeigt:

```
<COPY1> 1:*Di_
<COPY1> 1:*Die_
<COPY1> 1:*Dies_
```

<COPYUP>

TASTE

ESC T

ZWECK

Kopiert mehrere Zeichen bis zu einem bestimmten Zeichen.

KOMMENTARE

Nach Betätigung der <COPYUP> Taste werden sämtliche Zeichen bis zu einem bestimmten Zeichen aus der Schablone in die Befehslzeile kopiert. Bei diesem Zeichen handelt es sich um das Zeichen, das im Anschluß an <COPYUP> eingegeben wird. Dieses Zeichen wird weder kopiert noch am Bildschirm angezeigt. Nach Betätigung der <COPYUP> Taste wird die Schreibmarke zu dem einzelnen Zeichen bewegt, das Sie in dem Befehl angegeben haben. Enthält die Schablone das angegebene Zeichen nicht, so findet kein Kopiervorgang statt. Durch Betätigung von <COPYUP> wird der Einfügemodus automatisch ausgeschaltet.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei.
1.*_
```

Am Anfang des Editiervorgangs wird die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) an den Anfang der Zeile gesetzt. Nach Betätigung der <COPYUP> Taste werden sämtliche Zeichen bis zu dem unmittelbar nach <COPYUP> eingegebenen Zeichen kopiert.

```
                    1:*Dies ist eine Beispieldatei
<COPYUP> n 1:*Dies ist ei_
```

<COPYALL>

TASTE

ESC U

ZWECK

Kopiert die Schablone in die Befehlszeile.

KOMMENTARE

Durch Betätigung der <COPYALL> Taste werden die verbleibenden Zeichen aus der Schablone in die Befehlszeile kopiert. Ungeachtet der Position der Schreibmarke zum Zeitpunkt der Betätigung der <COPYALL> Taste, wird der Rest der Zeile angezeigt. Die Schreibmarke wird dann hinter das letzte Zeichen in der Zeile versetzt.

BEISPIELE

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei.
1:*_
```

Am Anfang des Editiervorgangs wird die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) an den Anfang der Zeile gesetzt. Nach Betätigung der <COPYALL> Taste werden sämtliche Zeichen aus der Schablone (die hier in der oberen angezeigten Zeile dargestellt werden) in die Zeile mit der Schreibmarke (die untere angezeigte Zeile) kopiert:

```
                1:*Dies ist eine Beispieldatei (Schablone)
<COPYALL>       1:*Dies ist eine Beispieldatei. (Befehlszeile)
```

Auch hier wird der Einfügemodus automatisch ausgeschaltet.

<SKIP1>

TASTE

ESC V

ZWECK

Überspringt ein Zeichen in der Schablone.

KOMMENTARE

Durch Betätigung der <SKIP1> Taste wird ein Zeichen in der Schablone übersprungen. Immer wenn die <SKIP1> Taste betätit wird, wird ein Zeichen nicht aus der Schablone kopiert. Die <SKIP1> Taste entspricht in ihrer Wirkung in etwa der <COPY1> Taste. Allerdings wird bei <SKIP1> ein Zeichen in der Schablone übersprungen, anstatt in die Befehlszeile kopiert zu werden.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei.
1:*_
```

Am Anfang des Editiervorgangs wird die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) an den Anfang der Zeile gesetzt. Durch Betätigung der <SKIP1> Taste wird das erste Zeichen ("D") übersprungen.

```
          1:*Dies ist eine Beispieldatei
<SKIP1>   1:*_
```

Die Position der Schreibmarke bleibt unverändert. Nur die Schablone ist betroffen. Um festzustellen, wieviele Zeichen der Zeile übersprungen wurden, wird die <COPYALL> Taste betätigt, die die Schreibmarke hinter das letzte Zeichen der Zeile setzt.

```
            1:*Dies ist eine Beispieldatei
<SKIP1>     1:*_
<COPYALL>   1:*ies ist eine Beispieldatei._
```

<SKIPUP>

TASTE

ESC W

ZWECK

Überspringt mehrere Zeichen in der Schablone bis zu dem angegebenen Zeichen.

KOMMENTARE

Durch Betätigung der <SKIPUP> Taste können Sie sämtliche Zeichen bis zu einem bestimmten Zeichen in der Schablone überspringen. Dieses Zeichen wird nicht kopiert und nicht am Bildschirm angezeigt. Enthält die Schablone das angegebene Zeichen nicht, so findet kein Überspringvorgang statt. Die <SKIPUP> Taste ist in ihrer Wirkung der <COPYUP> Taste ähnlich, nur werden bei <SKIPUP> Zeichen in der Schablone übersprungen, anstatt in die Befehslzeile kopiert zu werden.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei
1:*_
```

Am Anfang des Editiervorgangs wird die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) an den Anfang der Zeile gesetzt. Durch Betätigung der <SKIPUP> Taste werden sämtliche Zeichen in der Schablone bis zu dem Zeichen übersprungen, das direkt nach <SKIPUP> eingegeben wird:

```
                1:*Dies ist eine Beispieldatei
<SKIPUP> b      1:*_
```

Die Position der Schreibmarke wird nicht verändert. Um festzustellen, wieviele Zeichen der Zeile übersprungen wurden, betätigen Sie die <COPYALL> Taste, um die Schablone zu kopieren. Dadurch wird die Schreibmarke hinter das letzte Zeichen in der Zeile versetzt:

```
                1:*Dies ist eine Beispieldatei
<SKIPUP> b      1:*_
<COPYALL>       1:*Dies ist eine._
```

<VOID>

TASTE

ESC E

ZWECK

Beendet die Eingabe und löscht die Befehlszeile.

KOMMENTARE

Durch Betätigung der <VOID> Taste können Sie die Befehlszeile löschen. Die Schablone jedoch bleibt unverändert. Durch <VOID> wird außerdem ein umgekehrter Schrägstrich (\) ausgedruckt. Ferner erfolgen Carriage Return und Line Feed. Der Einfügemodus wird ausgeschaltet. Die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) wird an den Anfang der Zeile gesetzt. Durch Betätigung der <COPYALL> Taste wird die Schablone in die Befehlszeile kopiert. Die Befehlszeile wird in der Form angezeigt, die sie vor Betätigung der <VOID> Taste aufwies.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei
1:*_
```

Am Anfang des Editiervorgangs wird die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) an den Anfang der Zeile gesetzt. Hier soll davon ausgegangen werden, daß die Zeile durch "Beispieldatei" ersetzt werden soll:

```
1:*Dies ist eine Beispieldatei.
1:*Beispieldatei_
```

Um die gerade eingegebene Zeile (Beispieldatei) zu löschen und um "Dies ist eine Beispieldatei." beizubehalten, wird <VOID> betätigt. In der Zeile mit "Beispieldatei" erscheint ein umgekehrter Schrägstrich, um dem Benutzer mitzuteilen, daß diese Zeile gelöscht ist.

```
          1:*Dies ist eine Beispieldatei
<VOID>    1:*Beispieldatei
          1:_
```

Um die Originalzeile beizubehalten oder um andere Editierfunktionen auszuführen, wird die <CR> Taste betätigt. Wenn Sie <COPYALL> eingeben, wird die Originalschablone in die Befehlszeile kopiert:

```
<COPYALL> 1: Dies ist eine Beispieldatei._
```

<INSERT>

TASTE

ESC P

ZWECK

Leitet den Einfügemodus ein.

KOMMENTARE

Durch Betätigung der <INSERT> Taste geht EDLIN in den Einfügemodus. Die gegenwärtige Position der Schreibmarke in der Schablone wird nicht verändert. Die Schreibmarke bewegt sich mit jedem eingefügten Zeichen. Nach Beendigung der Einfügung von Zeichen wird die Schreibmarke jedoch wieder zu dem Zeichen bewegt, auf dem es vor der Einfügung stand. So werden Zeichen vor dem Zeichen eingefügt, auf das die Schreibmarke zeigt.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei.
1:*_
```

Am Anfang des Editiervorgangs wird die Schreibmarke (sie ist von dem Unterstreichungszeichen dargestellt) an den Anfang der Zeile bewegt. Als Beispiel betätigen Sie die Tasten <COPYUP> und "b":

```
                1:*Dies ist eine Beispieldatei.
<COPYUP> b      1:*Dies ist eine_
```

Nun betätigen Sie die <INSERT> Taste. Danach werden die Zeichen "aufbereitete" und ein Leerzeichen eingefügt:

```
                                1:*Dies ist eine Beispieldatei.
<COPYUP> b                      1:*Dies ist eine_
<COPYUP> aufbereitete           1:*Dies ist eine aufbereitete_
```

Wird nun die <COPYALL> Taste betätigt, so wird der Rest der Schablone in die Zeile kopiert:

```
                1:*Dies ist eine aufbereitete
<COPYALL>       1:*Dies ist eine aufbereitete Beispieldatei._
```

Wenn Sie die <CR> Taste betätigen, wird der Rest der Schablone abgeschnitten. Die Befehlszeile endet dann mit dem Ende der Einfügung:

<INSERT> aufbereitete<CR> 1:*Dies ist eine aufbereitete _

<EXIT>

TASTE

ESC Q

ZWECK

Leitet den Ersatzmodus ein.

KOMMENTARE

Durch Betätigung der <EXIT> Taste verläßt EDLIN den Einfügemodus und geht in den Ersatzmodus. Sämtliche nun eingegebene Zeichen überschreiben und ersetzen die Zeichen in der Schablone. Wenn Sie mit dem Editieren einer Zeile beginnen, ist der Ersatzmodus wirksam. Wird die <CR> Taste betätigt, so wird der Rest der Schablone gelöscht.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei
1:*_
```

Am Anfang des Editiervorgangs befindet sich die Schreibmarke (, die durch das Unterstreichungszeichen dargestellt wird,) am Beginn der Zeile. Hier soll nun davon ausgegangen werden, daß <COPYUP> b, <INSERT> aufbereitete, <EXIT> Text und danach <COPYALL> eingegeben wird:

```
                        1:*Dies ist eine Beispieldatei
<COPYUP> b              1:*Dies ist eine _
<INSERT> aufbereitete   1:*Dies ist eine aufbereitete _
<EXIT> Text             1:*Dies ist eine aufbereitete Text_
<COPYALL>               1:*Dies ist eine aufbereitete Textda-
                          tei._
```

In diesem Beispiel wurde "aufbereitete" eingefügt. "Beispiel" wurde durch "Text" ersetzt. Wenn Sie Zeichen über die Länge der Schablone hinaus eingeben, werden die restlichen Zeichen in der Schablone automatisch angehängt, sobald <COPYALL> betätigt wird.

<NEWLINE>

TASTE

↵

ZWECK

Erstellt eine neue Schablone.

KOMMENTARE

Durch Betätigung der <NEWLINE> Taste wird die gegenwärtige Befehlszeile in die Schablone kopiert. Der Inhalt der alten Schablone wird gelöscht. Nach Betätigung der <NEWLINE> Taste wird ein @-Zeichen ausgegeben. Außerdem erfolgen Carriage Return und Line Feed. Ferner wird die Befehlszeile gelöscht und der Einfügemodus ausgeschaltet.

HINWEIS: <NEWLINE> führt dieselbe Funktion wie die <VOID> Taste aus, nur wird im ersteren Fall die Schablone geändert und ein @-Zeichen anstelle eines umgekehrten Schrägstrichs (\) ausgedruckt.

BEISPIEL

Hier soll von folgender Bildschirmanzeige ausgegangen werden:

```
1:*Dies ist eine Beispieldatei
1:*_
```

Am Anfang des Editiervorgangs steht die Schreibmarke (sie wird von dem Unterstreichungszeichen dargestellt) am Beginn der Zeile. Gehen wir davon aus, daß <COPYUP> b, <INSERT> aufbereitete, <REPLACE> Text und danach <COPYALL> eingegeben werden:

	1:*Dies ist eine Beispieldatei
<COPYUP> b	1:*Dies ist eine _
<INSERT> aufbereitete	1:*Dies ist eine aufbereitete _
<REPLACE> Text	1:*Dies ist eine aufbereitete Text_
<COPYALL>	1:*Dies ist eine aufbereitete Textdatei._

Hier soll nun die Zeile zur neuen Schablone werden. Drücken Sie deshalb die <NEWLINE> Taste.

```
<NEWLINE> 1:*Dies ist eine aufbereitete Textdatei. @
```

Das @ gibt an, daß diese neue Zeile nun die neue Schablone ist. Mit Hilfe dieser neuen Schablone können Sie weitere Editierarbeiten ausführen.

EDLIN-BEFEHLE

In diesem Abschnitt werden die einzelnen EDLIN-Befehle beschrieben, die Editierfunktionen bei Textzeilen durchführen. Bevor ein EDLIN-Befehl benutzt wird, sollten Sie die Konventionen und Optionen für sämtliche Befehle durchlesen.

FORMATKONVENTIONEN

1. Pfadnamen können als Optionen für Befehle benutzt werden. Durch Eingabe von EDLIN\BIN\USER\JOE\TEXT.TXT kann beispielsweise die TEXT.TXT-Datei in dem Unterverzeichnis JOE aufbereitet werden.

2. Ausgehend von der aktuellen Zeile (, der Zeile mit dem Sternchen,) können Sie einzelne Zeilen bearbeiten. Zur Angabe von Zeilen vor der aktuellen Zeile ist ein Minuszeichen mit einer Zahl zu benutzen. Zur Angabe von Zeilen hinter der aktuellen Zeile ist ein Pluszeichen mit einer Zahl zu benutzen.

   Beispiel:

   -10, +10L

   Durch diesen Befehl wird ab der zehnten Zeile vor der aktuellen Zeile bis einschließlich zur zehnten Zeile hinter der aktuellen Zeile aufgelistet.

3. Sie können mehrere Befehle in einer Befehlszeile erteilen. Wenn Sie einen Befehl zur Aufbereitung einer einzelnen Zeile mit Hilfe einer Zeilennumer (<ZEILE>) ausgeben, müssen die Befehle in der Zeile jeweils durch einen Strichpunkt voneinander getrennt werden. Ansonsten können die Befehle ohne besondere Trennzeichen aufeinander folgen. Bei einem Search- oder Replace-Befehl können Sie die <Zeichenfolge> mit <CONTROL-Z> anstatt mit <CR> beenden.

   Beispiel:

   15;-5,+5L

   Die Befehlszeile in dem nächsten Beispiel sucht nach "Diese Zeichenfolge". Danach werden fünf Zeilen vor und fünf Zeilen hinter der Zeile mit der übereinstimmenden Zeichenfolge

angezeigt. Verläuft die Suche nicht erfolgreich, so handelt es sich bei den angezeigten Zeilen um die Zeilennummern ausgehend von der aktuellen Zeile.

SDiese Zeichenfolge <CONTROL-Z> -5, +L

4. Sie dürfen EDLIN-Befehle mit oder ohne Leerzeichen zwischen der Zeilennummer und dem Befehl eingeben. Soll beispielsweise Zeile 6 gelöscht werden, so ist der Befehl 6D gleichbedeutend mit dem Befehl 6 D.

5. Ein Steuerzeichen (wie beispielsweise CONTROL-C) kann in den Text eingefügt werden, indem das Anführungszeichen CONTROL-V während der Arbeit im Einfügemodus vor diesem Zeichen benutzt wird. Durch CONTROL-V wird MS-DOS angewiesen, den nächsten eingegebenen Großbuchstaben als Steuerzeichen zu erkennen. Außerdem können Sie ein Steuerzeichen in jedem der Zeichenfolgenparameter von Search oder Replace benutzen, indem Sie das Sonder-Anführungszeichen einsetzen. Zum Beispiel:

   S<CONTROL-V>Z
   sucht das erste Auftreten von
   CONTROL-Z in einer Datei

   R<CONTROL-V>Z<CONTROL-Z>foo
   ersetzt jedes Auftreten von
   CONTROL-Z in einer Datei durch foo

   S<CONTROL-V>C<CONTROL-Z>bar
   ersetzt jedes Auftreten von
   CONTROL-C durch bar

   CONTROL-V kann durch Eingabe von CONTROL-V-V in den Text eingefügt werden.

6. Mit dem CONTROL-Z Zeichen teilen Sie EDLIN normalerweise mit, "dies ist das Ende der Datei". Stehen CONTROL-Z Zeichen an anderer Stelle in der Datei, so muß EDLIN mitgeteilt werden, daß diese anderen Steuerzeichen nicht "Dateiende" bedeuten. Mit dem /B Schalter wird EDLIN angewiesen, sämtliche CONTROL-Z Zeichen in der Datei zu ignorieren und die gesamte Datei anzuzeigen.

Die folgende Abbildung faßt die EDLIN-Befehle zusammen. Sie werden im Anschluß an die Beschreibung der Befehlsoptionen genauer erläutert.

Befehl	Zweck
<Zeile>	Editiert die Zeilennummer
A	Hängt Zeilen an
C	Kopiert Zeilen
D	Löscht Zeilen
E	Beendet den Editiervorgang
I	Fügt Zeilen ein
L	Listet Text auf
M	Verschiebt Zeilen
P	Speichert Text seitenweise
Q	Verläßt den Editiervorgang
R	Ersetzt Zeilen
S	Sucht Text ab
T	Überträgt Text
W	Schreibt Zeilen

Abb. 7.2 EDLIN-Befehle

BEFEHLSOPTIONEN

Mehrere EDLIN-Befehle lassen eine oder mehrere Optionen zu. Die Auswirkung einer Befehlsoption ist unterschiedlich, je nachdem, mit welchem Befehl sie benutzt wird. Die nachfolgende Liste beschreibt jede Option.

<Zeile>

<Zeile> gibt eine vom Benutzer eingegebene Zeilennummer an. Zeilennummern müssen durch ein Komma oder ein Leerzeichen von anderen Zeilennummern, anderen Optionen und von dem Befehl getrennt werden.

Für die Angabe von <Zeile> stehen Ihnen drei Möglichkeiten zur Verfügung:

- Zahl (n). Jede beliebige Zahl, die kleiner ist als 65534 — Wird eine größere Zahl als die größte bestehende Zeilennummer angegeben, so bezieht sich <Zeile> auf die Zeile hinter der letzten bisher existierenden Zeilennummer.
- Punkt (.) — Wird ein Punkt für <Zeile> angegeben, so bezieht sich <Zeile> auf die aktuelle Zeilennummer. Die aktuelle Zeile ist die zuletzt aufbereitete Zeile. Hier handelt es sich nicht unbedingt um die letzte angezeigte Zeile. Die aktuelle Zeile wird am Bildschirm durch ein Sternchen (*) zwischen der Zeilennummer und dem ersten Zeichen gekennzeichnet.

- Doppelkreuz (#) — Das Doppelkreuz zeigt die Zeile hinter der letzten Zeilennummer an. Wenn Sie # für <Zeile> angeben, ist dies gleichbedeutend mit der Angabe einer Zahl, die größer als die der letzten Zeilennummer ist.

<CR>

Falls Sie Carriage Return ohne eine Angabe für <Zeile> betätigen, wird EDLIN angewiesen, einen dem Befehl entsprechenden Standardwert zu benutzen.

?

Mit dem Fragezeichen wird EDLIN angewiesen, den Benutzer zu fragen, ob die richtige Zeichenfolge gefunden wurde. Das Fragezeichen wird nur mit den Replace- und Search-Befehlen benutzt. Bevor EDLIN mit der Aufbereitung fortfährt, wartet es entweder auf die Eingabe von Y oder <CR> für eine bejahende Antwort oder auf die Eingabe eines beliebigen anderen Zeichens für eine verneinende Antwort.

<Zeichenfolge>

<Zeichenfolge> stellt zu suchende, zu ersetzende oder Ersatztexte dar. Die Option <Zeichenfolge> wird nur mit den Search- und Replace-Befehlen benutzt. Jede <Zeichenfolge> ist durch <CONTROL-Z> oder <CR> abzuschließen. Die Beschreibung des Replace-Befehls gibt weitere Einzelheiten an. Zwischen Zeichenfolgen oder zwischen einer Zeichenfolge und dem Befehlsbuchstaben dürfen keine Leerzeichen stehen, es sei denn, diese Leerzeichen sollen Teil der Zeichenfolge sein.

(A)PPEND

NAME

Append

ZWECK

Fügt die angegebene Anzahl von Zeilen von der Platte der gerade im Speicher aufbereiteten Datei hinzu. Die Zeilen werden an das Ende der Zeilen angefügt, die gerade im Speicher vorhanden sind.

SYNTAX

[<n>] A

KOMMENTARE

Dieser Befehl ist nur von Bedeutung, wenn die gerade aufbereitete Datei für den Maschinenspeicher zu groß ist. Beim Starten von EDLIN werden so viele Zeilen wie möglich zur Aufbereitung in den Speicher eingelesen.

Um den nicht in den Speicher passenden Rest der Datei aufzubereiten, müssen Sie bereits aufbereitete Zeilen auf Platte schreiben. Danach können die noch nicht aufbereiteten Zeilen mit dem Append-Befehl von der Platte in den Speicher geladen werden. (In der Beschreibung des Write-Befehls in diesem Kapitel wird erläutert, wie Sie aufbereitete Zeilen auf Platte schreiben können.)

Wird die Anzahl von anzuhängenden Zeilen nicht angegeben, so werden so langen Zeilen angefügt, bis der zur Verfügung stehende Speicherplatz zu drei Vierteln gefüllt ist. Ist der zur Verfügung stehende Speicherplatz schon zu drei Vierteln gefüllt, so werden keine Zeilen angehängt.

Nachdem der Append-Befehl die letzte Zeile der Datei in den Speicher eingelesen hat, wird die Meldung "End of input file" (Ende der Eingabedatei) angezeigt.

NAME

Copy

ZWECK

Kopiert eine Reihe von Zeilen an eine angegebene Zeilennummer. Mit Hilfe der Option <Zähler> können die Zeilen beliebig oft kopiert werden.

SYNTAX

[<Zeile>], [<Zeile>], <Zeile>, [<Zähler>] C

KOMMENTARE

Wenn Sie in <Zähler> keine Zahl angeben, kopiert EDLIN die Zeilen einmal. Sollten Sie die erste oder die zweite <Zeile> weglassen, so wird standardmäßig die aktuelle Zeile genommen. Nach dem Kopiervorgang wird die Datei automatisch neu numeriert.

Die Zeilennummern dürfen sich nicht überlappen, ansonsten kommt es zu einer Meldung "Entry error" (Eingabefehler). 3,20, 15C würde beispielsweise zu einer Fehlermeldung führen.

BEISPIELE

Hier soll davon ausgegangen werden, daß folgende Datei vorhanden ist und aufbereitet werden soll:

```
1: Dies ist eine Beispieldatei,
2: mit der das Kopieren von Zeilen gezeigt wird.
3: Sehen Sie, was geschieht, wenn
4: der Copy-Befehl
5: (der C-Befehl) benutzt wird,
6: um Text in der Datei zu kopieren.
```

Diesen gesamten Textblock können Sie durch Ausgabe des folgenden Befehls kopieren:

```
1,6,7C
```

Das Ergebnis:

```
1: Dies ist eine Beispieldatei,
2: mit der das Kopieren von Zeilen gezeigt wird.
```

```
   3: Sehen Sie, was geschieht, wenn
   4: der Copy-Befehl
   5: (der C-Befehl) benutzt wird,
   6: um Text in der Datei zu kopieren.
   7: Dies ist eine Beispieldatei,
   8: mit der das Kopieren von Zeilen gezeigt wird.
   9: Sehen Sie, was geschieht, wenn
  10: der Copy-Befehl
  11: (der C-Befehl) benutzt wird,
  12: um Text in der Datei zu kopieren.
```

Soll Text in anderen Text eingefügt werden, so muß in der dritten Option <Zeile> die Zeile angegeben werden, vor der der kopierte Text stehen soll. Angenommen, es sollen Zeilen kopiert und in die folgende Datei eingefügt werden:

```
   1: Dies ist eine Beispieldatei,
   2: mit der das Kopieren von Zeilen gezeigt wird.
   3: Sehen Sie, was geschieht, wenn
   4: der Copy-Befehl
   5: (der C-Befehl) benutzt wird,
   6: um Text in der Datei zu kopieren.
   7: COPY kann auch benutzt werden,
   8: um Textzeilen in die
   9: Mitte der Datei zu kopieren.
  10: Ende der Beispieldatei.
```

Der Befehl 3,6,9C führt zu folgender Datei:

```
   1: Dies ist eine Beispieldatei,
   2: mit der das Kopieren von Zeilen gezeigt wird.
   3: Sehen Sie, was geschieht, wenn
   4: der Copy-Befehl
   5: (der C-Befehl) benutzt wird,
   6: um Text in der Datei zu kopieren.
   7: COPY kann auch benutzt werden,
   8: um Textzielen in die
   9: Mitte der Datei zu kopieren.
  10: Sehen Sie, was geschieht, wenn
  11: der Copy Befehl
  12: (der C-Befehl) benutzt wird,
  13: um Text in der Datei zu kopieren.
  14: Ende der Beispieldatei.
```

NAME

Delete

ZWECK

Löscht einen bestimmten Bereich von Zeilen in einer Datei.

SYNTAX

[<Zeile>] [,<Zeile>] D

KOMMENTARE

Wenn Sie die erste Option <Zeile> weglassen, so wird standardmäßig die aktuelle Zeile genommen (d.h. die Zeile mit dem Sternchen neben der Zeilennummer). Wenn Sie die zweite Option <Zeile> weglassen, so wird nur die erste <Zeile> gelöscht. Nachdem Zeilen gelöscht wurden, wird die Zeile unmittelbar hinter dem gelöschten Bereich zur aktuellen Zeile. Sie erhält die Zeilennummer, die die erste gelöschte <Zeile> vor dem Löschen hatte.

BEISPIELE

Hier wird davon ausgegangen, daß folgende Datei vorhanden ist und aufbereitet werden soll:

```
 1:  Dies ist eine Beispieldatei,
 2:  mit der dynamische Zeilennummern gezeigt werden sollen.
 3:  Sehen Sie, was geschieht, wenn Sie
 4:  Delete und Insert
 .
 .
 .
25:  (die D- und I-Befehle) benutzen,
26:  um den Text
27:* in Ihrer Datei aufzubereiten.
```

Für das Löschen mehrerer Zeilen müssen Sie <Zeile>,<Zeile> D eingeben:

```
5,24D
```

Das Ergebnis:

```
1:  Dies ist eine Beispieldatei,
2:  mit der das Kopieren von Zeilen gezeigt wird.
```

```
3:  Sehen Sie, was geschieht, wenn Sie
4:  Delete und Insert
5:  (die D- und I-Befehle) benutzen,
6:  um den Text
7:* in Ihrer Datei aufzubereiten.
```

Sie können eine einzelne Zeile löschen:

```
6D
```

Das Ergebnis:

```
1:  Dies ist eine Beispieldatei,
2:  mit der dynamische Zeilennummern gezeigt werden.
3:  Sehen Sie, was geschieht, wenn Sie
4:  Delete und Insert
5:  (die D- und I-Befehle) benutzen,
6:* in Ihrer Datei aufzubereiten
```

Sie können ebenfalls einen Bereich von Zeilen aus einer Datei löschen:

```
1:  Dies ist eine Beispieldatei,
2:  mit der dynamische Zeilennummern gezeigt werden sollen.
3:* Sehen Sie, was geschieht, wenn Sie
4:  Delete und Insert
5:  (die D- und I-Befehle) benutzen,
6:  um den Text
7:  in Ihrer Datei aufzubereiten.
```

Um einen Bereich von Zeilen ab der aktuellen Zeile zu löschen, geben Sie

```
,6D
```

ein. Das Ergebnis sieht folgendermaßen aus:

```
1:  Dies ist eine Beispieldatei,
2:  mit der dynamische Zeilennummern gezeigt werden sollen.
3:* in Ihrer Datei.
```

Die Zeilen werden automatisch neu numeriert.

<Zeile> EDIT

NAME

Edit

ZWECK

Editiert Textzeilen.

SYNTAX

[<Zeile>]

KOMMENTARE

Wird eine Zeilennummer eingegeben, so zeigt EDLIN die Zeilennummer und den Text an. Auf der nachfolgenden Zeile druckt EDLIN die Zeilennummer dann erneut aus. Sie können nun die Zeile editieren. Für das Editieren der Zeile kann jeder der EDLIN-Editierbefehle benutzt werden. Der bestehende Text der Zeile wird als Schablone benutzt, bis Sie die <CS> Taste betätigen.

Wenn Sie keine Zeilennummer eingeben (d.h. wenn Sie nur die <CR> Taste betätigen, wird die Zeile nach der aktuellen Zeile (die durch ein Sternchen gekennzeichnet ist) editiert. Sind an der aktuellen Zeile keine Änderungen erforderlich und steht die Schreibmarke am Anfang oder Ende der Zeile, so müssen Sie nur die <CR> Taste betätigen, um die Zeile unverändert zu belassen.

ACHTUNG

Wenn Sie die <CR> Taste betätigen, während die Schreibmarke in der Mitte der Zeile steht, wird der Rest der Zeile gelöscht.

BEISPIEL

Hier wird davon ausgegangen, daß folgende Datei vorhanden ist und aufbereitet werden soll:

```
1: Dies ist eine Beispieldatei,
2: mit der das
3: Editieren von
4:* Zeile vier gezeigt werden soll.
```

Um Zeile 4 zu editieren, geben Sie einfach

```
4
```

ein.

Der Inhalt der Zeile erscheint, wobei die Schreibmarke unter der Zeile steht:

```
4:* Zeile vier gezeigt werden soll
4:*_
```

Nun können Sie folgende Eingabe mit der <COPYALL> Editiertaste vornehmen:

```
<INSERT> Zahl      4:  Zahl-
<COPYALL> <CR>     4:  Zahl vier.
                   5:*_
```

(E)ND

NAME

End

ZWECK

Beendet die Editionsarbeiten.

SYNTAX

E

KOMMENTARE

Dieser Befehl sichert die editierte Datei auf Platte, benennt die Originaleingabedatei in <Dateiname>.BAK um und verläßt dann EDLIN. Bei einer Neuerstellung einer Datei während des Editiervorgangs wird keine .BAK-Datei erstellt.

Bei dem E-Befehl sind keine Optionen zulässig. Deshalb kann EDLIN nicht mitgeteilt werden, auf welchem Laufwerk die Datei zu sichern ist. Das Laufwerk, auf dem die Datei gesichert werden soll, muß bei Beginn der Editorarbeiten ausgewählt werden. Wenn Sie das Laufwerk beim Start von EDLIN nicht ausdrücklich angeben, wird die Datei auf der Platte in dem Standardlaufwerk gesichert. Die Datei kann dann immer noch mit dem MS-DOS Befehl COPY auf ein anderes Laufwerk kopiert werden.

Der Benutzer muß sich vergewissern, daß die Platte über ausreichend Speicherplatz für die gesamte Datei verfügt. Verfügt die Platte nicht über ausreichend freien Speicherplatz, so wird der Schreibvorgang abgebrochen und die editierte Datei ist verloren, auch wenn ein Teil der Datei schon auf die Platte geschrieben ist.

BEISPIEL

```
E <CR>
```

Nach Ausführung des E-Befehls wird die Systemmeldung von MS-DOS (beispielsweise A>) angezeigt.

(I)NSERT

NAME

Insert

ZWECK

Fügt Text unmittelbar vor der angegebenen <Zeile> ein.

SYNTAX

[<Zeile>] I

KOMMENTARE

Beim Erstellen einer neuen Datei muß der I-Befehl angegeben werden, bevor der Text eingegeben (eingefügt) werden kann. Der Text beginnt mit der Zeilennummer 1. Nachfolgende Zeilennummern werden automatisch nach jeder Betätigung von <CR> angezeigt.

EDLIN bleibt im Einfügemodus, bis <CONTROL-C> eingegeben wird. Nach Abschluß der Einfügung und Verlassen des Einfügemodus wird die Zeile unmittelbar im Anschluß an die eingefügten Zeilen zur aktuellen Zeile. Sämtliche Zeilennummern, die auf den eingefügten Abschnitt folgen, werden um die Anzahl von eingefügten Zeilen erhöht.

Wird <Zeile> nicht angegeben, so wird standardmäßig die aktuelle Zeilennummer genommen. In diesem Fall werden die Zeilen unmittelbar vor der aktuellen Zeile eingefügt. Stellt <Zeile> eine größere Nummer als die letzte Zeilennummer dar, oder wird ein Doppelkreuz (#) als <Zeile> angegeben, so werden die eingefügten Zeilen an das Ende der Datei angehängt. In diesem Fall wird die letzte eingefügte Zeile zur aktuellen Zeile.

BEISPIELE

Hier wird davon ausgegangen, daß folgende Datei vorhanden ist und aufbereitet werden soll:

```
1: Dies ist eine Beispieldatei,
2: mit der dynamische Zeilennummern gezeigt werden sollen.
3: Sehen Sie, was geschieht, wenn Sie
4: Delete und Insert
5: (die D- und I-Befehle)
```

```
6:  für das Aufbereiten von Text
7:* in Ihrer Datei benutzen.
```

Um Text vor einer bestimmten Zeile einzufügen, bei der es sich nicht um die aktuelle Zeile handelt, geben Sie <Zeile> I ein:

```
7I  7I
```

Das Ergebnis:

```
7:_
```

Nun wird der neue Text für Zeile 7 eingegeben:

```
7:  und das Neunumerieren von Zeilen
```

Drücken Sie <Control-Z>, um die Einfügung zu beenden.

```
8:  <CONTROL-Z>
```

Die Eingabe L listet die Datei auf.

Das Ergebnis:

```
1:  Dies ist eine Beispieldatei,
2:  mit der dynamische Zeilennummern gezeigt werden sollen.
3:  Sehen Sie, was geschieht, wenn Sie
4:  Delete und Insert
5:  (die D- und I-Befehle)
6:  für das Aufbereiten von Text
7:  und das Neunumerieren von Zeilen
8:* in Ihrer Datei benutzen.
```

Um Zeilen unmittelbar vor der aktuellen Zeile einzufügen, wird

```
I
```

eingegeben.

Das Ergebnis:

```
8:  —
```

Nun können Sie den folgenden Text einfügen und mit <Control-Z> abschließen:

```
8: damit sie fortlaufend werden
9: <CONTROL-Z>
```

Um die Datei aufzulisten und das Ergebnis zu sehen, geben Sie nun L ein.

Das Ergebnis:

```
1: Dies ist eine Beispieldatei,
2: mit der dynamische Zeilennummern gezeigt werden sollen.
3: Sehen Sie, was geschieht, wenn Sie
4: Delete und Insert
5: (die D- und I-Befehle)
6: für das Aufbereiten von Text
7: und das Neunumerieren von Zeilen
8: damit sie fortlaufend werden
9:* in Ihrer Datei benutzen.
```

Um neue Zeilen an das Ende der Datei anzuhängen, ist

```
10I
```

einzugeben. Dies führt zu folgender Anzeige:

```
10:_
```

Nun können Sie die folgenden neuen Zeilen eingeben:

```
10: Der Insert-Befehl kann neue Zeilen
11: in die Datei setzen; dies ist problemlos,
12: da die Zeilennummern dynamisch sind;
13: sie gehen bis 65533.
```

Die Einfügung wird durch Betätigung von <CONTROL-Z> in Zeile 14 beendet. Die neuen Zeilen stehen am Ende aller vorhergehenden Zeilen in der Datei. Geben Sie nun den List-Befehl, L, ein:

Das Ergebnis:

```
1: Dies ist eine Beispieldatei,
2: mit der dynamische Zeilennummern gezeigt werden sollen.
3: Sehen Sie, was geschieht, wenn Sie
4: Delete und Insert
```

```
 5: (die D- und I-Befehle)
 6: für das Aufbereiten von Text
 7: und das Neunumerieren von Zeilen,
 8: damit sie fortlaufend werden,
 9: in Ihrer Datei benutzen.
10: Der Insert-Befehl kann neue Zeilen
11: in die Datei setzen; dies ist problemlos,
12: da die Zeilennummern dynamisch sind;
13: sie gehen bis 65533.
```

(L)IST

NAME

List

ZWECK

Listet einen Bereich von Zeilen, einschließlich der beiden angegebenen Zeilen, auf.

SYNTAX

[<Zeile>] [,<Zeile>] L

KOMMENTARE

Werden eine oder beide Optionen weggelassen, so werden Standardwerte geliefert. Wird die erste Option wie in folgendem Fall weggelassen:

,<Zeile> L

so beginnt die Anzeige elf Zeilen vor der aktuellen Zeile und endet mit der angegebenen <Zeile>. Das einleitende Komma ist erforderlich, da die erste Option weggelassen wurde.

HINWEIS: Steht die angegebene <Zeile> mehr als elf Zeilen vor der aktuellen Zeile, so sieht die Anzeige aus, als hätten Sie beide Optionen weggelassen.

Wenn Sie die zweite Option weglassen wie im folgenden Fall:

<Zeile> L

so werden 23 Zeilen angezeigt, wobei mit der angegebenen <Zeile> begonnen wird.

Werden beide Parameter weggelassen, wie bei:

L

so werden 23 Zeilen angezeigt: die elf Zeilen vor der aktuellen Zeile, die aktuelle Zeile und die elf Zeilen nach der aktuellen Zeile. Sind vor der aktuellen Zeile keine elf Zeilen vorhanden, so werden mehr als elf Zeilen nach der aktuellen Zeile angezeigt, um ein Gesamtergebnis von 23 Zeilen zu erzielen.

BEISPIELE

Hier wird davon ausgegangen, daß folgende Datei vorhanden ist und aufbereitet werden soll:

```
 1: Dies ist eine Beispieldatei,
 2: mit der dynamische Zeilennummern gezeigt werden sollen.
 3: Sehen Sie, was geschieht, wenn Sie
 4: Delete und Insert
 5: die (D- und I-Befehle)
 .
 .
 .
15:* Die aktuelle Zeile enthält ein Sternchen.
 .
 .
 .
26: für das Aufbereiten von Text
27: in Ihrer Datei benutzen.
```

Um einen Bereich von Zeilen ohne Bezugnahme auf die aktuelle Zeile aufzulisten, geben Sie <Zeile>,<Zeile> L ein:

```
2,5L
```

Das Ergebnis:

```
 2: mit der dynamische Zeilennummern gezeigt werden sollen.
 3: Sehen Sie, was geschieht, wenn Sie
 4: Delete und Insert
 5: (die D- und I-Befehle)
```

Um einen Bereich von Zeilen, beginnend mit der aktuellen Zeile, aufzulisten, wird ,<Zeile> L eingegeben:

```
,26L
```

Das Ergebnis:

```
15:* Die aktuelle Zeile enthält ein Sternchen.
 .
 .
 .
26: für das Aufbereiten von Text
```

Um einen Bereich von 23 Zeilen um die aktuelle Zeile herum aufzulisten, geben Sie nur L ein.

```
L
```

Das Ergebnis:

```
   4:  Delete und Insert
   5:  (die D- und I-Befehle)
   .
   .
   .
  13:  Die aktuelle Zeile liegt in der Mitte des Bereichs.
  14:  Die aktuelle Zeile wird durch L-Befehl nicht verändert.
  15:* Die aktuelle Zeile enthält ein Sternchen.
   .
   .
   .
  26:  für das Aufbereiten von Text.
```

NAME

Move

ZWECK

Übertragung eines Textbereichs an eine bestimmte Zeilennummer.

SYNTAX

[<Zeile>],[<Zeile>],<Zeile>M

KOMMENTARE

Sie können den MOVE-Befehl einsetzen, um einen Text zwischen der ersten <Zeile> und der zweiten <Zeile> an eine andere Stelle in der Datei zu übertragen. MOVE führt eine Neunumerierung der Zeilen durch.

BEISPIEL:

```
,+25,100M
```

Der Text ab der aktuellen Zeile wird in einer Länge von 25 Zeilen nach Zeile 100 übertragen. Falls eine Überlappung der Zeilenbereiche vorkommt, gibt EDLIN die Meldung "Entry error" (unzulässige Eingabe) aus.

Das folgende Beispiel zeigt Ihnen, wie Sie die Zeilen 20-30 nach Zeile 100 übertragen können:

```
20,30,100M
```

(P)AGE

NAME

Page

ZWECK

Blättert eine Datei mit Seiten von jeweils 23 Zeilen durch.

SYNTAX

[<Zeile>] [,<Zeile>]P

KOMMENTARE

Wenn Sie die erste <Zeile> weglassen, wird für diese Zeilennummer standardmäßig die aktuelle Zeile plus Eins genommen. Wenn Sie die zweite <Zeile> weglassen, werden 23 Zeilen aufgelistet. Die neue aktuelle Zeile wird zur letzten angezeigten Zeile und wird mit einem Sternchen gekennzeichnet.

(Q)UIT

NAME

Quit

ZWECK

Beendet den Editiervorgang, sichert aber keine während der Editierung vorgenommenen Änderungen und geht wieder in das MS-DOS Betriebssystem.

SYNTAX

Q

KOMMENTARE

EDLIN fragt den Benutzer, ob er die Änderungen tatsächlich nicht sichern möchte.

Wenn Sie den Editiervorgang beenden wollen, geben Sie Y ein. Die während der Editierung vorgenommenen Änderungen werden nicht gesichert. Eine .BAK-Datei wird ebenfalls nicht erstellt. Weitere Informationen über die .BAK-Datei befinden sich in diesem Kapitel (siehe End-Befehl).

Soll die Editierarbeit doch noch fortgesetzt werden, so wird N oder ein beliebiges anderes Zeichen eingegeben.

HINWEIS: Beim Start löscht EDLIN jede vorhergehende Kopie der .BAK-Datei, um Platz für die neue Kopie zu schaffen. Wird die Meldung "Abort edit (Y/N)?" (Editierung abbrechen (J/N) ?) mit Y beantwortet, so ist die vorhergehende Sicherheitskopie verloren.

BEISPIEL

```
Q
Abort edit (Y/N) ? Y RETURN
A>_
```

(R)EPLACE

NAME

Replace

ZWECK

Ersetzt jedes Auftreten einer Textfolge in einem bestimmten Bereich durch eine andere Textfolge oder eine Folge von Leerzeichen.

SYNTAX

[<Zeile>] [,<Zeile>] [?] R <Zeichenfolge1> <CONTROL-Z> <Zeichenfolge2>

KOMMENTARE

Überall, wo <Zeichenfolge1> vorkommt, wird sie durch <Zeichenfolge2> ersetzt. Jede Zeile, in der ein Austausch vorgenommen wird, wird angezeigt. Wird <Zeichenfolge1> zwei oder mehrmals in einer Zeile durch <Zeichenfolge2> ersetzt, so wird die Zeile bei jedem Austausch angezeigt. Nachdem jedes Auftreten von <Zeichenfolge1> in dem angegebenen Bereich durch <Zeichenfolge2> ersetzt wurde, wird der R-Befehl beendet und die Eingabeaufforderung in Form eines Sternchens wird erneut angezeigt.

Vorausgesetzt, daß Sie eine zweite Zeichenfolge als Ersatz angeben wollen, muß <Zeichenfolge1> von <Zeichenfolge2> durch <CONTROL-Z> getrennt sein. <Zeichenfolge2> muß ebenfalls mit der Kombination <CONTROL-Z> <CR> oder mit einem einfachen <CR> beendet werden.

Wenn Sie <Zeichenfolge1> weglassen, nimmt Replace den Wert der alten <Zeichenfolge1> an. Ist keine alte <Zeichenfolge1> vorhanden, (d.h. es handelt sich um den ersten Austausch), so wird das Austauschverfahren sofort beendet. Wenn Sie <Zeichenfolge2> weglassen, kann <Zeichenfolge1> mit <CR> beendet werden. Wenn Sie die erste <Zeile> in dem Bereichsparameter weglassen (wie in ,<Zeile>), so wird als erste <Zeile> standardmäßig die Zeile hinter der aktuellen Zeile genommen. Wenn Sie die zweite <Zeile> weglassen (wie bei <Zeile> oder <Zeile>,), so entspricht die zweite <Zeile> standardmäßig # (#gibt die Zeile hinter der letzten Zeile der Datei an).

Wenn Sie <Zeichenfolge1> mit <CONTROL-Z> beenden, und ist keine <Zeichenfolge2> vorhanden, so wird für <Zeichenfolge2> eine leere Zeichenfolge genommen, die dann zur neuen Ersatzfolge wird. So löscht beispielsweise:

R <Zeichenfolge2> <CONTROL-Z> <CR>

das Auftreten von <Zeichenfolge1>.

R <Zeichenfolge1> <CR> und
R <CR>

hingegen ersetzt <Zeichenfolge1> durch die alte <Zeichenfolge2> und die alte <Zeichenfolge1> durch die alte <Zeichenfolge2>. "Alt" bezieht sich auf eine vorhergehende Zeichenfolge, die Sie entweder in einem Search- oder einem Replace-Befehl angegeben haben.

Wird die Option des Fragezeichens (?) angegeben, so hält der Replace-Befehl bei jeder Zeile mit einer Zeichenfolge inne, die mit <Zeichenfolge1> übereinstimmt. Diese Zeile wird dann mit <Zeichenfolge2> als Ersatz angezeigt. Danach wird die Eingabeaufforderung "O.K.?." angezeigt. Betätigt der Benutzer die Taste Y oder <CR>, so wird <Zeichenfolge1> durch <Zeichenfolge 2> ersetzt. Danach wird nach dem nächsten Auftreten von <Zeichenfolge1> gesucht. Auch dann wird wieder die Eingabeaufforderung "O.K.?" angezeigt. Dieses Verfahren wird bis zum Ende des Bereichs oder bis zum Ende der Datei fortgesetzt. Nachdem das letzte Auftreten von <Zeichenfolge1> gefunden wurde, zeigt EDLIN die Eingabeaufforderung in Form eines Sternchens.

Wenn Sie eine andere Taste als die Taste Y oder <CR> im Anschluß an die Eingabeaufforderung "O.K.?" betätigen, wird die <Zeichenfolge1> unverändert in der Zeile belassen. Replace geht in diesem Fall zum nächsten Auftreten von <Zeichenfolge1> weiter. Ist <Zeichenfolge1> mehr als einmal in einer Zeile vorhanden, so wird jedes Auftreten von <Zeichenfolge1> einzeln ersetzt. Nach jedem Austausch wird die Eingabeaufforderung "O.K.?" angezeigt. Auf diese Weise wird nur die gewünschte <Zeichenfolge1> ersetzt. Ein unerwünschter Austausch findet nicht statt.

BEISPIELE

Hier wird davon ausgegangen, daß folgende Datei vorhanden ist und aufbereitet werden soll:

```
 1: Dies ist eine Beispieldatei,
 2: mit der dynamische Zeilennummern gezeigt werden sollen.
 3: Sehen Sie, was geschieht, wenn Sie
 4: Delete und Insert
 5: (die D- und I-Befehle)
 6: für das Aufbereiten von Text
 7: in Ihrer Datei benutzen.
 8: Der Insert-Befehl kann neue Zeilen
 9: in die Datei setzen; dies ist problemlos,
10: da die Zeilennummern dynamisch sind;
11: sie gehen bis 65533.
```

Um nun das Auftreten von <Zeichenfolge1> in einem bestimmten Bereich durch <Zeichenfolge2> zu ersetzen, wird:

```
2,12 Rin <CONTROL-Z> auf <CR>
```

eingegeben.

Das Ergebnis:

```
 4: Delete und Aufsert
 7: auf Ihrer Datei benutzen
 8: Der Aufsert-Befehl kann neue Zeilen
 9: auf die Datei setzen; dies ist problemlos.
```

In dem obigen Beispiel wurden nun einige Zeichenfolgen ersetzt, die nicht ersetzt werden sollten. Um dies zu vermeiden und um jeden Austausch bestätigen zu lassen, kann dieselbe Originaldatei mit einem etwas anderen Befehl benutzt werden.

Um nur ein bestimmtes Auftreten der ersten <Zeichenfolge> durch die zweite <Zeichenfolge> zu ersetzen, wird im nächsten Beispiel folgende Eingabe vorgenommen:

```
2?Rin <CONTROL-Z> auf <CR>
```

Das Ergebnis:

```
 4: Delete und Aufsert
O.K.? N
```

7: auf Ihrer Datei benutzen.
O.K.? Y
8: Der Aufsert-Befehl kann neue Zeilen
O.K.? N
*_
9: auf die Datei setzen; dies ist problemlos.
O.K.? Y
*_

Geben Sie den List-Befehl (L) ein, um das Ergebnis dieser Änderungen darzustellen:

.
.
4: Delete und Insert
7: auf Ihrer Datei benutzen.
8: Der Insert Befehl kann neue Zeilen
9: auf die Datei setzen; dies ist problemlos,
.
.

(S)EARCH

NAME

Search

ZWECK

Sucht den angegebenen Zeilenbereich nach einer bestimmten Textfolge ab.

SYNTAX

[<Zeile>] [,<Zeile>] [?] S <Zeichenfolge> <CR>

KOMMENTARE

Die <Zeichenfolge> muß mit <CR> beendet werden. Die erste Zeile, die mit <Zeichenfolge> übereinstimmt, wird angezeigt und wird zur aktueller Zeile. Wenn Sie die Option des Fragezeichens nicht angeben, wird der Search-Befehl beendet, sobald eine Übereinstimmung gefunden wird. Enthält keine Zeile eine Übereinstimmung mit <Zeichenfolge>, so erscheint die Meldung "Not found" (nicht gefunden).

Ist die Option des Fragezeichens (?) in dem Befehl enthalten, so zeigt EDLIN die erste Zeile mit einer übereinstimmenden Zeichenfolge an. Dann wird die Meldung "O.K.?" ausgegeben. Betätigen Sie die Taste Y oder <CR>, so wird die Zeile zur aktuellen Zeile und die Suche wird beendet. Wenn Sie eine andere Taste betätigen, so wird die Suche fortgesetzt, bis eine weitere Übereinstimmung gefunden ist, oder bis sämtliche Zeilen abgesucht wurden. In diesem Fall wird die Meldung "Not found" (nicht gefunden) angezeigt.

Wenn Sie die erste <Zeile> weglassen (wie bei ,<Zeile> S < Zeichenfolge>), wird standardmäßig als erste <Zeile> die Zeile im Anschluß an die aktuelle Zeile genommen. Wenn Sie die zweite <Zeile> weglassen (wie bei <Zeile> S <Zeichenfolge> oder <Zeile>, S <Zeichenfolge>), so entspricht die zweite <Zeile> standardmäßig # (Zeile hinter der letzten Zeile der Datei). Dies entspricht der Angabe <Zeile>, # S <Zeichenfolge>. Wird <Zeichenfolge> weggelassen, so nimmt der Search-Befehl die alte Zeichenfolge, sofern vorhanden. ("Alt" bezieht sich hier auf eine in einem vorhergehenden Search- oder Replace-Befehl angegebene Zeichenfolge.) Ist keine alte Zeichenfolge vorhanden (d.h. es wurde vorher keine Suche oder kein Austausch vorgenommen, so wird der Befehl sofort beendet.

BEISPIELE

Hier wird davon ausgegangen, daß folgende Datei vorhanden ist und aufbereitet werden soll:

```
 1: Dies ist eine Beispieldatei,
 2: mit der dynamische Zeilennummern gezeigt werden sollen.
 3: Sehen Sie, was geschieht, wenn Sie
 4: Delete und Insert
 5: (die D- und I-Befehle)
 6: für das Aufbereiten von Text
 7: in Ihrer Datei benutzen.
 8: Der Insert-Befehl kann neue Zeilen
 9: in die Datei setzen; dies ist problemlos,
10: da die Zeilennummern dynamisch sind;
11:* sie gehen bis 65533.
```

Um nun nach dem ersten Auftreten der Zeichenfolge "und" zu suchen, wird folgende Eingabe vorgenommen:

```
2,12 Sund <CR>
```

Die folgende Zeile wird angezeigt:

```
4: Delete und Insert
```

Um nun die Zeichenfolge "und" in Zeile 5 zu erhalten, müssen Sie den Suchbefehl durch Eingabe von:

```
<SKIP1> <COPYALL> ,12Sund <CR>
```

ändern.

Die Suche wird dann in der Zeile hinter der aktuellen Zeile (Zeile 4) fortgesetzt, da keine erste Zeile angegeben wurde. Das Ergebnis:

```
5: (die D- und I-Befehle)
```

Um nun verschiedene Zeichenfolgen abzusuchen bis die richtige Zeichenfolge gefunden wird, müssen Sie

```
1,? Sund
```

eingeben.

Das Ergebnis:

```
4: Delete und Insert
O.K.? _
```

Wenn Sie eine beliebige Taste (mit Ausnahme der Tasten Y oder <CR>) betätigen, so wird die Suche fortgesetzt. Geben Sie N ein:

```
O.K.? N
```

Daraufhin erscheint am Bildschirm:

```
5: (die D- und I-Befehle)
O.K.?_
```

Drücken Sie Y, um die Suche zu beenden:

```
O.K.? Y
*_
```

Um die Suche nach der Zeichenfolge XYZ ohne die Überprüfung (O.K.?) einzuleiten, geben Sie

```
SXYZ
```

ein.

EDLIN zeigt eine Übereinstimmung an und fährt mit der Suche nach derselben Zeichenfolge fort, sobald der Benutzer den S-Befehl ausgibt:

```
S
```

EDLIN zeigt eine weitere Übereinstimmung an.

```
S
```

EDLIN zeigt an, daß die Zeichenfolge nicht gefunden wurde.

HINWEIS: Für <Zeichenfolge> wird standardmäßig eine in einem vorhergehenden Replace- oder Search-Befehl angegebene Zeichenfolge genommen.

(T)RANSFER

NAME

Transfer

ZWECK

Fügt den Inhalt von <Dateiname> in die gerade aufbereitete Datei ein. Wenn Sie <Zeile> weglassen, wird die aktuelle Zeile benutzt.

SYNTAX

[<Zeile>] T <Dateiname>

KOMMENTARE

Dieser Befehl ist besonders nützlich, wenn Sie den Inhalt einer Datei in eine andere Datei oder in den gerade eingegebenen Text einfügen wollen. Der übertragene Text wird bei der durch <Zeile> angegebenen Zeilennummer eingefügt. Die Zeilen werden neu numeriert.

(W)RITE

NAME

Write

ZWECK

Schreibt eine angegebene Anzahl von Zeilen von den gerade im Speicher aufbereiteten Zeilen auf Platte. Beim Schreiben der Zeilen auf Platte wird mit Zeilennummer 1 begonnen.

SYNTAX

[<n>]W

KOMMENTARE

Dieser Befehl ist nur sinnvoll, wenn die gerade aufbereitete Datei für den Maschinenspeicher zu groß ist. Beim Start liest EDLIN Zeilen in den Speicher bis der Speicher zu drei Vierteln gefüllt ist.

Um den Rest der Datei aufzubereiten, müssen die im Speicher editierten Zeilen auf Platte geschrieben werden. Dann können Sie die zusätzlichen Zeilen mit Hilfe des Append-Befehls von der Platte in den Speicher laden.

HINWEIS: Wird keine Anzahl von Zeilen angegeben, so werden die Zeilen so lange in den Speicher geschrieben, bis dieser zu drei Vierteln gefüllt ist. Ist der Speicher schon zu mehr als drei Vierteln gefüllt, werden keine Zeilen geladen. Sämtliche Zeilen werden neu numeriert, so daß die erste verbleibende Zeile die Zeilennummer 1 erhält.

FEHLERMELDUNGEN

Findet EDLIN einen Fehler, so wird eine der folgenden Fehlermeldungen angezeigt:

Cannot edit .BAK file--rename file
(.BAK-Datei kann nicht editiert werden — Datei umbenennen)

Erläuterung
Sie haben versucht, eine Datei mit einer Dateinamenerweiterung .BAK zu editieren. .BAK-Dateien können nicht editiert werden, da diese Erweiterung für Sicherheitskopien reserviert ist.

Maßnahme
Wenn Sie die .BAK-Datei für die Editierung benötigen, besteht die Möglichkeit, die Datei mit einer anderen Erweiterung umzubenennen. Die .BAK-Datei kann jedoch auch kopiert werden, wobei Sie ihr eine andere Dateinamenerweiterung zuweisen müssen.

No room in directory for file
(Kein Platz für Datei im Inhaltsverzeichnis)

Erläuterung
Beim Erstellen einer neuen Datei war entweder das Dateiinhaltsverzeichnis voll oder es wurde ein ungültiges Plattenlaufwerk oder ein ungültiger Dateiname angegeben.

Maßnahme
Prüfen Sie die Befehlszeile, mit der EDLIN gestartet wurde, auf Angabe eines ungültigen Dateinamens oder eines ungültigen Plattenlaufwerks. Ist der Befehl nicht mehr am Bildschirm und haben Sie bereits einen neuen Befehl eingegeben, so kann der Startbefehl für EDLIN durch Betätigung der <COPYALL> Taste wiederhergestellt werden.

Enthält diese Befehlszeile keine unzulässigen Einträge, so sollte das CHKDSK-Programm für das angegebene Plattenlaufwerk ausführen. Ergibt sich aus der Statusmeldung, daß das Plattenverzeichnis voll ist, so muß die Platte herausgenommen werden. Legen Sie dann eine neue Platte ein und rufen Sie das Format-Programm auf.

Entry Error
(Eingabefehler)

Erläuterung
Der letzte eingegebene Befehl enthielt einen Syntaxfehler.

Maßnahme
Geben Sie den Befehl mit der richtigen Syntax erneut ein und drücken Sie <CR>.

Line too long
(Zeile zu lang)

Erläuterung
Bei einem Replace-Befehl hat eine als Ersatz angegebene Zeichenfolge die zulässige Länge von 253 Zeichen überschritten. EDLIN hat den Replace-Befehl abgebrochen.

Maßnahme
Die lange Zeile muß in zwei Zeilen aufgeteilt werden. Danach können Sie Replace-Befehl erneut ausführen.

Disk full - - file write not completed
(Platte voll — Dateischreibvorgang nicht beendet)

Erläuterung
Der Benutzer hat einen END-Befehl ausgegeben, aber die Platte verfügt nicht über genügend freien Speicherplatz für die ganze Datei. EDLIN hat den E-Befehl abgebrochen und den Benutzer wieder in das Betriebssystem zurückgeführt. Unter Umständen wurde ein Teil der Datei auf die Platte geschrieben.

Maßnahme
Nur ein Teil (wenn überhaupt) der Datei wurde gesichert. Dieser Teil der Datei sollte eigentlich gelöscht und der Editiervorgang erneut begonnen werden. Nach diesem Fehler steht die Datei nicht zur Verfügung. Vor dem Beginn des neuen Editiervorgangs sollten Sie dafür sorgen, daß die Platte über ausreichend freien Speicherplatz für die Datei verfügt.

Incorrect DOS version
(Falsche DOS-Version)

Erläuterung
Sie haben versucht, EDLIN unter einer MS-DOS-Version auszuführen, bei der es sich nicht um die Version 2.0 oder eine höhere Version handelte.

Maßnahme
Sie müssen mit der MS-DOS-Version 2.0 oder einer höheren Version arbeiten.

Invalid drive name or file
(Ungültiger Laufwerk- oder Dateiname)

Erläuterung
Beim Starten von EDLIN wurde kein gültiger Laufwerk- oder Dateiname angegeben.

Maßnahme
Geben Sie den richtigen Laufwerk- oder Dateinamen an.

Filename must be specified
(Dateiname muß angegeben werden)

Erläuterung
Beim Starten von EDLIN wurde kein Dateiname angegeben.

Maßnahme
Geben Sie einen Dateinamen an.

Invalid parameter
(Ungültiger Parameter)

Erläuterung
Beim Starten von EDLIN wurde ein anderer Schalter als der Schalter /B angegeben.

Maßnahme
Geben Sie beim Starten von EDLIN den Schalter /B an.

Insufficient memory
(Nicht ausreichender Speicherplatz)

Erläuterung
Für die Ausführung von EDLIN ist nicht ausreichend Speicherplatz vorhanden.

Maßnahme
Vor dem erneuten Start von EDLIN muß Speicherplatz freigegeben werden, indem Sie Dateien löschen oder auf Platte schreiben.

File not found
(Datei nicht gefunden)

Erläuterung
Der während eines Transfer-Befehls angegebene Dateiname wurde nicht gefunden.

Maßnahme
Bei der Ausgabe eines Transfer-Befehls müssen Sie einen gültigen Dateinamen angeben.

Must specify destination number
(Bestimmungsnummer muß angegeben werden)

Erläuterung
Für einen COPY- oder MOVE-Befehl haben Sie keine Bestimmungszeilennummer angegeben.

Maßnahme
Geben Sie den Befehl mit einer Bestimmungszeilennummer erneut aus.

Not enough room to merge the entire file
(Nicht genügend Platz, um die ganze Datei einzufügen)

Erläuterung
Während eines Transfer-Befehls war nicht genügend Speicherplatz vorhanden, um die ganze Datei aufzunehmen.

Maßnahme
Bevor diese Datei übertragen werden kann, muß Speicherplatz freigegeben werden. Sie müssen Dateien löschen oder auf Platte schreiben.

DAS DIENSTPROGRAMM FÜR DEN DATEIVERGLEICH (FC)

ALLGEMEINE INFORMATION

Es ist oft nützlich, Dateien auf einer Platte miteinander zu vergleichen. Wurde eine Datei kopiert und sollen die Kopien später miteinander verglichen werden um festzustellen, bei welcher Kopie es sich um die aktuelle Kopie handelt, können Sie das MS-Dienstprogramm für den Dateivergleich (FC) benutzen.

Das Dienstprogramm für den Dateivergleich vergleicht den Inhalt von zwei Dateien. Der Unterschied zwischen den beiden Dateien kann an die Konsole oder an eine dritte Datei ausgegeben werden. Bei den miteinander verglichenen Dateien kann es sich entweder um Quellendateien (Dateien mit Quellenanweisungen einer Programmiersprache) oder um Binärdateien handeln (Dateien, die vom Assembler, dem Link-Dienstprogramm MS-LINK oder von einem höheren Sprachcompiler erstellt werden).

Die Vergleiche werden auf eine von zwei Arten durchgeführt: entweder Zeile für Zeile oder Byte für Byte. Bei dem zeilenweisen Vergleich werden Zeilenblöcke hervorgehoben, die sich bei den beiden Dateien voneinander unterscheiden. Diese Zeilenblöcke werden dann ausgedruckt. Bei dem byteweisen Vergleich werden die Bytes angezeigt, die sich bei den beiden Dateien unterscheiden.

EINSCHRÄNKUNGEN BEI QUELLENVERGLEICHEN

FC benutzt einen großen Speicherplatz als Puffer für die Speicherung der Quellendateien. Sind die Quellendateien größer als der zur Verfügung stehende Speicherplatz, so vergleicht FC die Teile, die in den Puffer geladen werden können. Stimmen keine Zeilen in den in dem Puffer befindlichen Teilen der Dateien überein, so zeigt FC nur die folgende Meldung an:

```
FILES ARE DIFFERENT
(Dateien unterscheiden sich)
```

Bei Binärdateien, die größer als der zur Verfügung stehende Speicherplatz sind, vergleicht FC beide Dateien vollständig miteinan-

der, wobei der sich im Speicher befindliche Teil mit dem nächsten Teil von der Platte überlagert wird. Sämtliche Unterschiede werden auf dieselbe Art und Weise wie die Dateien ausgegeben, die vollständig in den Speicher passen.

DATEISPEZIFIKATIONEN

Sämtliche Dateispezifikationen benutzen die folgende Syntax:

[d:] <Dateiname> [<.erw>]

d: ist die Bezeichnung für das Plattenlaufwerk. Wenn Sie die Laufwerkbezeichnung weglassen, nimmt FC standardmäßig das Standardlaufwerk des Betriebssystems (das aktuelle Laufwerk).

Dateiname ist ein aus 1-8 Zeichen bestehender Name der Datei.

.erw ist eine aus 1-3 Zeichen bestehende Erweiterung des Dateinamens.

BENUTZUNG DES DIENSTPROGRAMMS FÜR DEN DATEIVERGLEICH

FC benutzt folgende Syntax:

FC [/# /B /W /C] <Dateiname1> <Dateiname2>

FC vergleicht die erste Datei (Dateiname1) mit der zweiten Datei (Dateiname2) und berichtet über eventuelle Unterschiede zwischen den beiden Dateien. Bei beiden Dateinamen kann es sich um Pfadnamen handeln. Zum Beispiel:

FC B: \FOO \BAR \FILE1.TXT \BAR \FILE2.TXT

FC nimmt FILE1.TXT in dem Inhaltsverzeichnis \FOO \BAR von Platte B und vergleicht sie mit FILE2.TXT in dem Inhaltsverzeichnis \BAR. Da für Dateiname2 kein Laufwerk angegeben wird, geht FC davon aus, daß sich das Inhaltsverzeichnis \BAR auf der Platte in dem Standardlaufwerk befindet.

FC-SCHALTER

Mit dem Dienstprogramm für den Dateivergleich können vier Schalter benutzt werden:

/B

Bestimmt einen Binärvergleich der beiden Dateien. Die beiden Dateien werden Byte für Byte miteinander verglichen, wobei bei einer Nichtübereinstimmung kein Versuch der Neusynchronisierung unternommen wird. Die Nichtübereinstimmungen werden folgendermaßen ausgedruckt:

```
--ADDRS----F1----F2-
xxxxxxxx      yy     zz
```

(wobei xxxxxxxx die relative Adresse des Bytepaares zum Anfang der Datei darstellt). Die Adressen beginnen bei 00000000. yy und zz sind die nichtübereinstimmenden Bytes von Datei1 bzw. Datei2. Enthält eine der Dateien weniger Daten als die andere, so wird eine Meldung ausgedruckt. Endet beispielsweise Datei1 vor Datei2, so zeigt FC:

```
***Data left in F2***
(***Daten übrig in F2***)
```

an.

/#

steht für eine Zahl von eins bis neun. Mit diesem Schalter wird die Anzahl von Zeilen angegeben, die übereinstimmen muß, damit Dateien wieder als übereinstimmend angesehen werden, nachdem ein Unterschied festgestellt wurde. Wird dieser Schalter nicht angegeben, so wird ein Standardwert von 3 genommen. Dieser Schalter wird nur bei Quellenvergleichen benutzt.

/W

Veranlaßt FC, "Leerstellen" (Tabulatorzeichen und Leerzeichen) während des Vergleichs zu komprimieren, so werden mehrere aufeinanderfolgende Leerstellen in einer Zeile als eine einzige Leerstelle betrachtet. Auch wenn FC die Leerstellen komprimiert, ignoriert es sie nicht. Die beiden einzigen Ausnahmen sind die Anfangs- und End-Leerstellen in einer Zeile. Sie werden ignoriert. Im folgenden Beispiel stellt das Unterstreichungszeichen eine Leerstelle dar):

```
___Weitere_Daten_suchen___
```

stimmt überein mit

Weitere_Daten_suchen

und mit

_____Weitere___Daten___suchen _____

stimmt jedoch nicht überein mit

____Weiteredaten____suchen.

Dieser Schalter wird nur bei Quellenvergleichen benutzt.

/C

Bei diesem Schalter ignoriert das Vergleichsverfahren die Schreibweise der Buchstaben. Sämtliche Buchstaben in den Dateien werden als Großbuchstaben angesehen. Zum Beispiel:

Viel_WEITERE___Daten___WERDEN___NICHT_GEFUNDEN

stimmt überein mit

Viel_weitere___daten___werden___nicht___gefunden

Werden die Optionen /W und /C angegeben, so komprimiert FC die Leerstellen und ignoriert die Schreibweise.

Zum Beispiel:

___DATEN___wurden___gefunden

stimmt überein mit:

daten___wurden___gefunden

Dieser Schalter wird nur in Quellenvergleichen benutzt.

ANGABE DER UNTERSCHIEDE

Das Dienstprogramm für den Dateivergleich gibt die Unterschiede zwischen den beiden angegebenen Dateien an, indem es zuerst den ersten Dateinamen und dann die Zeilen anzeigt, die sich bei den beiden Dateien unterscheiden. Darauf folgt die erste Zeile, die in beiden Dateien übereinstimmt. Danach zeigt FC den Namen der zweiten Datei an, gefolgt von den sich unterscheidenden Zeilen. Darauf folgt wiederum die erste bei den Dateien übereinstimmende Zeile. Der Standardwert für die Anzahl von Zeilen, die zwischen den Dateien übereinstimmen müssen, beträgt 3. (Wenn Sie diese Standardvorgabe ändern wollen, müssen Sie die Anzahl von Zeilen mit dem #-Schalter angeben). Zum Beispiel:

```
    . . .
    . . .
----------<Dateiname1>
<Unterschied>

<1. Zeile, die in Datei1 mit Datei2 übereinstimmt>

----------<Dateiname2>
<Unterschied>

<1. Zeile, die in Datei 2 mit Datei1 übereinstimmt>
--------------------------------------------------

    . . .
    . . .
```

FC listet weiterhin jeden Unterschied auf.

Sind zu viele Unterschiede (zu viele unterschiedliche Zeilen) vorhanden, so gibt das Programm einfach an, daß sich die Dateien unterscheiden. Der Ablauf von FC wird dann beendet.

Werden nach dem ersten angetroffenen Unterschied keine Übereinstimmungen mehr gefunden, so zeigt FC:

```
***Files are different***
(***Dateien unterscheiden sich***)
```

an. Darauf wird wieder die MS-DOS Systemmeldung in Form des Lufwerkbuchstabens (z.B. A>) angegeben.

UMLEITEN DER FC-AUSGABE AN EINE DATEI

Die Unterschiede und Übereinstimmungen zwischen den beiden angegebenen Dateien werden auf dem Bildschirm des Benutzers angezeigt, es sei denn, die Ausgabe wird an eine Datei umgeleitet. Dies wird auf auf dieselbe Art und Weise erreicht, wie bei der Umleitung von MS-DOS Befehlen (siehe Kapitel 4).

Um Datei1 und Datei2 miteinander zu vergleichen und die FC-Ausgabe an DIFFER.TXT zu senden, wird folgende Eingabe vorgenommen:

```
FC Datei1  Datei2 > DIFFER.TXT
```

Die Unterschiede und Übereinstimmungen zwischen Datei1 und Datei2 werden in die Datei DIFFER.TXT in dem Standardlaufwerk gesetzt.

BEISPIELE

Beispiel 1

Hier wird davon ausgegangen, daß diese beiden ASCII-Dateien auf Platte stehen:

ALPHA.ASM	BETA.ASM
Datei A	Datei B
A	A
B	B
C	C
D	G
E	H
F	I
G	J
H	1
I	2
M	P
N	Q
O	R
P	S
Q	T
R	U
S	V
T	4
U	5
V	W
W	X
X	Y
Y	Z
Z	

Um die beiden Dateien miteinander zu vergleichen und die Unterschiede auf dem Bildschirm anzuzeigen, wird folgende Eingabe vorgenommen:

```
FC ALPHA.ASM BETA.ASM
```

FC vergleicht ALPHA.ASM mit BETA.ASM und zeigt die Unterschiede am Bildschirm an. Alle anderen Standardvorgaben bleiben unverändert. (Die Standardvorgaben sind: keine Benutzung von Tabulatorzeichen, Leerzeichen oder Kommentaren bei Übereinstimmungen, und Ausführung eines Quellenvergleichs der beiden Dateien.)

Die Ausgabe am Bildschirm sieht folgendermaßen aus (wobei die Anmerkungen nicht angezeigt werden):

```
-------------- ALPHA.ASM
D                                   HINWEIS: Die ALPHA-
E                                   Datei enthält defg, BETA
F                                   enthält g.
G

-------------- BETA.ASM
G

----------------------------------

-------------- ALPHA.ASM
M                                   HINWEIS: Die ALPHA-
N                                   Datei enthält mno an
O                                   den Stellen, an denen
P                                   BETA j12 enthält.

-------------- BETA.ASM
J
1
2
P

----------------------------------

-------------- ALPHA.ASM
W                                   HINWEIS: Die ALPHA-
                                    Datei enthält w an den
                                    Stellen, an denen BETA
                                    45 w enthält.

-------------- BETA.ASM
4
5
W
```

Beispiel 2

Die Unterschiede können am Zeilendrucker ausgedruckt werden. In diesem Beispiel müssen vier aufeinanderfolgende Zeilen identisch sein, um eine Übereinstimmung darzustellen.

Eingabe:

```
FC /4 ALPHA.ASM  BETA.ASM < PRN
```

Die folgende Ausgabe wird am Zeilendrucker ausgegeben:

```
----------ALPHA.ASM
D
E
F
G
H
I
M
N                              HINWEIS: p ist die erste
O                              einer Folge von vier Über-
P                              einstimmungen


----------BETA.ASM
G
H
I
J
1
2
P

--------------------

----------ALPHA.ASM
W                              HINWEIS: w ist die erste
                               einer Folge von vier Über-
                               einstimmungen

----------BETA.ASM
4
5
W
```

Beispiel 3

Bei diesem Beispiel wird ein Binärvergleich vorgenommen. Danach erscheinen die Unterschiede am Bildschirm. In diesem Beispiel werden die beiden selben Quellendateien wie in den vorhergehenden Beispielen benutzt.

Eingabe:

```
FC /B ALPHA.ASM  BETA.ASM
```

Der Binärvergleich in diesem Beispiel wird durch den Schalter /B bestimmt. Diesen Schalter und alle anderen Schalter müssen Sie vor den Dateinamen in der FC-Befehlszeile eingeben. Folgende Anzeige erscheint am Bildschirm:

```
--ADDRS-----F1---F2--
 00000009      44    47
 0000000C      45    48
 0000000F      46    49
 00000012      47    4A
 00000015      48    31
 00000018      49    32
 0000001B      4D    50
 0000001E      4E    51
 00000021      4F    52
 00000024      50    53
 00000027      51    54
 0000002A      52    55
 0000002D      53    56
 00000030      54    34
 00000033      55    35
 00000036      56    57
 00000039      57    58
 0000003C      58    59
 0000003F      59    5A
 00000042      5A    1A
```

FEHLERMELDUNGEN

Wenn FC einen Fehler entdeckt, werden eine oder mehrere der folgenden Fehlermeldungen angezeigt:

Incorrect DOS version
(Falsche DOS-Version)
FC wird unter einer anderen Version als der MS-DOS-Version 2.0 oder einer höheren Verion ausgeführt.

Invalid parameter: <option>
(Ungültiger Parameter: <Option>)
Einer der angegebenen Schalter ist ungültig.

File not found: <filename>
(Datei nicht gefunden: <Dateiname>)
FC konnte den angegebenen Dateinamen nicht finden.

Read error in: <filename>
(Lesefehler in: <Dateiname>)
FC konnte nicht die ganze Datei lesen.

Invalid number of parameters
(Ungültige Anzahl von Parametern)
In der FC-Befehlszeile wurde die falsche Anzahl von Optionen angegeben.

LINKPROGRAMM (MS-LINK)

ALLGEMEINE INFORMATIONEN

In diesem Kapitel wird das Linkprogramm, MS-LINK, beschrieben. Vor der Benutzung von MS-LINK sollten Sie das vollständige Kapitel lesen.

HINWEIS: Das Lesen dieses Kapitels ist nur dann erforderlich, wenn Sie Programme kompilieren und binden wollen.

MS-LINK führt folgende Funktionen aus:

- Kombiniert einzeln erzeugte Objektmoduln in einem verschiebbaren Lademodul — einem Programm, das der Benutzer ausführen kann.
- Sucht Bibliotheksdateien nach Definitionen von nicht aufgelösten externen Referenzen ab.
- Löst externe Querverweise auf.
- Erstellt eine Auflistung, in der sowohl die Auflösung von externen Referenzen als auch Fehlermeldungen enthalten sind.

ÜBERBLICK ÜBER DAS PROGRAMM

Sie schreiben Ihr Programm im Quellencode. Dieser Quellencode durchläuft einen Compiler, der Objektmoduln erzeugt. Die Objektmoduln müssen dem Bindeverfahren unterworfen werden, um in die Maschinensprache übersetzt zu werden, die der Computer direkt verstehen kann. Diese Maschinensprache ist die für die Ausführung von Programmen erforderliche Form.

Möglicherweise möchte der Benutzer verschiedene Programme binden (kombinieren) und zusammen ausführen. Jedes der Programme kann sich auf Symbole beziehen, die in einem anderen Objektmodul definiert sind. Dieser Bezug wird als externe Referenz bezeichnet.

MS-LINK kombiniert verschiedene Objektmoduln in einem verschieblichen Lademodul oder einer Ausführungsdatei (die als

.EXE- oder ausführbare Datei bezeichnet wird). Während des Kombinierens der Moduln gewährleistet MS-LINK, daß sämtliche externen Referenzen zwischen den Objektmoduln aufgelöst werden. LINK kann verschiedene Bibliotheksdateien nach Definitionen von externen Referenzen absuchen, die in den Objektmoduln nicht definiert sind.

MS-LINK erzeugt eine Listdatei, die die aufgelösten externen Referenzen enthält, und zeigt auch eventuelle Fehlermeldungen an.

MS-LINK nutzt den verfügbaren Speicherplatz so gut wie möglich. Ist der verfügbare Speicherplatz erschöpft, so erstellt MS-LINK eine temporäre Plattendatei namens VM.TMP.

In Abbildung 9.1 werden die verschiedenen Teile der MS-LINK Operation dargestellt.

BENÖTIGTE DEFINITIONEN

Einige der in diesem Kapitel benutzten Ausdrücke werden nachstehend erläutert. Diese Erläuterungen werden Ihnen zu einem besseren Verständnis von MS-LINK verhelfen. Werden aus der BASIC-, Pascal- oder einer höheren Programmiersprache kompilierte Objektmoduln gebunden, so brauchen diese Ausdrücke generell nicht bekannt zu sein. Werden jedoch Programme in der Assemblersprache geschrieben und kompiliert, so müssen Sie den Betrieb von MS-LINK und die in diesem Abschnitt beschriebenen Definitionen näher kennenlernen.

Bei MS-DOS kann der Speicher in Segmente, Klassen und Gruppen unterteilt werden. In Abbildung 9.2 wird dieses Konzept dargelegt.

Hier soll davon ausgegangen werden, daß die drei Segmente über die folgenden Namen verfügen.

	Segmentname	Segmentklassen-name
Segment 1	PROG.1	CODE
Segment 2	PROG.2	CODE
Segment 3	PROG.3	DATA

Die Segmente 1, 2 und 12 haben zwar verschiedene Segmentnamen, können jedoch denselben Segmentklassennamen aufweisen

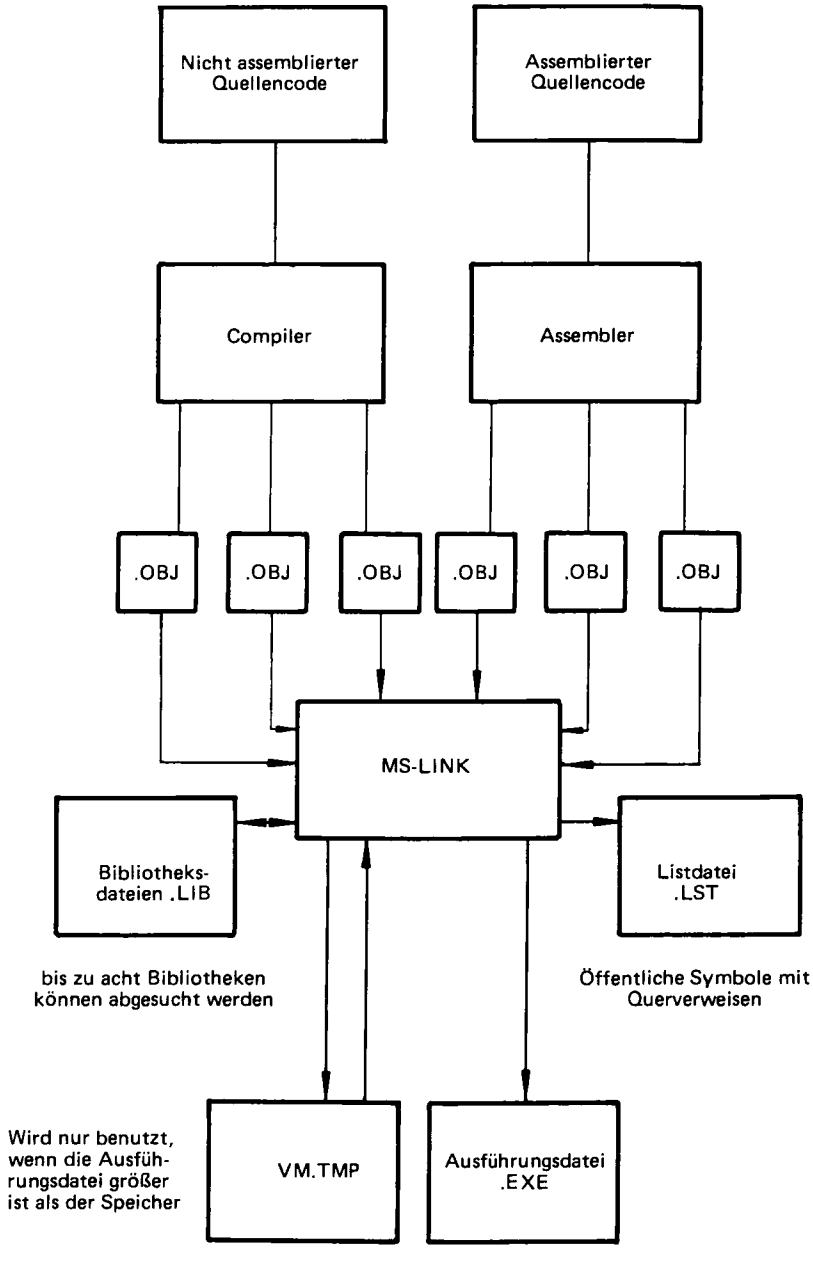
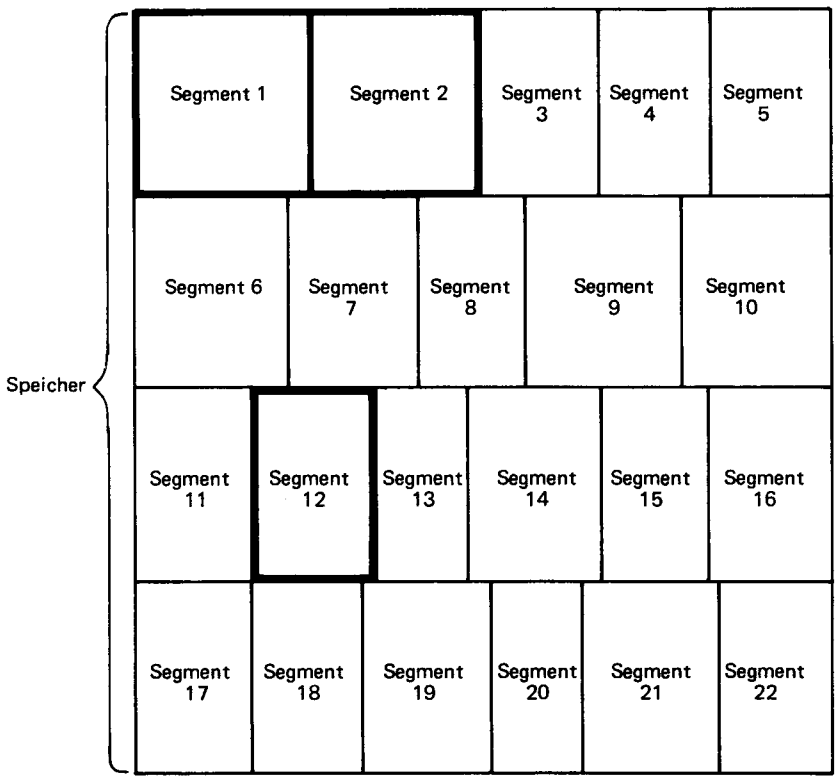


Abb. 9.1 MS-LINK Operation

oder auch nicht. Die Segmente 1, 2 und 12 bilden eine Gruppe mit einer Gruppenadresse, die der niedrigsten Adresse von Segment 1 entspricht (d.h. der niedrigsten Adresse im Speicher).

Jedes Segment verfügt über einen Segmentnamen und einen Klassennamen. MS-LINK lädt sämtliche Segmente nach Klassennamen vom ersten angetroffenen Segment bis zum letzten in den Speicher. Sämtliche derselben Klasse zugeordneten Segmente werden fortlaufend in den Speicher geladen.



Fett umrandeter Bereich = eine Gruppe (64 KB adressierbar)

Abb. 9.2 Aufteilung eines Speichers

Während der Verarbeitung bezieht sich MS-LINK auf die Segmente über ihre Adressen im Speicher. Zu diesem Zweck sucht MS-LINK nach Segmentgruppen.

Eine Gruppe ist eine Sammlung von Segmenten, die in einen 64 KB-Speicherbereich paßt. Die Segmente müssen nicht fortlaufend sein, um eine Gruppe zu bilden (siehe Abb. 9.2). Die Adresse einer Gruppe ist die niedrigste Adresse eines Segments in dieser Gruppe. Während des Bindens analysiert MS-LINK die Gruppen und nimmt dann durch die Speicheradresse dieser Gruppe Bezug auf die Segmente. Ein Programm kann aus einer oder mehreren Gruppen bestehen.

Wenn Sie in der Assemblersprache schreiben, können Sie die Gruppen- und Klassennamen in dem Programm zuweisen. In höheren Programmiersprachen (BASIC, COBOL, FORTRAN, Pascal) wird die Benennung automatisch vom Compiler durchgeführt.

VON MS-LINK BENUTZTE DATEIEN

MS-LINK arbeitet mit einer oder mehreren Eingabedateien, erzeugt zwei Ausgabedateien, kann eine temporäre Plattendatei erstellen und kann angewiesen werden, bis zu acht Bibliotheksdateien abzusuchen.

Für jeden Dateityp kann eine dreiteilige Dateispezifikation angegeben werden. Das Format für die Dateispezifikationen bei MS-LINK ist identisch mit dem einer Plattendatei:

[d:]<Dateinamen>[<.erw>]

- d: ist die Laufwerkbezeichnung. Zulässige Laufwerkbezeichnungen für MS-LINK sind A: bis 0:. Der Doppelpunkt ist immer als Teil der Laufwerkbezeichnung erforderlich.
- Dateiname ist ein beliebiger zulässiger Dateiname aus 1-8 Zeichen.
- .erw ist eine aus einem bis drei Zeichen bestehende Dateinamenerweiterung. Der Punkt ist immer als Teil der Erweiterung erforderlich.

Eingabedateierweiterungen

Werden in der Dateispezifikation der Eingabedatei (Objektdatei) keine Dateinamenerweiterungen angegeben, so erkennt MS-LINK standardmäßig die folgenden Erweiterungen:

.OBJ Objektdatei
.LIB Bibliotheksdatei

Ausgabedateierweiterungen

MS-LINK hängt die folgenden Standarderweiterungen an die Ausgabedateien (Ausführungs- und Listdateien) an:

.EXE Ausführungsdatei (darf nicht überschrieben werden)
.MAP Listdatei (darf überschrieben werden)

VM.TMP Datei (Temporäre Datei)

MS-LINK benutzt den verfügbaren Speicherplatz für den LINK-Vorgang. Ergeben die zu bindenden Dateien eine Ausgabedatei, die den zur Verfügung stehenden Speicherplatz überschreitet, so erstellt MS-LINK eine temporäre Datei namens VM.TMP. Es speichert sie auf die Diskette in dem Standardlaufwerk. MS-LINK gibt dann folgende Meldung aus:

VM.TMP has been created.
Do not change disk in drive, <d:>

(VM.TMP wurde erstellt.
Diskette in Laufwerk <d:> nicht auswechseln.)

Nach dem Erscheinen dieser Meldung darf die Diskette erst nach Beendigung des LINK-Vorgangs aus dem Standardlaufwerk herausgenommen werden. Wenn Sie die Diskette herausnehmen, kann auf das Ergebnis des LINK-Vorgangs kein Verlaß mehr sein. MS-LINK zeigt unter Umständen folgende Fehlermeldung an:

Unexpected end of file on VM.TMP
(Unerwartetes Dateiende bei VM.TMP)

Der Inhalt von VM.TMP wird im Anschluß an die Systemmeldung "Run file:" (Ausführungsdatei:) in die Datei geschrieben. VM.TMP ist nur eine Arbeitsdatei. Sie wird am Ende des LINK-Vorgangs gelöscht.

ACHTUNG

VM.TMP darf nicht als Dateiname für irgendeine Datei benutzt werden. Ist schon eine Datei namens VM.TMP in dem Standardlaufwerk vorhanden und fordert MS-LINK die VM.TMP Datei an, so löscht MS-LINK die bereits auf der Diskette befindliche VM.TMP Datei und erstellt eine neue VM.TMP. Auf diese Weise ist der Inhalt der vorhergehenden VM.TMP Datei verloren.

BENUTZUNG VON MS-LINK

STARTEN VON MS-LINK

MS-LINK erfordert zwei Arten von Eingaben: einen Befehl für das Starten von MS-LINK und Antworten auf die Eingabeaufforderungen. Darüber hinaus steuern sechs Schalter die MS-LINK Einrichtungen. Normalerweise werden sämtliche Befehle für MS-LINK von der Terminaltastatur eingegeben. Wahlweise können die Antworten auf die Eingabeaufforderungen und eventuelle Schalter in einer Antwortdatei enthalten sein. Befehlszeichen unterstützen Sie bei der Ausgabe von Befehlen an MS-LINK.

Es gibt drei Möglichkeiten, MS-LINK zu starten. Bei der ersten Methode werden die Befehle als Antwort auf einzelne Eingabeaufforderungen eingegeben. Bei der zweiten Methode werden sämtliche Befehle in der für das Starten von MS-LINK benutzten Zeile eingegeben. Um MS-LINK mit der dritten Methode zu starten, muß eine Antwortdatei erstellt werden. Diese Antwortdatei enthält alle erforderlichen Befehle und teilt MS-LINK beim Starten mit, wo sich diese Datei befindet.

Methode 1	LINK
Methode 2	LINK <Dateinamen> [/Schalter]
Methode 3	LINK @<Dateispez>

Zusammenfassung der Methoden für das Starten von MS-Link

Methode 1: Eingabeaufforderungen

Um MS-LINK mit Methode 1 zu starten, geben Sie

```
LINK
```

ein. MS-LINK wird in den Speicher geladen. Danach zeigt MS-LINK nacheinander vier Eingabeaufforderungen an. Der Benutzer muß die Eingabeaufforderungen beantworten, damit MS-LINK bestimmte Aufgaben ausführen kann.

Am Ende jeder Zeile kann der Benutzer einen oder mehrere Schalter eingeben, vor denen das Schalterzeichen steht (in diesem Fall ein Schrägstrich /).

Die Eingabeaufforderungen werden in der nachfolgenden Tabelle zusammengefaßt. In dem Abschnitt "Eingabeaufforderungen" werden sie im einzelnen beschrieben.

EINGABEAUFFORDERUNG	ANTWORTEN
Object Modules [.OBJ]: (Objektmoduln [.OBJ]:)	Listen Sie die zu bindenden .OBJ-Dateien auf. Sie müssen durch Leerzeichen oder Pluszeichen (+) voneinander getrennt werden. Ist ein Pluszeichen das letzte eingegebene Zeichen, so wird die Eingabeaufforderung erneut angezeigt. Da keine Standardvorgabe vorhanden ist, müssen Sie eine Angabe machen.
Run File [Object-file .EXE]: (Ausführungsdatei [Objektdatei.EXE])	Den Dateinamen für den ausführbaren Objektcode angeben. Die Standardvorgabe ist der erste Objektdateiname .EXE. (Die Erweiterung des Ausgabedateinamens kann nicht geändert werden.)
List File [Run-file .MAP]: (Listdatei [Ausführungsdatei .MAP]:)	Den Dateinamen für die Auflistung angeben. Die Standardvorgabe ist der RUN-Dateiname.
Libraries [ ]: (Bibliotheken [ ]:)	Listen die abzusuchenden Dateinamen auf, wobei die Dateinamen durch Leerzeichen oder Pluszeichen (+) voneinander getrennt sein müssen. Ist ein Pluszeichen das letzte eingegebene Zeichen, so wird die Eingabeaufforderung erneut angezeigt. Die Standardvorgabe ist "keine Suche". (Die Dateinamenerweiterungen werden in .LIB geändert.)

Methode 2: Befehlszeile

Um MS-LINK mit Methode 2 zu starten, müssen Sie sämtliche Befehle in einer Zeile eingeben. Die auf LINK folgenden Einträge sind Antworten auf die Eingabeaufforderungen. Die Eingabefelder für die verschiedenen Eingabeaufforderungen müssen durch Kommas voneinander getrennt sein. Folgende Syntax ist zu benutzen:

LINK <Objekt-Liste>,<Ausführungsdatei>,<Listdatei>,
<Bibliotheks-Liste>[/Schalter...]

- Objekt-Liste ist eine Liste von Objektmoduln, die durch Pluszeichen voneinander getrennt sind.
- Ausführungsdatei ist der Name der Datei, die die ausführbare Ausgabe empfangen soll.
- Listdatei ist der Name der Datei, die die Auflistung aufnehmen soll.
- Bibliotheks-Liste ist eine Liste von Bibliotheksmoduln, die abgesucht werden müssen.
- /Schalter bezieht sich auf wahlweise Schalter, die im Anschluß an jede der Antworten angegeben werden können (unmittelbar vor den Kommas oder nach <Bibliotheks-Liste>,).

Um die Standardvorgabe für ein Feld auszuwählen, geben Sie einfach ein zweites Komma ohne Leerzeichen zwischen den beiden Kommas ein.

```
LINK
FUN+TEXT+TABLE+CARE/P/M,,FUNLIST,COBLIB.LIB
```

Durch diesen Befehl wird MS-LINK geladen. Danach werden die Objektmoduln FUN.OBJ,TEXT.OBJ,TABLE.OBJ und CARE.OBJ geladen. Dann hält MS-LINK (aufgrund des /P-Schalters) inne. MS-LINK bindet die Objektmoduln, sobald Sie eine beliebige Taste betätigen. Danach erzeugt es ein globales Symbolabbild (/M-Schalter), nimmt standardmäßig die Ausführungsdatei FUN.EXE, erstellt eine Listdatei namens FUNLIST.MAP und sucht die Bibliotheksdatei COBLIB.LIB ab.

Methode 3: Antwortdatei

Um MS-LINK mit Methode 3 zu starten, müssen Sie

```
LINK @<Dateispez>
```

eingeben.

Filespec ist der Name der Antwortdatei. Diese enthält Antworten auf die MS-LINK-Eingabeaufforderungen, die unter Methode 1 erläutert wurden. Sie darf auch von Schaltern Gebrauch machen. Die Auswahl einer Dateinamenerweiterung für die Antwortdatei steht Ihnen frei. Die Methode 3 ermöglicht die Eingabe des Befehls, der MS-LINK einleitet, wahlweise von der Tastatur oder von einer Batch-Datei. Weitere Eingriffe Ihrerseits sind dann nicht mehr erforderlich.

Um diese Möglichkeit zu verwirklichen, müssen Sie eine Antwortdatei erstellen. Jede Textzeile dieser Datei muß die Antwort auf eine MS-LINK-Eingabeaufforderung darstellen. Die Antworten müssen der Reihenfolge der Eingabeaufforderungen entsprechen, die unter Methode 1 beschrieben wurde. Die Antworten auf die Aufforderungen "Object Modules:" und "Libraries:" dürfen über die Länge einer Bildschirmzeile hinausgehen. Hierzu müssen Sie lediglich ein Pluszeichen (+) am jeweiligen Zeilenende eingeben.

Schalter sowie Befehle sind genau so aufzubereiten, wie Sie sie von der Tastatur eingeben würden.

Beim Ablauf des MS-LINK-Vorgangs erscheinen die Eingabeaufforderung und die der Antwortdatei entnommenen Antworten. Im Falle, daß Angaben (z.B. Dateinamen, das Eingabe-Steuerzeichen Semikolon, CR) fehlen, gibt MS-LINK die Eingabeaufforderung aus, der nicht entsprochen wurde, und wartet auf eine gültige Eingabe. Erst nach dieser Eingabe wird der Linkvorgang fortgesetzt.

Beispiel einer Antwortdatei:

```
FUN TEXT TABLE CARE
/PAUSE/MAP
FUNLIST
COBLIB.LIB
```

Hierdurch wird MS-LINK angewiesen, die vier Objektmoduln FUN, TEXT, TABLE und CARE zu laden. Bevor eine Aufstellung der externen Symbole erzeugt wird, hält MS-LINK inne, damit Sie die Diskette wechseln können. (Hierzu können Sie die Erläuterungen zu /PAUSE in dem Abschnitt "Switches" lesen). Sobald Sie eine Taste drücken, werden den Ausgabedateien die Namen FUN.EXE und FUNLIST.MAP zugeteilt. MS-LINK sucht dann die Bibliotheksdatei COBLIB.LIB ab, wobei die Standardvorgaben für die Schalter in Kraft treten.

EINGABE-STEUERZEICHEN

MS-LINK verwendet drei Eingabe-Steuerzeichen

Pluszeichen (+)

Das Pluszeichen dient der Trennung von Eingaben sowie der Verlängerung der Befehlszeile über die Länge einer Bildschirmzeile hinaus. Letztere Verwendung kann nur bei Antworten auf "Object Modules:" oder "Libraries:" vorkommen. Unmittelbar auf das Pluszeichen folgt dann <CR>. Solange Sie +<CR> eingeben, wird MS-LINK Modulnamen fordern. Wenn Ihre Antworten auf eine Eingabeaufforderung vollständig sind, drücken Sie einfach <CR>. Zur Trennung von Objektmoduln dürfen Sie anstelle des Pluszeichens ein Leerzeichen verwenden.

Beispiel:

```
Object Modules [.OBJ]: FUN TEXT TABLE CARE+<CR>
Object Modules [.OBJ]: FOO+FLIPFLOP+JUNQUE+<CR>
Object Modules [.OBJ]: CORSAIR<CR>
```

Semikolon (;)

Um die Standardvorgaben für die verbleibenden Eingabeaufforderungen zu bewirken, können Sie ein Semikolon und unmittelbar darauf <CR> an jeder beliebigen Stelle nach der ersten Eingabeaufforderung (Run File:) eingeben. Damit erübrigt sich das mehrmalige Betätigen von <CR>.

HINWEIS: Nach der ;<CR> Folge haben Sie keine Möglichkeit, auf weitere Eingabeaufforderung zu antworten. Deshalb müssen Sie <CR> ohne Semikolon benutzen, wenn Sie nur einzelne Eingabeaufforderungen überspringen wollen.

Beispiel:

```
Objekt Modules [.OBJ]: FUN TEXT TABLE CARE<CR>
Run Module [FUN.EXE]:;<CR>
```

Keine weiteren Eingabeaufforderungen werden angezeigt. MS-LINK benutzt die Standardvorgaben (einschließlich FUN. MAP für die Listdatei).

<CONTROL-C>

Mit der <CONTROL-C> Taste können Sie den LINK-Vorgang jederzeit abbrechen. Sollten Sie eine fehlerhafte Antwort, wie beispielsweise einen falschen Dateinamen oder einen nicht richtig buchstabierten Dateinamen, eingeben, so müssen Sie <CONTROL-C> betätigen, um MS-LINK zu beenden. Danach rufen Sie MS-LINK erneut auf. Wenn Sie nach der fehlerhaften Eingabe die <CR> Taste noch nicht betätigt haben, können die fehlerhaften Zeichen mit der Rücktaste gelöscht werden.

EINGABEAUFFORDERUNGEN

MS-LINK fordert den Benutzer zur Beantwortung von vier Eingabeaufforderungen in Form von Texten auf. Gibt der Benutzer eine Antwort auf eine Eingabeaufforderung ein und betätigt er <CR>, so wird die nächste Eingabeaufforderung angezeigt. Nachdem die letzte Eingabeaufforderung beantwortet wurde, beginnt MS-LINK automatisch ohne weiteren Befehl mit dem Binden. Nachdem der LINK-Vorgang beendet ist, kehrt MS-LINK zu der Betriebssystemebene zurück. Die Anzeige der Systemmeldung des Betriebssystems bedeutet, daß MS-LINK erfolgreich beendet wurde. Konnte MS-LINK nicht erfolgreich beendet werden, so zeigt MS-LINK eine entsprechende Fehlermeldung an.

MS-LINK fordert den Benutzer zur Eingabe der Namen der Objekt-, Ausführungs- und Listdateien und der Bibliotheken auf. Die Eingabeaufforderungen werden in der Reihenfolge aufgelistet, in der sie am Bildschirm angezeigt werden. Die Standardantwort erscheint in eckigen Klammern [ ] im Anschluß an die Eingabeaufforderung, sofern für die Eingabeaufforderung eine Standardantwort vorhanden ist. Bei der Eingabeaufforderung "Object Modules:" ist jedoch kein vorher festgelegter Dateiname vorhanden, so daß der Benutzer einen Dateinamen eingeben muß.

Object Modules [.OBJ]:
(Objektmoduln) [.OBJ]:)

Der Benutzer muß eine Liste der zu bindenden Objektmoduln eingeben. MS-LINK geht standardmäßig davon aus, daß die Dateinamenerweiterung .OBJ ist. Verfügt ein Objektmodul über eine andere Dateinamenerweiterung, so muß diese Erweiterung angegeben werden. Ansonsten kann die Erweiterung weggelassen werden.

Die Moduln müssen durch Pluszeichen (+) voneinander getrennt sein.

Hier wird nochmals darauf hingewiesen, daß MS-LINK Segmente in der angetroffenen Reihenfolge in Klassen lädt. Auf diese Weise kann der Benutzer die Reihenfolge festlegen, in der die Objektmoduln von MS-LINK gelesen werden.

Run File [First-Object-filename.EXE]:
(Ausführungsdatei [Erster Objekt-Dateiname.EXE]:)

Durch Eingabe eines Dateinamens wird eine Datei für die Speicherung der Ausführungsdatei (ausführbaren Datei) erstellt, die sich aus dem LINK-Vorgang ergibt. Sämtliche Ausführungsdateien erhalten die Dateinamenerweiterung .EXE, selbst wenn Sie eine andere Erweiterung als .EXE angeben.

Wird auf die Eingabeaufforderung "Run File:" keine Antwort eingegeben, so benutzt MS-LINK den ersten als Antwort auf die Eingabeaufforderung "Object Modules:" eingegebenen Dateinamen als Dateinamen für die Ausführungsdatei.

Beispiel:

Run File [FUN.EXE]: B:PAYROLL/P

Mit dieser Antwort wird MS-LINK angewiesen, die Ausführungsdatei PAYROLL.EXE in Laufwerk B: zu erstellen. Außerdem hält MS-LINK inne, so daß der Benutzer eine neue Diskette für die Aufnahme der Ausführungsdatei einlegen kann.

List File [Run-Filename.MAP]:
(Listdatei [Ausführungs-Dateiname.MAP]:)

Die Listdatei enthält einen Eintrag für jedes Segment in den Eingabemoduln (Objektmoduln). Jeder Eintrag zeigt die Adressierung in der Ausführungsdatei.

Standardmäßig wird der Dateiname der Ausführungsdatei mit der Standard-Dateinamenerweiterung .MAP benutzt.

Libraries [ ]:
(Bibliotheken [ ]:)

Gültige Antworten bestehen aus bis zu acht Bibliotheksdateinamen oder einfach einer Zeilenschaltung. (Eine Zeilenschaltung bedeutet, daß keine Bibliothekssuche stattfinden soll.) Bibliotheksdateien müssen von einem Bibliotheksdienstprogramm erstellt worden sin. Standardmäßig nimmt MS-LINK die Dateinamenerweiterung .LIB für Bibliotheksdateien.

Bibliotheksdateinamen müssen durch Leerzeichen oder Pluszeichen (+) getrennt werden.

MS-LINK sucht die Bibliotheksdateien in der aufgeführten Reihenfolge ab, um die externen Referenzen aufzulösen. Wenn es das Modul findet, das das externe Symbol definiert, verarbeitet MS-LINK dieses Modul als weiteres Objektmodul.

Kann MS-LINK keine Bibliotheksdatei auf den Disketten in den Diskettenlaufwerken finden, so zeigt es folgende Meldung an:

Cannot find library<library-name>
Type new drive letter:

(Kann Bibliothek<Bibliotheksname> nicht finden
Neuen Laufwerkbuchstaben eingeben:)

In diesem Fall muß der Benutzer den Buchstaben für eine Laufwerkbezeichnung (beispielsweise B) eingeben.

SCHALTER VON MS-LINK

Die sechs Schalter von MS-LINK steuern verschiedene Funktionen von MS-LINK. Die Schalter müssen am Ende einer Antwort auf eine Eingabeaufforderung eingegeben werden, unabhängig von der für das Starten von MS-LINK benutzten Methode. Die Schalter können am Ende einer beliebigen Antwort gruppiert sein. Sie können jedoch auch am Ende von mehreren Antworten angegeben werden. Wird am Ende einer Antwort mehr als ein Schalter eingegeben, so muß vor jedem Schalter ein Schrägstrich (/) stehen.

Sämtliche Schalter können abgekürzt werden. Die einzige Einschränkung besteht darin, daß eine Abkürzung folgerichtig vom ersten bis zum letzten eingegebenen Buchstaben sein muß. Zwischenräume oder Umstellungen sind nicht zulässig. In diesem Zusammenhang wird auf die nachfolgende Aufstellung von gültigen und ungültigen Abkürzungen hingewiesen.

Zulässig	Unzulässig
/D	/DSL
/DS	/DAL
/DSA	/DLC
/DSALLOCA	/DSALLOCT

/DSALLOCATE

Mit dem /DSALLOCATE-Schalter wird MS-LINK angewiesen, sämtliche Daten in den oberen Bereich des Datensegments zu laden. Ansonsten lädt MS-LINK sämtliche Daten in den unteren Bereich des Datensegments. Bei der Ausführung wird der DS-Zeiger auf die niedrigstmögliche Adresse gesetzt, damit das gesamte DS-Segment benutzt werden kann. Wird der /DSALLOCATE-Schalter zusammen mit dem standardmäßigen Laden in den unteren Bereich (d.h. der Schalter /HIGH wird nicht benutzt) verwendet, so kann ein Anwenderprogramm dynamisch über den Speicherplatz unter dem Bereich verfügen, der ausdrücklich innerhalb von DGROUP zugeordnet ist. Dennoch kann dieser Bereich von demselben DS-Zeiger adressiert werden. Diese dynamische Zuordnung wird bei Pascal- und FORTRAN-Programmen benötigt.

HINWEIS: Das Anwenderprogramm kann dynamisch bis zu 64 KB (oder den tatsächlich zur Verfügung stehenden Speicherplatz) zuordnen, abzüglich dem innerhalb von DGROUP zugeordneten Speicherplatz.

/HIGH

Durch Benutzung des Schalters /HIGH speichert MS-LINK die Ausführungsdatei möglichst weit oben im Speicherbereich. Ansonsten legt MS-LINK die Ausführungsdatei möglichst weit unten im Bereich an.

ACHTUNG

Der Schalter /HIGH darf nicht mit Pascal- oder FORTRAN-Programmen benutzt werden.

/LINENUMBERS

Mit dem Schalter /LINENUMBERS wird MS-LINK angewiesen, die Zeilennummern und Adressen der Quellenanweisungen in den Eingabemoduln, in die Listdatei aufzunehmen. Ansonsten erscheinen keine Zeilennummern in der Listdatei.

HINWEIS: Nicht alle Compiler erzeugen Objektmoduln mit Informationen über die Zeilennummern. In diesen Fällen kann MS-LINK natürlich keine Zeilennummern aufnehmen.

/MAP

/MAP weist MS-LINK an, sämtliche in den Eingabemoduln definierten globalen Symbole aufzulisten. Wenn Sie /MAP nicht angeben, listet MS-LINK nur Fehler auf (einschließlich nicht definierter globaler Symbole).

Die Symbole werden alphabetisch aufgelistet. Für jedes Symbol listet MS-LINK seinen Wert und die Adresse der Ausführungsdatei als Segment:Offset auf. Die Symbole werden am Ende der Listdatei aufgelistet.

/PAUSE

Durch den Schalter /PAUSE unterbricht MS-LINK die LINK-Sitzung vorübergehend, wenn der Schalter gesetzt wird. Normalerweise führt MS-LINK die LINK-Sitzung von Anfang bis Ende ohne Stop durch. Mit diesem Schalter kann der Benutzer die Disketten herausnehmen, bevor MS-LINK die Ausführungsdatei (.EXE) ausgibt.

Stößt MS-LINK auf den Schalter /PAUSE, so zeigt es folgende Meldung an:

About to generate .EXE file
Change disk<hit any key>

(Im Begriff, eine .EXE-Datei zu erzeugen
Disketten austauschen <beliebige Taste betätigen>)

MS-LINK nimmt die Verarbeitung wieder auf, sobald Sie eine beliebige Taste betätigen.

ACHTUNG

Die Diskette für die Aufnahme der Listdatei oder die Diskette für die VM.TMP Datei, sofern diese erstellt wurde, dürfen Sie nicht entfernen.

/STACK: <number>

Der Eintrag für die Zahl stellt einen beliebigen positiven numerischen Wert bis zu 65.536 Bytes (in Hexadezimal-Zahlendarstellung) dar. Wird ein Wert von 1 bis 511 eingegeben, so benutzt MS-LINK 512. Wird der Schalter/STACK für den LINK-Vorgang nicht benutzt, so berechnet MS—LINK die erforderliche Stapelgröße automatisch.

Sämtliche Compiler und Assembler müssen in den Objektmoduln Informationen enthalten, anhand derer das Link-Programm die erforderliche Stapelgröße berechnen kann.

Mindestens ein Objektmodul (Eingabemodul) muß eine Anweisung für die Stapelzuweisung enthalten. Ist dies nicht der Fall, so zeigt MS-LINK folgende Fehlermeldung an:

WARNING: NO STACK STATEMENT
(WARNUNG: KEINE VORBELEGUNG DES STACK-SEGMENTS)

BEISPIEL FÜR DIE ANWENDUNG VON MS-LINK

In diesem Beispiel wird dargestellt, welche Informationen bei der Anwendung von MS-LINK angezeigt werden. Als Antwort auf die MS-DOS Systemmeldung geben Sie

LINK

ein. Das System zeigt folgende Meldungen und Eingabeaufforderungen an (Ihre Antworten sind unterstrichen):

```
Microsoft Object Linker V.2.00
(C) Copyright 1982 by Microsoft Inc.

Object Modules [.OBJ]: NCRIO SYSINIT
Run File [NCRIO.EXE]:
List File [NUL.MAP]:NCRIO /MAP
Libraries [.LIB]:;
```

Sie werden feststellen, daß einige Antworten MS-LINK steuern, während andere die Ausgabe beeinflussen:

- Durch Angabe von /MAP erhält der Benutzer sowohl eine alphabetische als auch eine chronologische Auflistung der öffentlichen Symbole (d.h. Symbole, die auch außerhalb des Moduls, in dem sie definiert sind, von Bedeutung sind).
- Wird die Eingabeaufforderung "List File:" mit PRN beantwortet, so kann die Ausgabe an den Drucker geleitet werden.
- Wenn Sie den Schalter /LINE angeben, erhalten Sie eine Auflistung sämtlicher Zeilennummern für alle Moduln. (Die Ausgabe bei Angabe des Schalters /LINE kann sehr umfangreich werden.)
- <CR> als Antwort auf die Eingabeaufforderung "Libraries:" bewirkt eine automatische Bibliothekssuche.

Nachdem MS-LINK sämtliche Bibliotheken gefunden hat, zeigt das Linkprogramm eine Liste der Segmente in der Reihenfolge ihres Auftretens innerhalb des Lademoduls an. Dies könnte wie folgt aussehen:

```
Anfang    Ende     Länge    Name

00000H    009ECH   09EDH    CODE
009F0H    01166H   0777H    SYSINITSEG
```

Die Informationen in den Anfang- und Ende-Spalten zeigen die aus 20 Bits bestehende Hexadezimaladresse jedes Segments relativ zu Adresse 0 an. Adresse 0 ist der Anfang des Lademoduls.

Bei den angezeigten Adressen handelt es sich nicht um die absoluten Adressen, an denen diese Segmente sich befinden. In dem *PROGRAMMIERER-HANDBUCH ('PROGRAMMER'S MANUAL')*

wird beschrieben, wie Sie die tatsächliche Adresse von relativ Null ermitteln können. Außerdem wird beschrieben, wie sich die absolute Adresse eines Segments ermitteln läßt.

Da der Schalter /MAP benutzt wurde, zeigt MS-LINK die öffentlichen Symbole mit Namen und Wert an. Zum Beispiel:

```
ADDRESS         PUBLICS_BY_NAME
009F:0012       BUFFERS
009F:0005       CURRENT_DOS_LOCATION
009F:0011       DEFAULT_DRIVE
009F:000B       DEVICE_LIST
009F:0013       FILES
009F:0009       FINAL_DOS_LOCATION
009F:000F       MEMORY_SIZE
990F:0000       SYSINIT

ADDRESS         PUBLICS BY VALUE
009F:0000       SYSINIT
009F:0005       CURRENT_DOS_LOCATION
009F:0009       FINAL_DOS_LOCATION
009F:000B       DEVICE_LIST
009F:000F       MEMORY_SIZE
009F:0011       DEFAULT_DRIVE
009F:0012       BUFFERS
009F:0013       FILES
```

FEHLERMELDUNGEN

Sämtliche Fehler führen zu einem Abbruch der LINK-Sitzung. Nachdem Sie die Ursache des Fehlers gefunden und den Fehler behoben haben, können Sie MS-LINK erneut starten. Die folgenden Fehlermeldungen werden von MS-LINK angezeigt.

ATTEMPT TO ACCESS DATA OUTSIDE OF SEGMENT BOUNDS, POSSIBLY BAD OBJECT MODULE
(Versuch, auf Daten außerhalb der Segmentgrenzen zuzugreifen, wahrscheinlich fehlerhaftes Objektmodul)

Wahrscheinlich ist eine fehlerhafte Objektdatei vorhanden.

BAD NUMERIC PARAMETER
(Falscher numerischer Parameter)

Der numerische Wert ist nicht in Ziffern angegeben.

CANNOT OPEN TEMPORARY FILE
(Temporäre Datei kann nicht eröffnet werden)

MS-LINK kann die Datei VM.TMP nicht erstellen, da das Inhaltsverzeichnis der Diskette voll ist. Neue Diskette einlegen. Die Diskette für die Aufnahme der LIST.MAP-Datei darf nicht entfernt werden.

ERROR: DUP RECORD TOO COMPLEX
(Fehler: DUP-Satz zu komplex)

Der DUP-Satz in einem Modul der Assemblersprache ist zu komplex. Den DUP-Satz in dem Assemblerprogramm vereinfachen.

ERROR: FIXUP OFFSET EXCEEDS FIELD WIDTH
(Fehler: Relativzeiger für die Fixierung überschreitet Feldbreite)

Eine Instruktion in der Assemblersprache bezieht sich auf eine Adresse mit einer Instruktion in kurzer Definitionsform (z.B. IMP SHORT) anstelle einer Instruktion in langer Definitionsform. Das Assembler-Quellenprogramm aufbereiten und neu assemblieren.

INPUT FILE READ ERROR
(Lesefehler bei der Eingabedatei)

Wahrscheinlich ist eine fehlerhafte Objektdatei vorhanden.

INVALID OBJECT MODULE
(Ungültiges Objektmodul)

Ein Objektmodul weist nicht die richtige Form auf oder ist nicht vollständig (als wäre die Assemblierung in der Mitte abgebrochen worden).

SYMBOL DEFINED MORE THAN ONCE
(Symbol mehr als einmal definiert)

MS-LINK hat zwei oder mehr Moduln gefunden, die einen einzigen Symbolnamen definieren.

PROGRAM SIZE OR NUMBER OF SEGMENTS EXCEEDS CAPACITY OF LINKER
(Programmgröße oder Segmentanzahl überschreitet Kapazität des Linkprogramms)

Die Gesamtgröße darf 384 KB nicht überschreiten. Die Anzahl von Segmenten darf 255 nicht überschreiten.

REQUESTED STACK SIZE EXCEEDS 64 K
(Erforderliche Stapelgröße überschreitet 64 K)
Eine Größe von mehr als oder gleich 64 KB muß mit dem Schalter /STACK angegeben werden.

SEGMENT SIZE EXCEEDS 64 K
(Segmentgröße überschreitet 64 K)
64 KB ist die Höchstgrenze für die Adressierung.

SYMBOL TABLE CAPACITY EXCEEDED
(Kapazität der Symboltabelle überschritten)
Sehr viele und/oder sehr lange Namen wurden eingegeben und überschreiten die Grenze von ca. 25 KB.

TOO MANY EXTERNAL SYMBOLS IN ONE MODULE
(Zu viele externe Symbole in einem Modul)
Die Grenze beträgt 256 externe Symbole pro Modul.

TOO MANY GROUPS
(Zu viele Gruppen)
Die Grenze beträgt 10 Gruppen.

TOO MANY LIBRARIES SPECIFIED)
(Zu viele Bibliotheken angegeben)
Der Grenzwert beträgt 8 Bibliotheken.

TOO MANY PUBLIC SYMBOLS
(Zu viele öffentliche Symbole)
Der Grenzwert beträgt 1.024 öffentliche Symbole.

TOO MANY SEGMENTS OR CLASSES
(Zu viele Segmente oder Klassen)
Der Grenzwert beträgt 256 (Segmente und Klassen zusammengenommen).

UNRESOLVED EXTERNAL: <list>
(Nicht aufgelöste externe Symbole: <Liste>)
In den aufgelisteten Symbolen ist kein Modul unter den angegebenen Moduln oder Bibliotheksdateien vorhanden, in dem das Symbol definiert wird.

VM READ ERROR
(VM Lesefehler)
Hier handelt es sich um einen Plattenfehler; er wird nicht von MS-LINK verursacht.

WARNING: NO STACK SEGMENT
(Warnung: Kein Stapelsegment)

Keines der angegebenen Objektmoduln enthält eine Anweisung zur Zuordnung von Stapelplatz, der Benutzer hat jedoch den Schalter /STACK eingegeben.

WARNING: SEGMENT OF ABSOLUTE OR UNKNOWN TYPE
(Warnung: Absolutes oder unbekanntes Segment)

Entweder ist ein fehlerhaftes Objektmodul vorhanden oder es wurde versucht, Moduln zu binden, die MS-LINK nicht verarbeiten kann (z.B. ein nicht verschieblicher Modul).

WRITE ERROR IN TMP FILE
(Schreibfehler in TMP-Datei)

Auf der Platte ist kein Platz mehr vorhanden, um die VM.TMP-Datei zu erweitern.

WRITE ERROR ON RUN FILE
(Schreibfehler bei Ausführungsdatei)

In diesem Fall ist meistens nicht genügend Platz auf der Diskette für die Ausführungsdatei vorhanden.

BETRACHTUNGEN IM ZUSAMMENHANG MIT DER BENUTZUNG VON ZWEI BETRIEBSSYSTEMEN

Der NCR DECISION MATE V kann mit mehr als einem Betriebssystem benutzt werden. Der Doppelprozessor beispielsweise ermöglicht die Verarbeitung sowohl von 8-Bit- als auch von 16-Bit-Anwendungen. Wenn Sie beabsichtigen, sowohl MS-DOS als auch ein anderes Betriebssystem einzusetzen, müssen Sie für den Schutz Ihrer Datendateien und der anderen Plattensoftware Sorge tragen. Im allgemeinen besteht die Datensicherung einfach im einwandfreien Etikettieren der Disketten, so daß immer mit kompatiblen Disketten gearbeitet wird. Außerdem müssen Sie sicherstellen, daß nur kompatible Software und kompatible Datendateien auf derselben Diskette stehen.

Dieses Verfahren der 'Trennung' gilt auch für eine Festplatte. Bei ihr kann eine der logischen Einheiten für die Benutzung durch MS-DOS formatiert sein, während die andere für ein anderes Betriebssystem reserviert ist.

Bei richtigem Einsatz erhöhen sich die Verarbeitungskapazitäten durch den Einsatz von zwei Betriebssystemen ganz wesentlich. Allerdings sollten Sie folgende Richtlinien beachten:

- Kompatible Betriebssystemsoftware, Anwendersoftware und Daten müssen auf denselben Disketten stehen.
- Sämtliche Disketten müssen eindeutig etikettiert sein. Dies gilt insbesondere für die Festplatte, auf der die logischen Einheiten nicht sichtbar sind.
- Es darf nie ein Befehl benutzt werden, der den Inhalt einer Platte zerstören könnte, ohne vorher den Inhalt der Platte zu ermitteln. Wenn Sie sich des Inhalts nicht sicher sind, sollten Sie ihn mit Hilfe eines entsprechenden Befehls ermitteln (beispielsweise mit dem MS-DOS Befehl CHKDSK).
- Schließlich sollten Sie stets Sicherheitskopien der wichtigen Software und wichtiger Daten anfertigen.

PLATTENFEHLER

Kommt es während der Ausführung eines Befehls oder eines Programms zu einem Plattenfehler, so macht MS-DOS drei weitere Versuche, die Operation auszuführen. Kann die Operation nicht erfolgreich beendet werden, so gibt MS-DOS eine Fehlermeldung in folgendem Format aus:

<yyy> ERROR WHILE<I/O action> ON DRIVE x
Abort, Ignore, Retry:_

(<yyy> FEHLER BEI <E/A Aktion> IN LAUFWERK x
Abbrechen, ignorieren, erneut versuchen:_)

In dieser Meldung kann <yyy> eine der folgenden Bedeutungen haben:

- WRITE PROTECT
- NOT READY
- SEEK
- DATA
- SECTOR NOT FOUND
- WRITE FAULT
- DISK

Bei der <E/A-Aktion> kann es sich um eine der folgenden Aktionen handeln:

- READING
- WRITING

Das Laufwerk <x> gibt das Laufwerk an, in dem der Fehler gefunden wurde.

MS-DOS wartet, bis der Benutzer eine der folgenden Antworten eingegeben hat:

A (Abort) (Abbrechen)
Das Programm beenden, das eine Lese- oder Schreiboperation auf der Platte angefordert hat.

I (Ignore) (Ignorieren)

Den fehlerhaften Sektor ignorieren und so tun, als wäre der Fehler nicht vorhanden.

R (Retry) (Erneut versuchen)

Die Operation wiederholen. Sie müssen diese Antwort benutzen, wenn der Fehler behoben wurde (wie beispielsweise bei den Fehlern NOT READY oder WRITE PROTECT).

Normalerweise werden Sie eine Wiederherstellung versuchen, indem Sie die Antworten in folgender Reihenfolge eingeben:

R (für einen erneuten Versuch)
A (um das Programm zu beenden und eine neue Diskette zu benutzen)

Eine weitere Fehlermeldung kann sich auf eine fehlerhafte Plattenlese- oder Plattenschreib-Operation beziehen:

FILE ALLOCATION TABLE BAD FOR DRIVE x
(Fehlerhafte Dateizuordnungstabelle für Laufwerk x).

Diese Meldung bedeutet, daß die Kopie einer der Zuordnungstabellen im Speicher über Zeiger zu nicht vorhandenen Blöcken verfügt. Möglicherweise wurde die Platte falsch formatiert oder vor der Benutzung überhaupt nicht formatiert. Bleibt dieser Fehler bestehen, so können Sie die Platte erst benutzen, nachdem Sie sie formatiert haben.

ERWERB UND INSTALLATION DER SOFTWARE

RICHTIGE WAHL

Der NCR DECISION MATE V mit MS-DOS ist ein überaus flexibles System, das nahezu jede Anwendungs- und Sprachsoftware unterstützt, solange sie mit MS-DOS kompatibel ist. Die mit dem Computer zu benutzende Software kann in drei Kategorien unterteilt werden:

- Von NCR vertriebene Software.
- Von NCR in dem DECISION MATE V erprobte Software, die jedem MS-DOS Benutzer zur Verfügung steht.
- *Nicht* von NCR getestete Software, die jedoch jedem MS-DOS Benutzer zur Verfügung steht.

Die Softwarepakete der ersten Gruppe können Sie direkt von NCR über Ihren NCR-Beauftragten oder Vertragshändler erhalten. Die Softwarepakete der anderen Kategorien können von einem NCR-Vertragshändler oder jedem namhaften Softwarehaus erworben werden. Sie sollten sich jedoch vergewissern, daß die Software auch in Ihrem Computer eingegesetzt werden kann. Mit den im folgenden erläuterten Fragen können Sie sich an jeden fachkundigen Händler wenden.

1. Fragen Sie, ob die Software auf einer für NCR MS-DOS formatierten Diskette vorhanden ist.
2. Steht die Software nicht auf einer für NCR MS-DOS formatierten Diskette, so können Sie fragen, ob sie auf einer für IBM-PC formatierte Diskette verfügbar ist. (Eine 5 1/4 Zoll Diskette mit einer Kapazität von 160/180/320/360 KB.) Sie können jede dieser Disketten mit Ihrem NCR DECISION MATE V benutzen.
3. Bei den meisten Anwenderpaketen und bei mancher Sprachsoftware ist ein Installationsprogramm für die Verbindung mit der jeweiligen Hardware erforderlich. Deshalb sollten Sie fragen, ob die Software bereits für die Benutzung in einem NCR DECISION MATE V installiert ist, oder ob das Paket ein Installationsprogramm enthält, damit Sie die Software selbst anpassen können. Ist die Software für einen MS-DOS ANSI-

Treiber oder ein Lear Siegler Terminal installiert, so kann sie problemlos in dem Computer installiert werden (siehe nächsten Abschnitt).

INSTALLATION DER SOFTWARE

Um Ihnen eine volle Ausnutzung der Möglichkeiten Ihres NCR DECISION MATE V zu ermöglichen, muß jede Anwendung an den Computer angepaßt bzw. in ihm installiert werden. Dieses Installationsverfahren ist für jede Anwendung anders. Deshalb enthält jedes fertig erworbene Paket, wie beispielsweise MULTIPLAN™, sein eigenes Programm für die Ausführung dieser Aufgabe. In der mit der Anwenderdiskette gelieferten Dokumentation wird beschrieben, wie dieses Programm ausgeführt wird. Bevor die Anwendung an den jeweiligen Bedarf angepaßt wird, ist die Anfertigung einer Kopie der erworbenen Diskette zu empfehlen. Diese Kopie sollten Sie dann als Arbeitsdiskette benutzen. (Siehe FORMAT- und DISKCOPY-Befehle in Kapitel 5.) Die Originaldiskette sollte an einem sicheren Ort aufbewahrt und nur zur Anfertigung von Kopien benutzt werden.

Am Anfang zeigen die meisten Installationsprogramme eine Liste mit Computerterminals an. Enthält diese Liste NCR DECISION MATE V, so können Sie ihn wählen. Ansonsten ist ein MS-DOS ANSI-Treiber, das Lear Siegler ADM-31 oder das Lear Siegler ADM-3A Terminal zu wählen. Tabelle 1 gibt Aufschluß über die Funktionen des MS-DOS ANSI-Treibers. Tabelle 2 enthält entsprechende Angaben für Lear Siegler Terminals.

Wird keines der obigen Terminals aufgeführt, so müssen Sie die Terminaleigenschaften des NCR DECISION MATE V dem Installationsprogramm der Anwendung mitteilen. Bei der Eingabe dieser Codes zur Installation eines Anwenderpakets sollten Sie die Tabellen als Anleitung benutzen. (All diese Steuerzeichen/Folgen sind für die Benutzung durch Anwendungen und nicht für die Eingabe über die Tastatur gedacht.)

Die anderen Tabellen enthalten Informationen, die Sie je nach Anwendung benötigen. Tabelle 3 enthält verschiedene Informationen. In Tabelle 4 werden Noten, Frequenzen und Zyklen für die Programmierung von Musikanwendungen angegeben. Für weitere Informationen sind am Ende dieses Abschnittes Umwandlungs- und Übersetzungstabellen enthalten.

INFORMATIONEN ÜBER FUNKTIONSEIGENSCHAFTEN

Vor den folgenden Funktionen muß ein Umschaltzeichen <ESC> (Hexadezimalwert 1B) und ein Zeichen für eine linke eckige Klammer <[> (Hexadezimalwert 5B stehen.

TABELLE FÜR MS-DOS ANSI-TREIBER (TABELLE 1)

Funktion	ASCII-Zeichenfolge	Hexadezimal-Zeichenfolge
Position der Schreibmarke	(Zeile#);(Spalte#)H	(Zeile#)3B(Spalte#)48
oder	(Zeile#);(Spalte#)f	(Zeile#)3B(Spalte#)66
Schreibmarke nach links	(# Spalten)D	(# Spalten)44
Schreibmarke nach rechts	(# Spalten)C	(# Spalten)43
Schreibmarke nach unten	(# Zeilen)B	(# Zeilen)42
Schreibmarke nach oben	(# Zeilen)A	(# Zeilen)41
Bericht über Einheitenstatus	6n	366E
Ausgabe über Position der Schreibmarke (wird nach 6n Funktion zurückgegeben)	(Zeile#);(Spalte#)R	(Zeile#)3B(Spalte#)52
Position der Schreibmarke sichern	s	73
Position der Schreibmarke wieder herstellen	u	75
Bildschirm löschen	2J	324A
Bis Zeilenende löschen	K	4B
Funktionstaste definieren	0;(fk#);"Zeichenfolge"p	303B (fk#)3B22 Zeichenfolge 2270
Funktionstastenerweiterung deaktivieren	0;0p	303B3070
Funktionstastenerweiterung aktivieren	0;99p	303B393970

Die in Klammern angegebenen Informationen stellen die vom Benutzer eingegebenen Daten dar. Die Klammern werden nicht mit eingegeben.

Um der Funktionstaste 18 beispielsweise CHKDSK <CR> zuzuweisen, wird folgende Eingabe vorgenommen:

<ESC>[0;18;"CHKDSK";13p

wobei 13 für <CR> steht, Hexadezimalwert 0D.

FUNKTIONSCODES DER TERMINALS (TABELLE 2)

Funktion	ASCII-Zeichenfolge	Hexadezimal-Zeichenfolge	Terminal*
Position der Schreibmarke	<ESC> =	1B3D	A,B
Zeile + Versatz	(Zeile#+32)	(Zeile#+20hex)	A,B
Spalte + Versatz	(Spalte#+32)	(Spalte#+20hex)	A,B
Schreibmarke nach links	^H	08	A,B
Schreibmarke nach rechts	^L	0C	A,B
Schreibmarke nach unten	^J	0A	A,B
Schreibmarke nach oben	^K	0B	A,B
Bildschirm löschen und Schreibmarke in Ausgangsstellung	^Z	1A	A,B
Bis Zeilenende löschen	^W	17	
Zeilenschaltung	^M	0D	A,B
Umschaltfolge	<ESC>	1B	A,B
Akustisches Signal	^G	07	A,B
Schreibmarke in Ausgangsposition	^~	1E	
Vor den folgenden Funktionen muß ein Umschaltzeichen <ESC>, Hexadezimalwert 1B, stehen.			
Löschen bis zum Zeilenende	T oder t	54 oder 74	A
Löschen bis zum Bildschirmende	Y oder y	59 oder 79	A
Bildschirm löschen und Schreibmarke in Ausgangsposition	: oder *	3A oder 2A	
Halbe Leuchtstärke ein (Bei einem Farbbildschirm wird rot beigemischt)	)	29	A
Halbe Leuchtstärke aus	(	28	A
Bildumkehr ein	G4	4734	A
Blinken ein	G2	4732	A
Bildumkehr und Blinken aus	G0	4730	A
Zeile einfügen	E	45	A
Zeichen einfügen	Q	51	A
Zeile löschen	R	52	A
Zeichen löschen	W	57	A
Musik**	M	4D	

Die in Klammern angegebenen Informationen stellen die vom Benutzer angegebenen Daten dar. Die Klammern sind nicht einzugeben.

* A = Lear Siegler ADM-31; B = Lear Siegler ADM-3A

** Sie können für Ihren NCR DECISION MATE V Musik programmieren. Beim Empfang der Zeichenfolge <ESC>M akzeptiert der Bildschirmtreiber die nächsten beiden Zahlen als Frequenz bzw. Tonlänge. Sie können die entsprechende Frequenzeingabe bzw. Anzahl von Zyklen für jeden Halbton mit Hilfe der Tabelle 4 am Ende dieses Abschnitts ermitteln.

VERSCHIEDENE INFORMATIONEN – Tabelle 3

	ANSI	ADM-3A/ADM-31
Anzahl von Zeilen*	24 (1–24)	24 (0–23)
Anzahl von Spalten	80 (1–80)	80 (0–79)
Ausgangsposition der Schreibmarke	1/1	0/0
Eingabe-/Ausgabetechnik	MS-DOS-Aufrufe und -Befehle (z.B. TYPE)	MS-DOS-Aufrufe und -Befehle (z.B. TYPE)
Schreibmarke Ein/Aus	Nicht aktiv	Nicht aktiv
Tastatur-Quittungston Ein/Aus	Nicht aktiv	Nicht aktiv

* Die Darstellung am Bildschirm kann von einer 25. Zeile Gebrauch machen (z.B. bei Fehlermeldungen). Diese Zeile wird beim Aufrollen des Bildschirms nicht beeinflußt.

MUSIKCODES – Tabelle 4

Note	Frequenz			Zyklen
	Dezimal	Hexa-dezimal	Taste	
Pause	32	20	Leer-zeichen	–
A	33	21	!	110
A#	34	22	"	116.5
B	35	23	#	123.5
C	36	24	$	131
C#	37	25	%	138.6
D	38	26	&	146.8
D#	39	37	'	155.8
E	40	28	(	164.8
F	41	29	)	174.6
F#	42	2A	*	185

MUSIKCODES – Tabelle 4 (Fortsetzung)				
Note	Frequenz			Zyklen
	Dezimal	Hexa-dezimal	Taste	
G	43	2B	+	196
G#	44	2C	,	208
A	45	2D	–	220
A#	46	2E	.	233
B	47	2F	/	246.9
C (Mittl. C)	48	30	0	261.6
C#	49	31	1	277.4
D	50	32	2	293.7
D#	51	33	3	311
E	52	34	4	329.6
F	53	35	5	349.2
F#	54	36	6	370
G	55	37	7	392
G#	56	38	8	415
A	57	39	9	440
A#	58	3A	:	465
B	59	3B	;	493.9
C	60	3C	<	523.2
C#	61	3D	=	563
D	62	3E	>	587.3
D#	63	3F	?	622
E	64	40	@	659.3
F	65	41	A	698.5
F#	66	42	B	740
G	67	43	C	784
G#	68	44	D	830
A	69	45	E	880
A#	70	46	F	932
B	71	47	G	987.8
C	72	48	H	1046.5
C#	73	49	I	1108.7
D	74	4A	J	1174.7

Wenn Ihr NCR-DECISION MATE V einen Farbbildschirm hat, können Sie die Grafik-Zeichen der folgenden Tabelle entnehmen. (Die nicht farbbezogenen Zeichen gelten auch für den schwarz/-weißen Bildschirm.

FARB- und GRAFIKCODES — Tabelle 5

Funktion	Dezimal vorher 27(ESC)91([) abschließend 109(m)	Hexadezimal Code vorher 1BH 5BH abschließend 6DH	Taste
GRAPHIK-ATTRIBUTE AUS	48	30	0
HALBE LEUCHTDICHTE AUS	49	31	1
BLINKEN EIN	53	35	5
UMKEHRUNG DES BILDSCHIRMS EIN	55	37	7
HALBE LEUCHTDICHTE EIN	56	38	8
SCHWARZER VORDER-GRUND	51 48	33 30	3 0
ROTER VORDERGRUND	51 49	33 31	3 1
GRÜNER VORDER-GRUND	51 50	33 32	3 2
GELBER VORDERGRUND	51 51	33 33	3 3
BLAUER VORDERGRUND	51 52	33 34	3 4
MAGENTA VORDER-GRUND	51 53	33 35	3 5
ZYAN VORDERGRUND	51 54	33 36	3 6
WEISSER VORDER-GRUND	51 55	33 37	3 7
SCHWARZER HINTER-GRUND	52 48	34 30	4 0
ROTER HINTERGRUND	52 49	34 31	4 1
GRÜNER HINTERGRUND	52 50	34 32	4 2
GELBER HINTERGRUND	52 51	34 33	4 3
BLAUER HINTERGRUND	52 52	34 34	4 4
MAGENTA HINTER-GRUND	52 53	34 35	4 5
ZYAN HINTERGRUND	52 54	34 36	4 6
WEISSER HINTERGRUND	52 55	34 37	4 7

Die obigen Codes können Sie beliebig verketten, ohne anfangs 1BH 5BH zu wiederholen. 3BH muß als Trennzeichen zwischen jedem Element stehen.

Beachten Sie, daß die Funktion "GRAFIK-ATTRIBUTE AUS" keine Farbangabe beeinflußt, außer wenn es sich aus dem Rücksetzen der Umkehrung oder der Leuchtdichte-Attribute ergibt.

ÜBERSETZUNGS-/UMWANDLUNGS-INFORMATIONEN

In den verschiedenen Ländern gibt es sprachabhängig einige Sonderzeichen. Daher stimmen bei manchen Beispielen in diesem Buch die Schriftzeichen (eingegebene oder angezeigte) teilweise nicht mit den Zeichen auf der Tastatur überein. Die untenstehende Tabelle zeigt Ersatzzeichen, die bei gleichem Hexadezimal-Code die Anforderungen des Betriebssystems erfüllen. Für Benutzer, die die vollständige Hexadezimaltabelle benötigen, wird die ASCII-Codetabelle mit dem USASI-Code im Anschluß an die Tastaturtabelle angegeben.

Landessprache	Hex Codes und äquivalente Zeichen											
	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
US-Englisch	#	$	@	[	\	]	^	`	{	|	}	~
UK-Englisch	£	$	@	[	\	]	↑	`	{	|	}	~
Französisch	£	$	à	°	ç	§	^	`	é	ù	é	¨
Deutsch	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
Schwedisch/Finnisch	#	¤	@	Ä	Ö	Å	^	`	ä	ö	å	¨
Dänisch/Norwegisch	£	$	@	AE	Ø	Å	^	`	æ	ø	å	¨
Spanisch	£	$	@	¡	Ñ	¿	^	`	{	ñ	}	~
Italienisch	£	$	§	°	ç	é	^	ù	à	ò	è	ì
Schweiz (französ.)	£	$	ç	à	é	è	^	`	ä	ö	ü	¨
Kanada (engl.)	#	$	@	[	\	]	^	`	£	|	¢	¨
Kanada (französ.)	#	$	@	[	ç	]	^	`	é	è	¢	¨
Südafrika (afrikaans)	#	$	@	Ê	'N	Ë	^	`	é	'n	ë	¨
Portugiesisch	£	$	@	Ä	Õ	Ç	`	`	á	ò	ç	~
Jugoslawisch	#	$	Đ	Ć	Č	Š	Ž	đ	ć	č	š	ž

Länderspezifische Zeichen

DIE AUTOMATISCHE SYSTEM-KONFIGURATION

Ein Computer benötigt meistens eine Anpassung des MS-DOS-Betriebssystems an die jeweilige Hardware-Umgebung; z.B. an einen bestimmten Drucker. Es ist zweckmäßig, wenn diese Anpassung bereits beim Laden des Betriebssystems erfolgt.

Die MS-DOS-Konfigurationsdatei (CONFIG.SYS) stellt eine wesentliche Hilfe bei der Konfiguration Ihres Systems dar: Anhand dieser Datei können Sie bewirken, daß Ihre eigenen Treiberprogramme (s. "MS-DOS PROGRAMMER's MANUAL") bereits beim Laden in das Betriebssystem aufgenommen werden. Die Konfigurationsdatei ist eine ASCII-Datei, die lediglich aus einer Anzahl von Kommandos besteht, die unmittelbar nach dem Laden des Betriebssystems ausgeführt werden. Somit erfolgt dieses Laden des Systems gemäß folgender Beschreibung:

1. Der Initialisierungssektor wird von der Diskette gelesen. Dieser enthält Maschinencode-Befehle, die das Laden des eigentlichen Betriebssystems einleiten.
2. Das MS-DOS-Betriebssystem sowie das hardware-abhängige BIOS werden von der Diskette in den Maschinenspeicher übertragen.
3. Bestimmte Initialisierungsroutinen werden ausgeführt.
4. Eine weitere Initialisierungsroutine liest die Konfigurationsdatei (CONFIG.SYS), sofern sie sich auf der Diskette befindet. Hierbei kommt die Installation der Treiberprogramme zur Ausführung. Gleichzeitig werden andere Benutzer-Optionen berücksichtigt. Anschließend erhält der Prozessor für MS-DOS-Kommandos die Programmsteuerung.

ERSTELLEN UND ÄNDERN DER CONFIG.SYS-DATEI

Wenn Ihre MS-DOS-DISKETTE die Datei CONFIG.SYS nicht enthält, können Sie mit Hilfe des MS-DOS-Editors, EDLIN, eine entsprechende Datei erstellen. Anschließend sollten Sie diese Datei in Ihr Stammverzeichnis auf der Diskette übertragen.

Nachstehend finden Sie eine Liste der Kommandos, die eine CONFIG.SYS-Datei beinhalten muß:

BUFFERS = <Zahl>

<Zahl>, eine Ganzzahl 1...99, gibt die Größe des Pufferbereiches in Sektoren an, den MS-DOS beim Laden des Betriebssystems einrichtet. Diese Zahl richtet sich nach der jeweiligen Installation. Eine häufig benutzte Zahl ist 10. Wenn Sie auf dieses Kommando in Ihrer CONFIG.SYS-Datei verzichten, wird die Zahl 2 von MS-DOS angenommen.

FILES = <Zahl>

<Zahl>, eine Ganzzahl 1...99, gibt die Anzahl der eröffneten Dateien an, zu denen die XENIX-Systemaufrufe Zugriff haben sollten. Diese Zahl richtet sich nach der jeweiligen Installation. Eine häufig benutzte Zahl ist 10. Wenn Sie auf dieses Kommando in Ihrer CONFIG.SYS-Datei verzichten, wird die Zahl 8 von MS-DOS angenommen.

DEVICE = <Dateiname>

Das in der von <Dateiname> bezeichneten Datei enthaltene Treiberprogramm wird in die Systemaufstellung aufgenommen.

BREAK = <ON oder OFF>

Wenn Sie ON angeben, wird bei jedem Systemaufruf eine Prüfung auf CONTROL-C durchgeführt. Dies stellt eine verbesserte Möglichkeit beim vorzeitigen Beenden eines Programmes gegenüber früheren Versionen von MS-DOS dar. OFF setzt diese Prüfung außer Kraft. Wenn dieses Kommando in der CONFIG.SYS-Datei nicht vorhanden ist, wird OFF von MS-DOS angenommen (Vgl. Kapitel 5).

SHELL = <Dateiname>

Dieses Kommando bewirkt, daß der Prozessor für MS-DOS-Kommandos diejenigen Kommandos ausführt, die in der von <Dateiname> bezeichneten Datei enthalten sind.

Eine typische Konfiguration könnte folgenden Inhalt aufweisen:

```
BUFFERS = 10
FILES   = 10
DEVICE  = \BIN\NETWORK.SYS
BREAK   = ON
SHELL   = A:\BIN\COMMAND.COM A:\BIN/P
```

Gemäß den obengenannten Richtlinien wird BUFFERS sowie FILES jeweils der Wert 10 zugeordnet. Die Initialisierungsroutine der Betriebssystem sucht die Datei \BIN\NETWORK.SYS, damit das Treiberprogramm in das Betriebssystem aufgenommen werden kann. (Diese Datei wird in der Regel mit der Systemdiskette geliefert.) Wenn Sie eine eigene Datei als DEVICE einsetzen wollen, müssen Sie dafür Sorge tragen, daß diese Datei sich im Pfad befindet, den Sie im DEVICE-Kommando angeben.

Das oben angegebene Beispiel einer Konfigurationsdatei weist MS-DOS an, die auf der Diskette im Laufwerk A befindliche Datei COMMAND.COM des Verzeichnisses \BIN als Gegenstand des MS-DOS EXEC-Kommandos zu betrachten. Der Zusatz A:\BIN teilt der Datei COMMAND.COM mit, wo sie selbst zu finden ist. Diese Information ist unerläßlich, wenn diese Datei nicht nur einmal von der Diskette gelesen werden soll. Der Zusatz /P teilt der Datei COMMAND.COM mit, daß sie die erste auszuführende Datei ist, und daß sie infolgedessen für die Bearbeitung des MS-DOS EXIT-Kommandos verantwortlich ist.

ERGÄNZUNG

EINFÜHRUNG

Diese Ergänzung zu Ihrem Handbuch enthält die Beschreibung einiger Anwendungsprogramme und Demonstrationssoftwarepakete, die mit dem NCR DECISION MATE V eingesetzt werden können.

Wenn Sie EIN Disketten-Laufwerk haben, gehen Sie die folgenden Schritte, um Software zu starten. — Haben Sie ZWEI Disketten-Laufwerke, benützen Sie die Anweisungen in Klammern.

1. Schieben Sie die MS-DOS-Diskette ein. Drücken Sie die <CR>-Taste.
   (ZWEI Disketten-Laufwerke: Schieben Sie die MS-DOS-Diskette ins Laufwerk A ein und die zusätzliche Diskette in B).
2. Am Bildschirm erscheint die Systemmeldung A>. Entfernen Sie die MS-DOS-Diskette aus dem Laufwerk. Schieben Sie die zusätzliche Diskette ein. Geben Sie den Namen der Anwendung ein und drücken Sie die <CR>-Taste.
   (ZWEI: Geben Sie B> ein und drücken Sie die <CR>-Taste. Wenn B> am Bildschirm erscheint, geben Sie den Namen der Anwendung ein und drücken Sie die <CR>-Taste).

Für CATCHUM wird beispielsweise:

A> CATCHUM <CR>

eingegeben. Die Instruktionen für das Computerspiel werden dann auf dem Bildschirm angezeigt.

LADDER

LADDER ist ein Spielprogramm für Ihren NCR DECISION MATE V. Durch Eingabe von LADDER <CR> können Sie das Programm aufrufen. Ein "Hauptmenu" erscheint am Bildschirm.

Wenn Sie "I" eingeben, können Sie eine Spielbeschreibung lesen. Nach Drücken von <CR> erscheint wieder das Hauptmenu. Von diesem aus können Sie

mit P das Spiel beginnen
mit L die Schwierigkeitsstufe ändern

mit I die Spielbeschreibung lesen
mit E das Spiel beenden.

Geben Sie nun "P" ein, um das Spiel zu beginnen. Sie können zu jeder Zeit das Spiel beenden, indem Sie ↑ C (Drücken der C-Taste bei gedrückter CONTROL-Taste) eingeben.

CATCHUM

CATCHUM ist ein Spiel für einen oder zwei Spieler. Nachdem Sie CATCHUM <CR> eingegeben haben, erscheint das Hauptmenu.

Wenn Sie nun "I" eingeben, können Sie eine Spielbeschreibung lesen. Durch Drücken von <CR> kehren Sie dann zum Hauptmenu zurück. Jetzt können Sie:

mit 1 bzw. 2 die Zahl der Spieler angeben
mit L die Schwierigkeitsstufe ändern
mit I die Spielbeschreibung lesen
mit E das Spiel beenden.

Wenn Sie zu zweit spielen, werden Sie auch dem Computer mitteilen müssen, welcher Spieler gerade spielen will.

Sie können das Spiel zu jeder Zeit mit ↑ C (Drücken der C-Taste bei gedrückter CONTROL-Taste) beenden.

HINWEIS: In den Spielprogrammen LADDER und CATCHUM können Sie das Spielobjekt mit folgenden Tasten in die in Klammern angegebenen Richtungen bewegen: 8 (↑), 2 (↓), 4 (←), 6 (→).

DEMO5

DEMO5 erzeugt eine sich bewegende grafische Darstellung am Bildschirm, damit Sie die asugezeichnete Bildschirm-Auflösung und Verarbeitungsgeschwindigkeit Ihres NCR DECISION-MATE V erleben können. Sie können dieses Programm mit DEMO5 <CR> aufrufen. Sie können zu jeder Zeit das Programm durch ↑ C (Drükken der C-Taste bei gedrückter CONTROL-Taste) beenden.

CLOK

CLOK erzeugt am Bildschirm eine Uhr, die nach anfänglichem Einstellen immer die richtige Uhrzeit anzeigt. Sie können dieses Programm mit CLOK <CR> aufrufen. Zunächst werden Sie auf-
C-Taste bei

MM,DD,YY <CR>

Das bedeutet, daß Sie zuerst eine Zahl für den Monat, dann eine für den Tag und schließlich die Jahreszahl eingeben müssen. Diese drei Angaben werden jeweils durch Kommata getrennt (Beispiel: 4,26,83). Diese Eingabe schließen Sie mit <CR> ab.

Anschließend geben Sie die jetzige Uhrzeit ebenfalls mit Komma in der folgenden Reihenfolge ein:

HH,MM <CR> (d.h. Stunde, Minute. Beispiel: 12,31)

Sie können zu jeder Zeit das Programm durch ↑ C (Drücken der C-Taste bei gedrückter CONTROL-Taste beenden.

MUSIC

MUSIC kann Ihnen elf verschiedene Lieder in Ihrem NCR DECISION MATE V vorspielen. Geben Sie MUSIC <CR> ein. Die Liederauswahl erscheint am Bildschirm, und Sie können eine der elf Melodien (1-9, A oder B) aussuchen. Drücken Sie einfach die entsprechende Ziffer bzw. den entsprechenden Buchstaben an der Tastatur (ohne <CR>). Sie hören dann die gewählte Melodie. Anschließend erscheint wieder die Liederauswahl. Sie können eine neue Melodie aussuchen oder das Programm mit E (nicht ↑ C) beenden.

VEGAS

VEGAS (Very Easy Graphic Application System) ist ein vielseitiges Programm, mit dem Sie einen tabellarischen Zusammenhang z.B. aus dem Wirtschafts- oder Geschäftsbereich veranschaulichen können. Mit Hilfe dieses Programms können Sie Linien-, Balken- und kreisförmige Diagramme erstellen und sie an einem Drucker ausgeben. Dieses Programm, dessen eigentlicher Name "NCR

Business Graphics" ist, wurde in MS-BASIC® erstellt. Hinzu kommt die grafische Erweiterung von NCR. Das Programm müssen Sie mit VEGAS <CR> aufrufen, worauf das Hauptmenu erscheint.

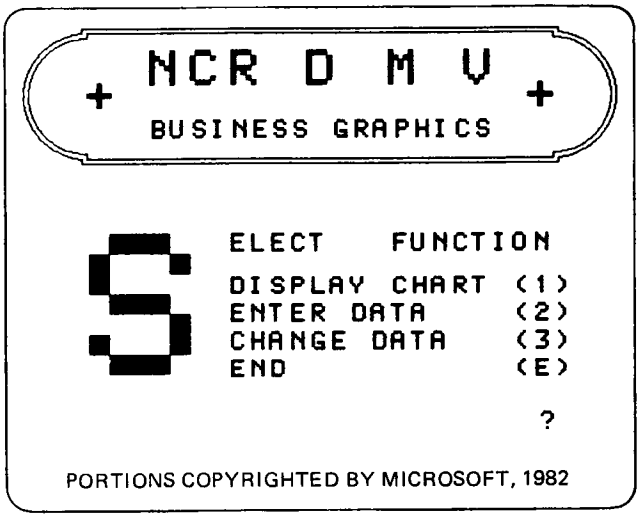


Abb. 1 Hauptmenu

Das Drücken von 1 zeigt ein bereits erstelltes Diagramm an. 2 gibt Ihnen die Gelegenheit, Daten zwecks Erstellens eines Diagramms einzugeben. 3 ermöglicht die Änderung bereits eingegebener Daten.

Vorausgesetzt, daß Sie VEGAS soeben von der Platte geladen haben, erscheint die in Abb. 2 dargestellte Aufforderung:

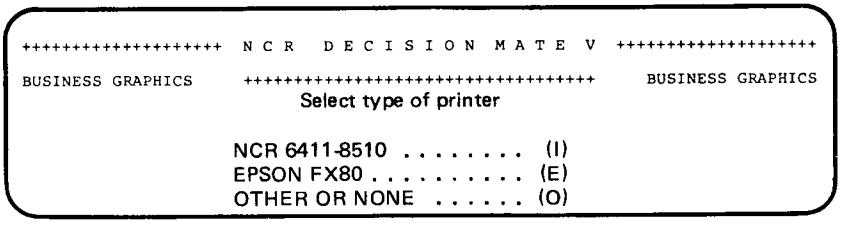


Abb. 2 Drucker-Auswahl

Jetzt können Sie dem Programm mitteilen, welcher Drucker angeschlossen ist.

Geben Sie I ein, wenn Ihr Drucker ein NCR 6411-8510 (ITOH M8510A) ist. Ein EPSON FX80 erfordert die Eingabe E. Falls Sie einen anderen oder keinen Drucker haben, geben Sie O ein.

HINWEISE: Bevor Sie Ihre Arbeit mit NCR Business Graphics fortsetzen, sei auf folgende Punkte hingewiesen:

- Wenn nur ein einziges Zeichen als Antwort angefordert wird, darf (CR) nicht gedrückt werden.
- Nachdem Sie eine Dateneingabe abgeschlossen haben, wird in der Regel "PRESS (CR) TO CONTINUE OR "R" TO RE-ENTER?" am Bildschirm erscheinen. Die Eingabe von R setzt die Dateneingabe zum Anfang des Bildschirms zurück. Sie können dann die Daten für diese Bildschirmseite neu eingeben.
- Rows (Items) sind (waagrechte) Zeilen am Bildschirm.
- Current Chart ist das zuletzt von Ihnen benutzte Diagramm.

Eingabe von Daten

Wenn Sie neue Daten zwecks Erstellens eines neuen Diagramms eingeben wollen, haben Sie im Hauptmenu 2 eingegeben. Am Bildschirm (Abb. 3) werden die Angaben angefordert, die für ein Diagramm erforderlich sind.

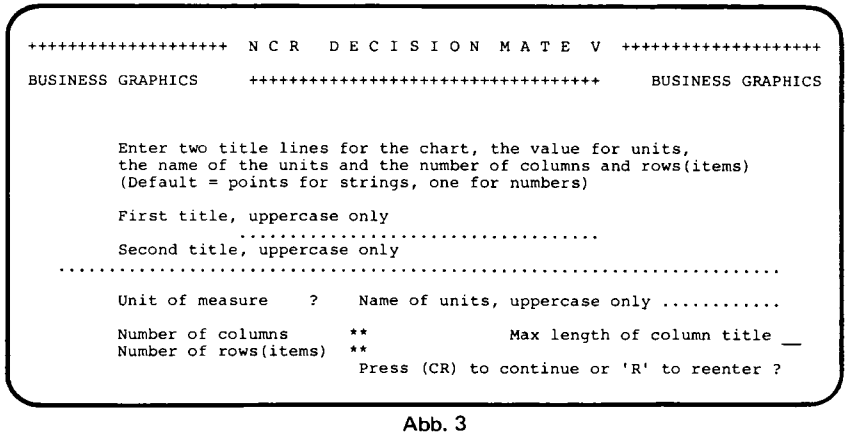


Abb. 3

Diese Fragen werden einzeln gestellt und müssen von Ihnen beantwortet werden:

First Title:	Die erste Überschrift; bis zu 36 Großbuchstaben; kein Komma
Second Title:	Die zweite Überschrift; bis zu 72 Großbuchstaben; kein Komma

Unit of Measure:	Größenordnung der Zahlen; Eingabe 0-9; Standardwert = 0; (0 = *1, 1 = *10, 2 = *100, 3 = *1000 usw.)
Name of Units:	Bezeichnung der Einheiten; bis zu 12 Großbuchstaben (z.B. DOLLAR, STUECK, usw.)
Number of Columns:	Anzahl der (senkrechten) Spalten; bis zu 48; Standardwert = 1
Number of Rows (Items):	Anzahl der (waagrechten) Zeilen; bis zu 12; Standardwert = 1

Wenn Sie anstelle einer Zahleneingabe nur <CR> drücken, wird der angegebene Standardwert vom Computer verstanden. Wenn Sie durch Drücken von <CR> auf die Eingabe eines angeforderten Buchstabentextes verzichten, wird eine Reihe von Punkten (. . . .) verstanden. Sobald Ihre Dateneingaben für diese Bildschirmseite vollständig sind, werden Sie aufgefordert, <CR> zu drücken. Auf Grund Ihrer bisherigen Eingaben wird eine zulässige Spaltenbreite in Zeichen vom Programm errechnet, die Ihnen dann mitgeteilt wird.

Als nächstes müssen Sie die Bezeichnungen für dir verschiedenen Spalten (Abb. 4) eingeben (bis zu insgesamt 59 Zeichen, vorbehaltlich der oben erwähnten Beschränkung der Spaltenbreite). Nach jeder Bezeichnung drücken Sie <CR>.

```
++++++++++++++++++++  N C R   D E C I S I O N   M A T E   V  ++++++++++++++++++++
BUSINESS GRAPHICS     ++++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS

        Enter column titles, use uppercase only
        (Default = points)

             ..............................


Column no. 1                    Press (CR) to continue or 'R' to reenter ?
```

Abb. 4

Die nächste Eingabe betrifft die Zeilen (siehe Abb. 5).

Row Titles:	Bezeichnungen der Zeilen; bis zu 12 Großbuchstaben, u.U. weniger, je nach Anzahl der Zeilen

Shade: Druckmuster-Kennzahl (siehe Ende dieses Abschnitts); Eingabe 0-15 zulässig; Standardwert = 0

```
++++++++++++++++++++ N C R  D E C I S I O N  M A T E  V ++++++++++++++++++++
BUSINESS GRAPHICS    ++++++++++++++++++++++++++++++++++++    BUSINESS GRAPHICS


        Enter row(item) titles, use uppercase only
        (Default = points)


.............
Shade c.    **



row(item) no.: 1              Press (CR) to continue or 'R' to reenter ?
```

Abb. 5

Anschließend müssen Sie den gewünschten Wert zu jeder (Tabellen-Position) eingeben. Das Programm zeigt zu jeder Position das Druckmuster an. Ihre Angaben ersetzen die jeweils erscheinenen Sternzeichen am Bildschirm (siehe Abb. 6).

```
                                ENTRY OF CHART VALUES

    ITEM TITLE/CD/CO                      COL 1
(item title, shade,       displayed)      *****









                               Press (CR) to continue or 'R' to reenter ?
```

Abb. 6

Values: Sie können zeilenweise die von Ihnen gewünschten Werte (nur Ganzzahlen $\leqslant 32760$) eingeben.

Ihre Dateneingabe für die Erstellung eines Diagramms ist somit vollständig. Das in Abb. 7 abgebildete Menu wird am Bildschirm ausgegeben. Sie können Ihre Daten auf Diskette sichern lassen (Eingabe 1), erneut vom Anfang an Daten eingeben (2), die Daten

am Drucker ausgeben lassen (3), oder zum Hauptmenu zurückkehren (G).

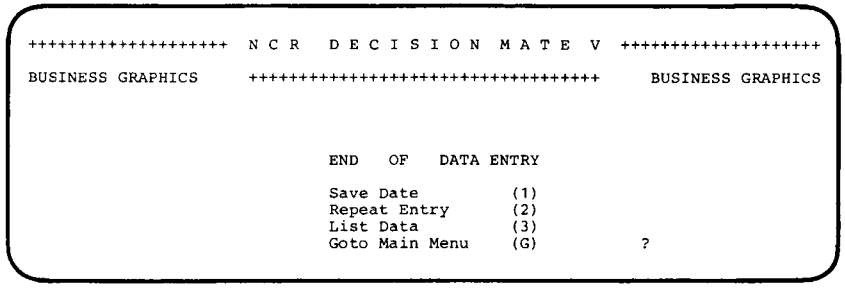


Abb. 7

Am sinnvollsten ist zunächst die Ausgabe am Drucker, damit Sie die Gelegenheit haben, Ihre eingegebenen Daten zu prüfen. Dann erscheint folgende Meldung am Bildschirm:

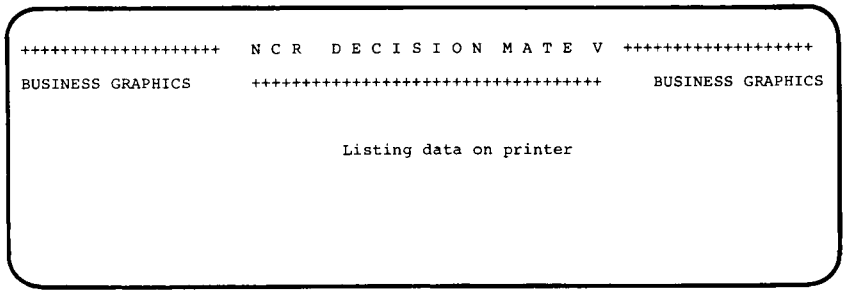


Abb. 8

Sobald der Drucker seine Arbeit beendet hat, wird das Auswahlmenu (Abb. 7) wiederholt. Wenn Ihre Daten keine Fehler enthalten, sollten Sie sie auf Diskette sichern. Falls Nachbesserungen erforderlich sind, bieten sich zwei Möglichkeiten an: Die Eingabe 2 bedeutet, daß Sie ganz von vorne anfangen wollen. Deshalb ist es bei nicht allzu umfangreichen Nachbesserungen sinnvoller, zuerst die Daten auf Diskette zu sichern (Eingabe 1). Sie können dann unter Verwendung von "CHANGE DATA" vom Hauptmenu aus (siehe Abb. 1) die erforderlichen Änderungen vornehmen. Die genaue Funktionsweise wird später in diesem Abschnitt erläutert.

Bevor die Daten auf Diskette geschrieben werden, müssen Sie bestimmen, ob die zuletzt vom Programm angelegte Datei von den neuen Daten überschrieben werden, oder ob VEGAS eine neue, zusätzliche Datei für die Daten eröffnen soll. Nach der Frage am Bildschirm "REPLACE (R) LAST RECORD IN FILE OR WRITE (N) RECORDS?" müssen Sie also R (Überschreiben) bzw. N (Neu anlegen) eingeben. Die nächste Frage — "PRESS (CR) TO CONTINUE OR "R" to REENTER?" — gibt Ihnen die Möglichkeit, diese Entscheidung durch Drücken von R zu revidieren. Wenn Sie mit Ihrer vorherigen Entscheidung einverstanden sind, drücken Sie einfach <CR>. Das Programm teilt Ihrer Datei eine Nummer zu, die selbstverständlich auch Ihnen mitgeteilt wird (siehe Abb. 9).

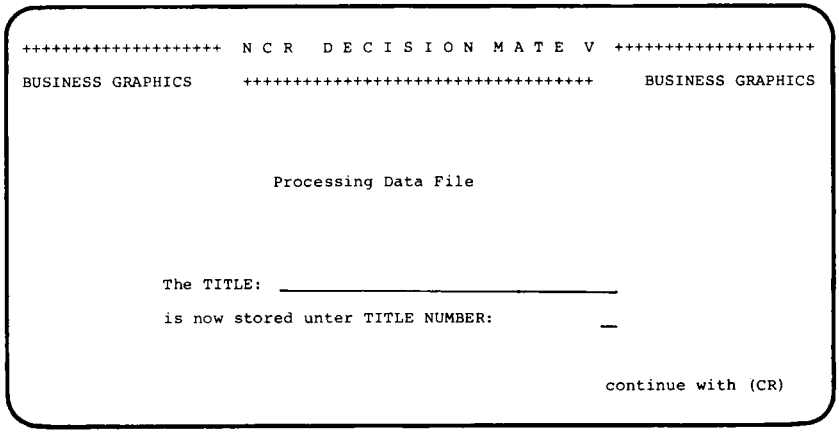


Abb. 9

Notieren Sie diese Nummer sowie den Verwendungszweck der soeben gesicherten Datei. Diese Information ist für eine zukünftige Benutzung dieser Datei unerläßlich. Von dem in Abb. 7 abgebildeten Menu aus können Sie durch Drücken von G zum Hauptmenu (Abb. 1) zurückkehren.

Erstellen eines Diagramms

Im Hauptmenu wählen Sie die Funktion 1 "DISPLAY CHART" (s. Abb. 1). Die Aufgabe dieser Funktion besteht darin, nach Ihrem Wunsch ein Linien-, Balken- oder kreisförmiges, "kuchenähnliches" Diagramm am Bildschirm auszugeben. Wenn Daten, die zu einem Diagramm gehören, sich noch im Maschinenspeicher befinden, werden Sie mit "USE CURRENT CHART? (Y/N)" gefragt, ob Sie diese Daten für das Diagramm benutzen wollen. Wenn

ja, dann geben Sie Y ein. Das Menu für die Darstellungsformen (Abb. 11) wird dann ausgegeben. Wenn nein, oder im Falle, daß sich keine Daten im Maschinenspeicher befinden, werden Sie aufgefordert, die der gewünschten Datei von VEGAS zugeteilte Nummer einzugeben (Abb. 10).

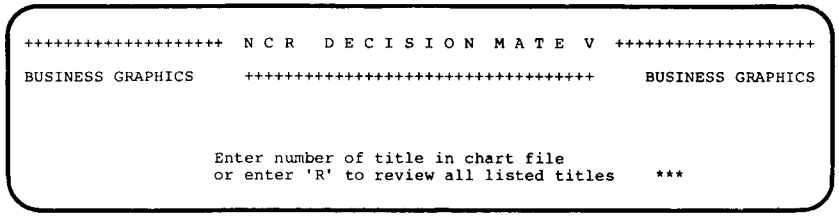


Abb. 10

Sollten Sie die Nummer der gewünschten Datei vergessen haben, können Sie durch Eingabe von R (mit anschließendem <CR>) eine Liste sämtlicher von VEGAS gesicherten Dateien am Bildschirm ausgeben lassen. (Ihre Version von VEGAS enthält unter Nr. 2 Beispieldaten für ein Diagramm.) Nach Einsicht in diese Liste drücken Sie <CR>. Es erscheint das Hauptmenu (Abb. 1), wo sie noch einmal die Funktion 1 ("DISPLAY CHART") wählen können.

Sobald Sie dem Programm die Nummer der gewünschten Date mitgeteilt haben, werden Sie gefragt, ob die Daten als Linien- (Eingabe 1), Balken- (2) oder kreisförmiges (3) Diagramm dargestellt werden sollen (Abb. 11).

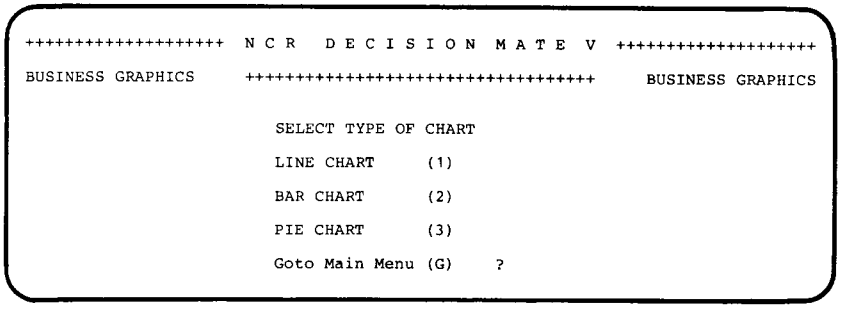


Abb. 11

Die Darstellung als Liniendiagramm erfordert zunächst die Eingabe einiger Parameter (Abb. 12).

Als erstes müssen Sie die erste und die letzte darzustellende Spalte ("column") eingeben. Beispiel: Gesetzt den Fall, daß Ihre

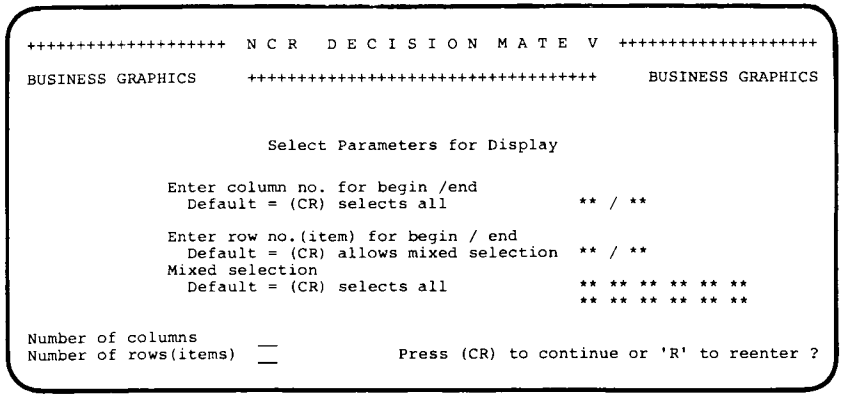


Abb. 12

dem Diagramm zugrundeliegende Datei Daten für 4 Spalten enthält, daß Sie aber lediglich die Spalten 3 und 4 darstellen wollen. In diesem Fall müssen Sie 3 <CR> und dann 4 <CR> eingeben. Wenn Sie auf diese Eingaben durch Drücken von <CR> verzichten, werden alle Spalten im Diagramm erscheinen.

Der nächste Schritt ist die Eingabe der ersten und der letzten Zeile ("row"), die ausgegeben werden soll. Beispiel: Wenn Ihre Datei Daten für 3 Zeilen enthält, können Sie wie bei den Spalten eine, zwei oder alle drei ausgeben lassen. Das Drücken von <CR> bewirkt, daß alle Zeilen in die Darstellung einbezogen werden, es sei denn, Sie machen von dem darauffolgenden Angebot einer gemischten Zeilenauswahl ("Mixed selection") Gebrauch. Hierdurch können Sie nach Belieben eine andere Reihenfolge der Zeilenausgabe bestimmen. Beispiel: Sie könnten die Zeilen in der Reihenfolge 3-1-2 ausgeben lassen. Ein einfaches Drücken von <CR> ohne vorherige Bestimmung einer gemischten Zeilenauswahl sorgt endgültig dafür, daß alle Zeilen ausgegeben werden. Nach nochmaligem Drücken von <CR> erscheint das Diagramm.

Bevor ein Balkendiagramm ausgegeben werden kann, muß das Programm wissen, ob die Balken aufeinandergestapelt ("stacked" = Eingabe 1) oder nebeneinander ("Side by side" = Eingabe 2) dargestellt werden sollen. Anschließend können Sie, wie beim Liniendiagramm beschrieben, die Parameter für Spalten und Zeilen eingeben (Abb. 13).

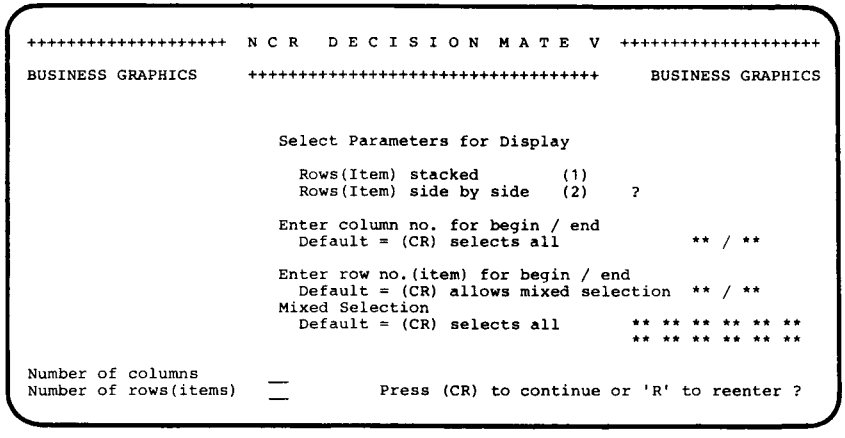


Abb. 13

Nach diesen Eingaben drücken Sie noch einmal <CR>. Das gewünschte Diagramm erscheint am Bildschirm.

Die kreisförmige, "kuchenähnliche" Darstellung darf aus bis zu 12 "Schnitten" bestehen. Zunächst müssen Sie zwischen zwei verschiedenen Darstellungsarten wählen: Sie können bestimmen, ob eine vollständige Spalte ("Several rows ..." = Eingabe 1) oder eine Zeile quer durch mehrere Saplten ("Several columns ..." = Eingabe 2) Gegenstand der Darstellung sein soll. Wenn Sie sich für eine Spalte entscheiden (s. Abb. 14), geben Sie anschließend die Nummer dieser Spalte ein. Als nächstes erwartet das Programm wie bei der Linien- und Balkendarstellung Ihre Angaben zur ersten und letzten Zeile, die in die Darstellung einbezogen werden soll. Und wie bei der Linien- und Balkendarstellung besteht ebenfalls die Möglichkeit einer gemischten Zeilenauswahl.

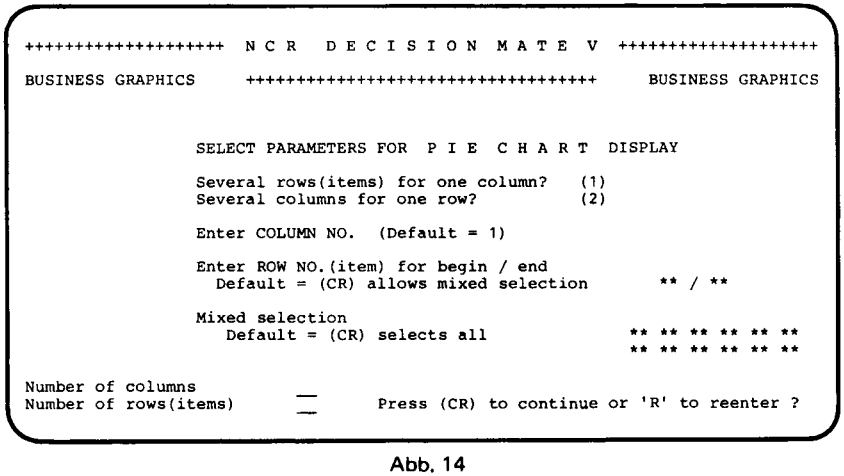


Abb. 14

Wenn Sie sich für eine Zeile quer durch mehrere Spalten entscheiden (Abb. 15), müssen Sie natürlich die Nummer der gewünschten Zeile eingeben. Danach machen Sie die übrigen Angaben zur Spaltenauswahl. Auch hier ist eine gemischte Auswahl möglich.

```
++++++++++++++++++++ N C R  D E C I S I O N  M A T E  V ++++++++++++++++++++

BUSINESS GRAPHICS     ++++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                SELECT PARAMETERS FOR  P I E  C H A R T  DISPLAY

                Several rows(items) for one column?      (1)
                Several columns for one row?             (2)

                Enter ROW NO.(item)    (Default = 1)

                Enter COLUMN NO. for begin / end
                  Default = (CR) allows mixed selection        ** / **

                Mixed selection
                  Default = (CR) selects all                ** ** ** ** ** **
                                                            ** ** ** ** ** **

Number of columns
Number of rows(items)          Press (CR) to continue or 'R' to reenter ?
```

Abb. 15

Bei der "kuchenähnlichen" Darstellung wird rechts des kreisförmigen Diagramms eine Lesehilfe ausgegeben. Hier können Sie die Bezeichnung der gewählten Spalte bzw. Zeile, die dazugehörigen Werte sowie deren Anteil in % des Ganzen erfahren. Anteile, die einzeln weniger als 1,5% des Ganzen ausmachen, werden in dem eigentlichen kreisförmigen Diagramm unter der Bezeichnung "MISC" (= sonstige) zusammengefaßt. Die Lesehilfe listet aber solche Werte einzeln auf.

Am Bildschirm unten links werden Sie bei jedem Diagramm einen Hinweis auf zusätzliche Ausgabemöglichkeiten bemerken. Durch die Eingabe von C können Sie eine sogenannte "Inversion" der Bildschirmausgabe bewirken: Zeichen und Zeichnungen, die vorher hell auf dunklem Hintergrund abgebildet wurden, erscheinen nun dunkel auf hellem Hintergrund. Ein nochmaliges Eingeben von C macht diese Inversion rückgängig. Mit P oder D wird Ihr Diagramm am Drucker ausgegeben, wobei D einen besonders hervorstechenden Druck durch zweimaliges Drucken bewirkt.

Falls Sie einen Plotter angeschlossen haben, erscheinen die Zeichen und Zeichnungen schwarz auf weißem Hintergrund, wenn die Inversion der Bildschirmausgabe außer Kraft ist.

Bei der kreisförmigen Darstellung kann auf Wunsch ein bestimmter Schnitt des "Kuchens" herausgetrennt werden. Dazu

muß die Bildschirmausgabe im invertierten Zustand sein. Sie müssen dann lediglich die Nummer des herauszutrennenden Schnittes mit entsprechendem Tastendruck angeben. Sie können beliebig viele Schnitte heraustrennen lassen. Diese können aber während der laufenden Diagrammausgabe nicht wieder eingesetzt werden.

Änderung von Daten

Die dritte Auswahlmöglichkeit im Hauptmenu (Abb. 1) — "CHANGE DATE" — ermöglicht die Änderung von Daten, die bereits vom Programm als Datei auf Diskette geschrieben wurden. Geben Sie 3 ein, um diese Funktion aufzurufen. Wenn Daten, die zu einem Diagramm gehören, sich noch im Maschinenspeicher befinden, werden Sie mit "USE CURRENT CHART? (Y/N)" gefragt, ob Sie diese Daten ändern wollen. Wenn ja, dann geben Sie Y ein. Das Änderungsverfahren wird dann eingeleitet (Abb. 17). Wenn nein, oder im Falle daß sich keine Daten im Maschinenspeicher befinden, werden Sie aufgefordert, die der gewünschten Datei von VEGAS zugeteilte Nummer einzugeben (Abb. 16).

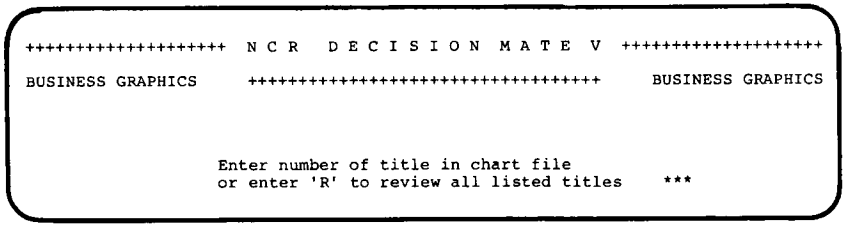


Abb. 16

Sollten Sie die Nummer der gesuchten Datei vergessen haben, können Sie durch Eingabe von R (mit anschließendem <CR>) eine sämtlicher von VEGAS gesicherten Dateien am Bildschirm ausgeben lassen. Nach Einsicht in diese Liste drücken Sie <CR>. Es erscheint das Hauptmenu (Abb. 1), wo Sie noch einmal die Funktion 3 ("CHANGE DATA") wählen können'

Die bisherigen Daten für Ihr Diagramm erscheinen einzeln am Bildschirm. Sie haben die Gelegenheit, an Stelle der Schreibmarke (oft als "Cursor" bezeichnet) eine Änderung einzugeben. An Stellen, wo keine Änderung getätigt werden soll, drücken Sie einfach <CR>. Die Schreibmarke springt dann zum nächsten Datenfeld. Es darf lediglich die Anzahl der Spalten sowie der Zeilen nicht geändert werden. Ferner sollten Sie beachten, daß eine pauschale Änderung der Größenordnung der Werte ("Unit of measure") eine Änderung sämtlicher Einzelwerte erforderlich macht.

Die folgenden Abbildungen geben Ihnen detaillierten Aufschluß über die Änderungsmöglichkeiten.

Sie haben die Möglichkeit, die übergeordnete sowie die zweite Überschriftszeile zu ändern (Abb. 17).

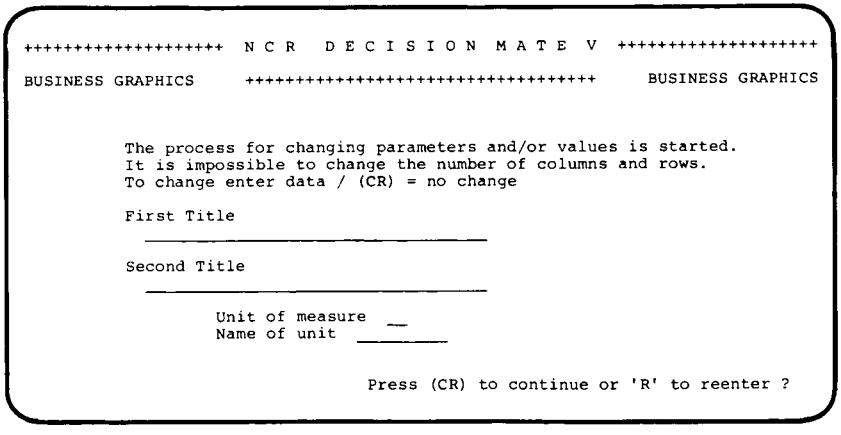


Abb. 17

Änderungen der Spaltenbezeichnungen (Abb. 18) sowie der Druckmuster der einzelnen Zeilen (Abb. 19) sind ebenfalls zulässig.

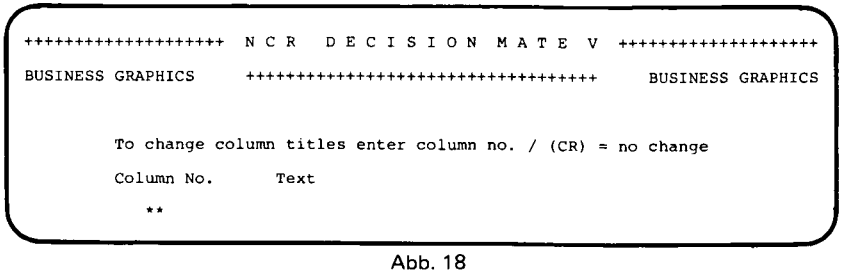


Abb. 18

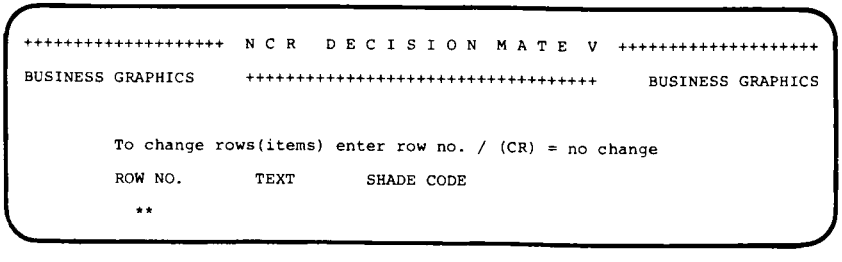


Abb. 19

Zum Schluß können Sie die Werte der einzelnen (tabellarischen) Positionen ändern (Abb. 20). Geben Sie die Nummer für Spalte und Zeile und den neuen Wert für diese Position ein. Nach der letzten Änderung geben Sie <CR> ohne Zahl ein.

```
++++++++++++++++++++  N C R   D E C I S I O N   M A T E   V  ++++++++++++++++++++

BUSINESS GRAPHICS     ++++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


        To change the value of a row(item) within a column enter
        the column no. and the row no. / (CR) = no change

                        Column No. Row No.    Value

                           **
```

Abb. 20

Nach Abschluß der oben beschriebenen Änderungen erscheint folgendes Menu:

```
++++++++++++++++++++  N C R   D E C I S I O N   M A T E   V  ++++++++++++++++++++

BUSINESS GRAPHICS     ++++++++++++++++++++++++++++++++++++     BUSINESS GRAPHICS


                                End of Change

                              Save Changes     (1)
                              Repeat Changes   (2)
                              List Changes     (3)
                              Goto Main Menu   (G)    ?
```

Abb. 21

Sie können durch Drücken von
1 die geänderte Datei auf Diskette sichern
2 das Änderungsverfahren erneut einleiten
3 die Änderungen am Drucker ausgeben lassen
G zum Hauptmenu (Abb. 1) zurückkehren.

Nachdem Sie die Änderungen mit Hilfe einer Druckausgabe noch einmal überprüft haben, können Sie Ihre Datei in der neuen geänderten Form auf Diskette sichern. Das Sichern wird dann am Bildschirm bestätigt (Abb. 22).

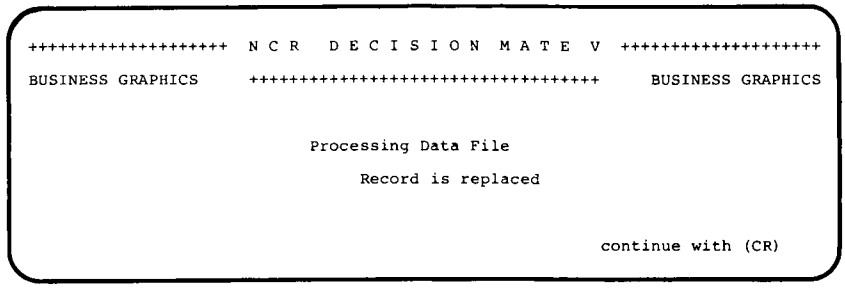
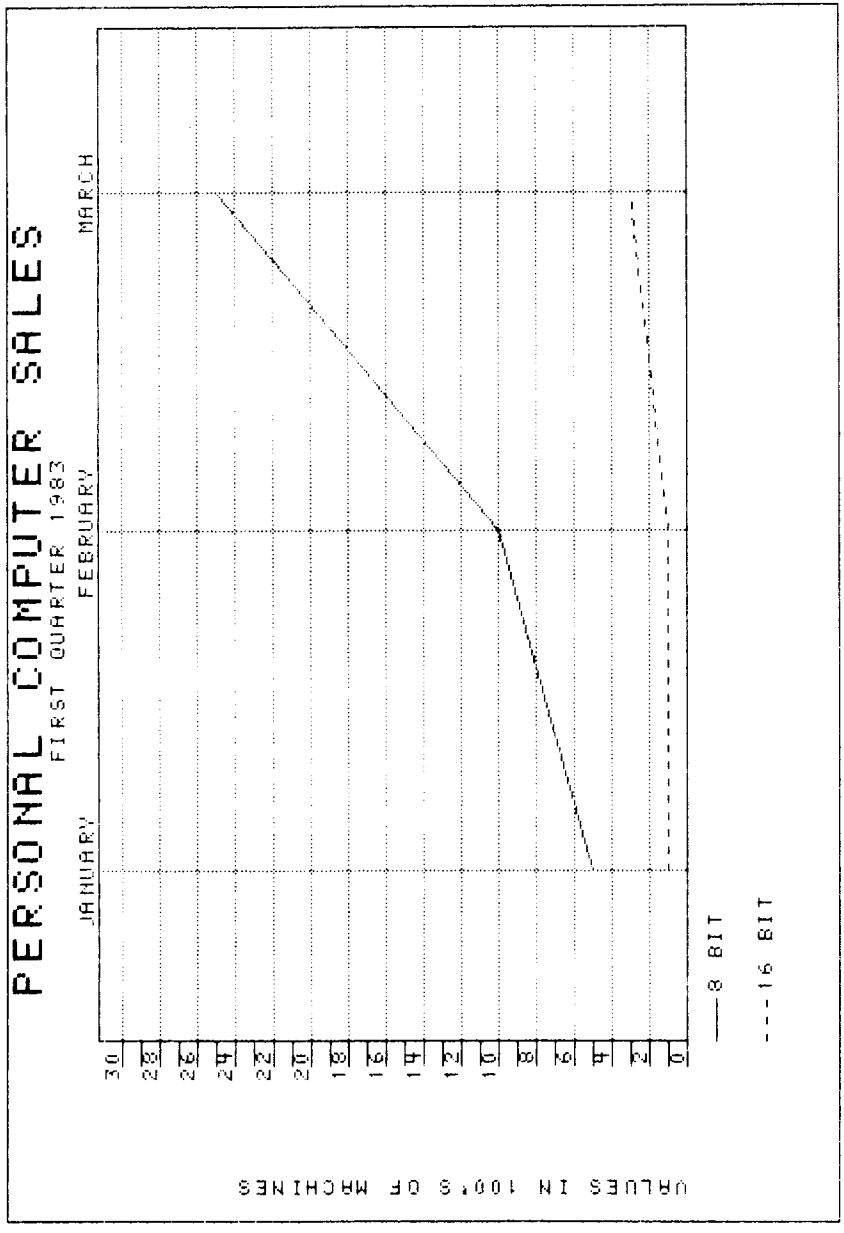


Abb. 22

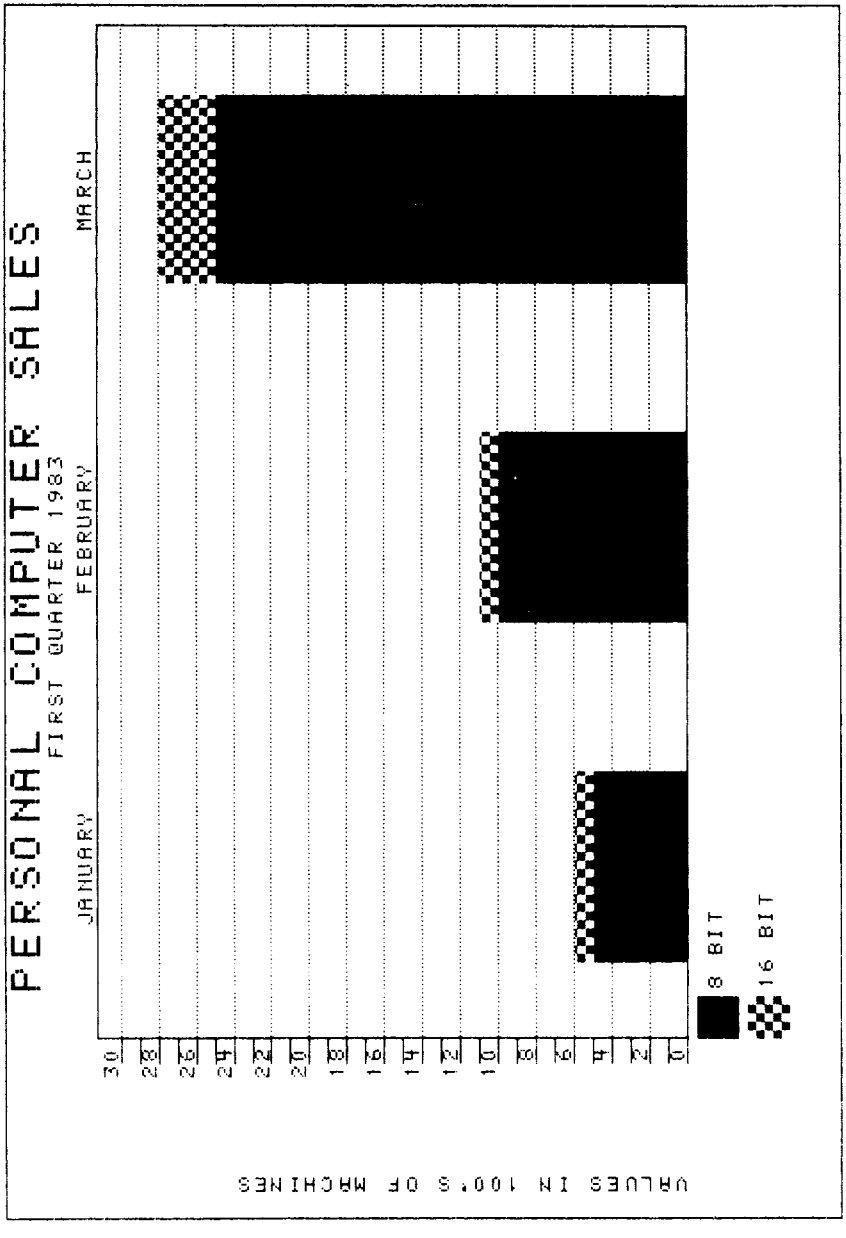
Nach abschließendem Drücken von <CR> erscheint das in Abb. 22 abgebildete Menu wieder. Die Weiterverarbeitung in VEGAS erfolgt nach der Rückkehr zum Hauptmenu.

L I S T O F V A L U E S F O R G R A P H I C - C H A R T

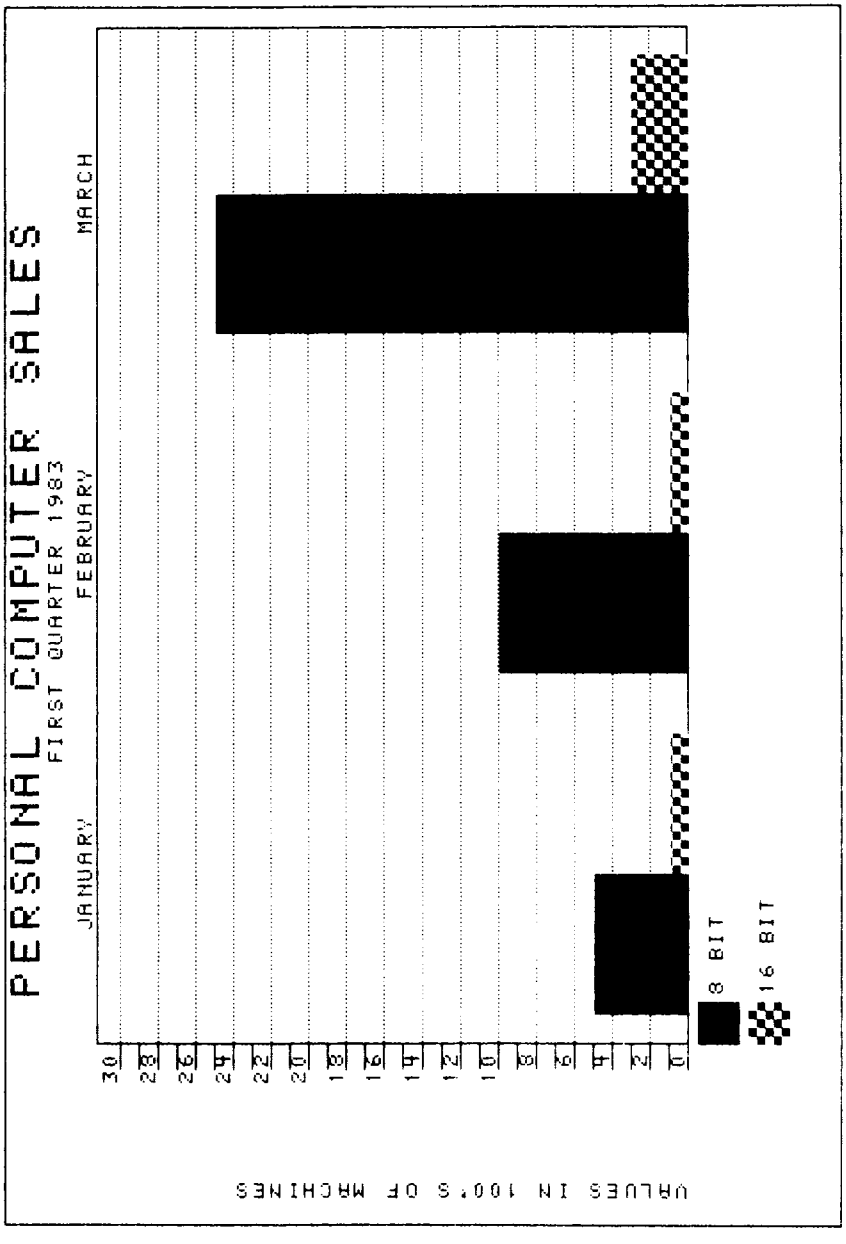
Title	:	PERSONAL COMPUTER SALES	
Second Title	:	FIRST QUARTER 1983	
Value	:	2	
Name of Units	:	MACHINES	
Number of COLUMNS / ROWS	:	3 / 2	
COLUMN / COLUMN-NO	:	ROW-NO / ROW	
	:	1/8 BIT	2/16 BIT
JANUARY 1	:	5	1
FEBRUARY 2	:	10	1
MARCH 3	:	25	3



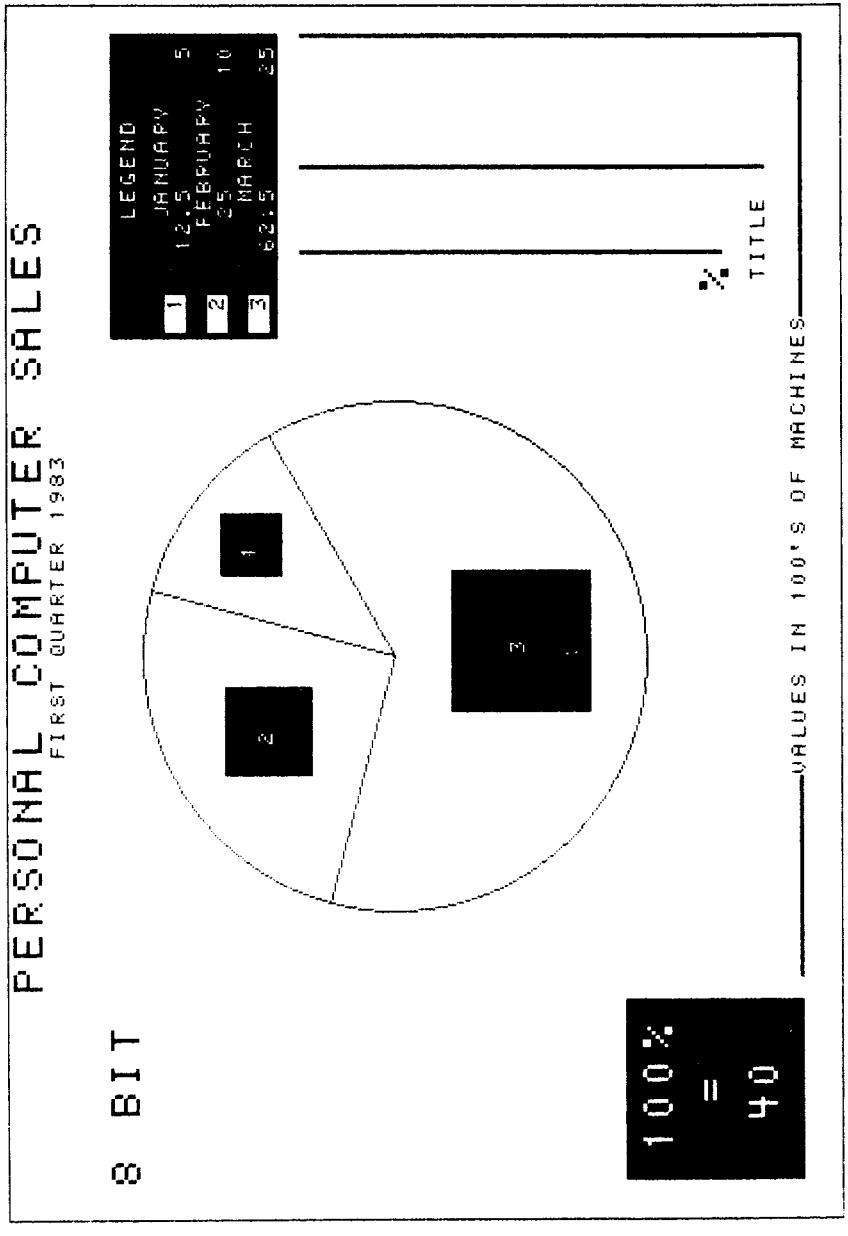
Liniendiagramm



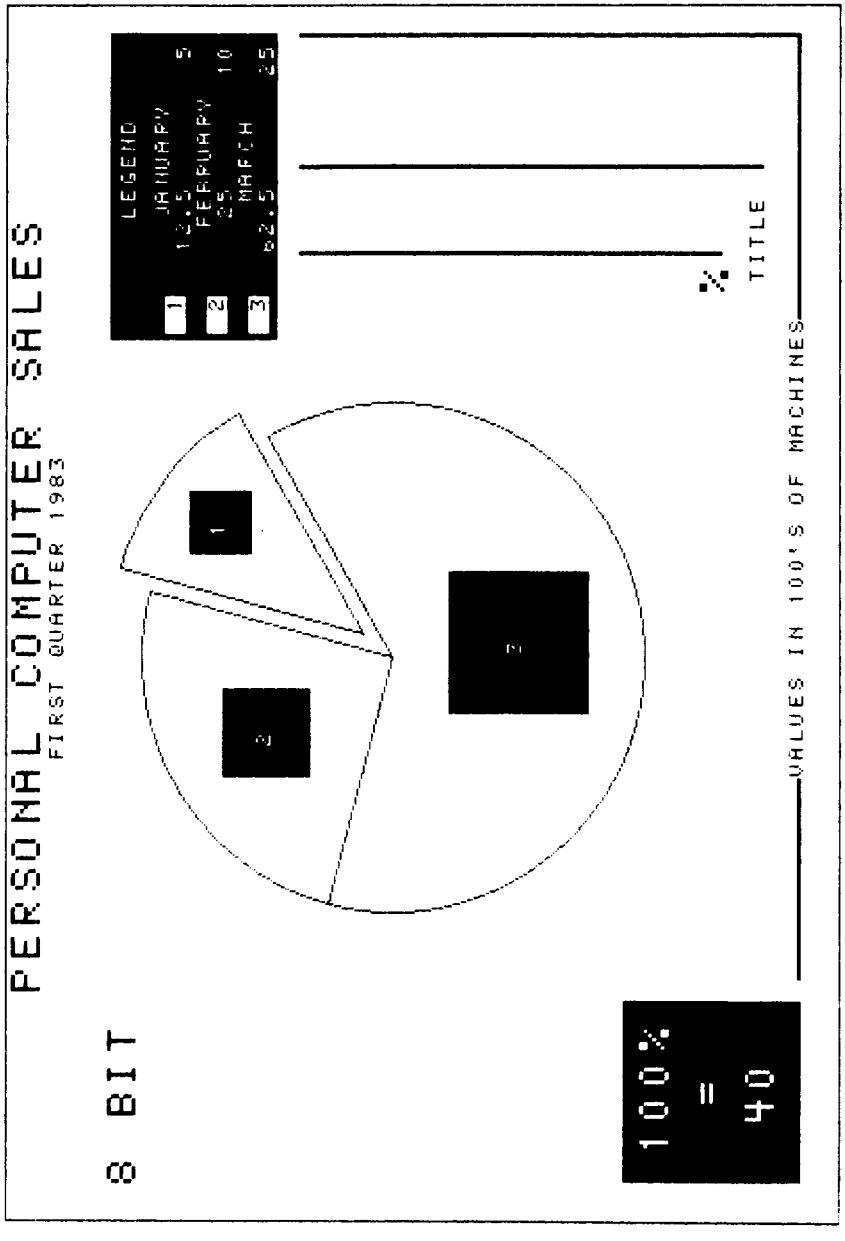
Balkendiagramm (gestapelt)



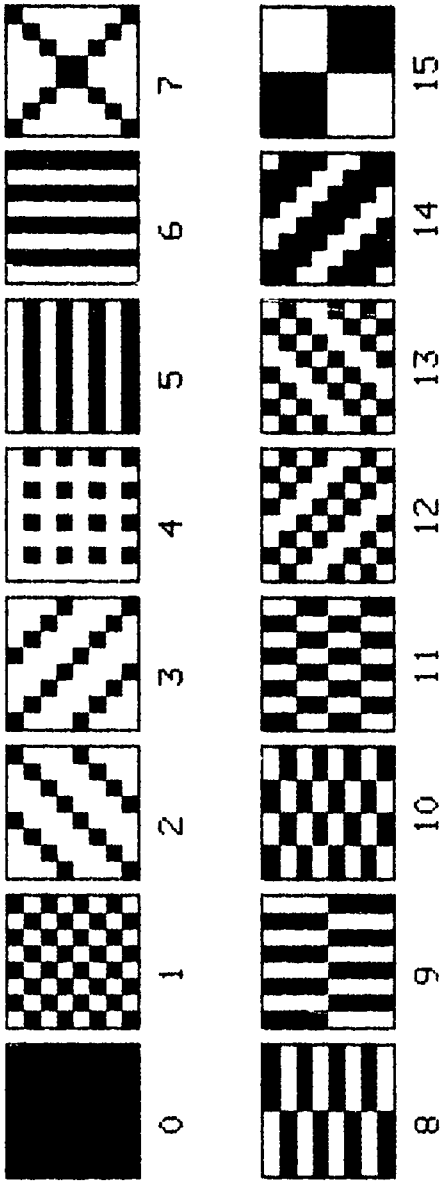
Balkendiagramm (nebeneinander)



Kreisförmiges Diagramm (Kuchendarstellung)



Kreisförmiges Diagramm (mit "Kuchenschnitt")



Druckmuster-Kennzahlen

Fehlermeldungen

Wenn Sie versuchen, eine ungültige Eingabe zu tätigen (z.B., Sie geben Buchstaben in ein Datenfeld ein, wofür eine Zahleneingabe erforderlich ist), wird diese Eingabe ohne ausdrückliche Fehlermeldung zurückgewiesen. VEGAS wartet dann auf eine gültige Eingabe.

Eine Fehlermeldung auf Systemebene kann vorkommen, während Sie versuchen, Daten für ein neues Diagramm auf Diskette zu sichern. VEGAS verfügt über zwei Dateien namens TITLE-F und VALUE-F, in denen sämtliche Daten für Ihre Diagramme gespeichert werden. Wenn kein Platz für eine Erweiterung dieser Dateien vorhanden ist, erscheint folgende Meldung:

Disk Full at address xxxx

Ihr NCR DECISION MATE V kehrt zur Systemebene zurück und zeigt das System-Bereitschaftszeichen an. Sie haben nun drei Möglichkeiten zur Wahl:

1. Löschen der Dateien TITLE-F und VALUE-F, um ihren Disketten-Speicherplatz freizugeben. (Das würde natürlich bedeuten, daß die Daten für sämtliche Diagramme vernichtet werden.)
2. Kopieren der Dateien TITLE-F und VALUE-F auf eine andere Diskette. (Die Daten werden aufbewahrt.)
3. Kopieren sämtlicher VEGAS-Dateien auf eine andere Diskette. Sie können dann diese zweite Diskette für zukünftige VEGAS-Anwendungen benutzen.

Die VEGAS-Dateien erkennen Sie daran, daß ihre ersten zwei Buchstaben jeweils VE sind. und daß sie immer die Namenserweiterung COM aufweisen. Wenn Sie sich für die dritte Möglichkeit entscheiden, müssen Sie ebenfalls die Datei BRUN.COM auf die zweite Diskette kopieren. Das folgende Beispiel zeigt, wie die VEGAS-Dateien vom Laufwerk A auf die Diskette im Laufwerk B kopiert werden können:

```
PIP B: = A:VE*.COM
PIP B: = A:BRUN.COM
```

WICHTIG: Bei der Benutzung dieser Möglichkeiten sollten Sie die Dateien TITLE-F und VALUE-F nicht auf die zweite Diskette kopieren. Diese Dateien werden nämlich auf der zweiten Diskette von VEGAS automatisch neu erstellt.

NCR

NCR DECISION MATE V

MS™-DOS Programmer's Manual

CP/M ist ein eingetragenes Warenzeichen von Digital Research.
MACRO-86, MS-CREF, MS-LINK, MS-LIB und MS-DOS (mit den Programmnamen EDLIN und DEBUG) sind Warenzeichen der Microsoft Corp.
Microsoft ist ein eingetragenes Warenzeichen der Microsoft Corp.

2. Auflage, Oktober 1983

NCR ist ständig bemüht, die Produkte im Zuge der Entwicklung von Technologie, Bauteilen, Soft- und Firmware dem neuesten Stand anzupassen. NCR behält sich deshalb das Recht vor, Spezifikationen ohne vorherige Ankündigung zu ändern.
Nicht alle hier beschriebenen Leistungen werden von NCR in allen Teilen der Welt vertrieben. Nähere Informationen bezüglich eventueller Einschränkungen oder auch Erweiterungen sowie den aktuellen Stand erfahren Sie von Ihrem Händler oder der nächstgelegenen NCR-Geschäftsstelle.

General Introduction

Chapter 1 System Calls

General Introduction

The **Microsoft (R) MS(tm)-DOS Programmer's Reference Manual** is a technical reference manual for system programmers. This manual contains a description and examples of all MS-DOS 2.0 system calls and interrupts (Chapter 1). Chapter 2, "MS-DOS 2.0 Device Drivers" contains information on how to install your own device drivers on MS-DOS. Two examples of device driver programs (one serial and one block) are included in Chapter 2. Chapter 3 through 5 contain technical information about MS-DOS, including MS-DOS disk allocation (Chapter 3), MS-DOS control blocks and work areas (Chapter 4), and EXE file structure and loading (Chapter 5).

Chapter 1
System Calls

1.1 INTRODUCTION

MS-DOS provides two types of system calls: interrupts and function requests. This chapter describes the environments from which these routines can be called, how to call them, and the processing performed by each.

1.2 PROGRAMMING CONSIDERATIONS

The system calls mean you don't have to invent your own ways to perform these primitive functions, and make it easier to write machine-independent programs.

1.2.1 Calling From Macro Assembler

The system calls can be invoked from Macro Assembler simply by moving any required data into registers and issuing an interrupt. Some of the calls destroy registers, so you may have to save registers before using a system call. The system calls can be used in macros and procedures to make your programs more readable; this technique is used to show examples of the calls.

1.2.2 Calling From A High-Level Language

The system calls can be invoked from any high-level language whose modules can be linked with assembly-language modules.

Calling from Microsoft Basic: Different techniques are used to invoke system calls from the compiler and interpreter. Compiled modules can be linked with assembly-language modules; from the interpreter, the CALL statement or USER function can be used to execute the appropriate 8086 object code.

Calling from Microsoft Pascal: In addition to linking with an assembly-language module, Microsoft Pascal includes a function (DOSXQQ) that can be used directly from a Pascal program to call a function request.

Calling from Microsoft FORTRAN: Modules compiled with Microsoft FORTRAN can be linked with assembly-language modules.

1.2.3 Returning Control To MS-DOS

Control can be returned to MS-DOS in any of four ways:

1. Call Function Request 4CH

```
MOV  AH,4CH
INT  21H
```

   This is the preferred method.

2. Call Interrupt 20H:

```
INT  20H
```

3. Jump to location 0 (the beginning of the Program Segment Prefix):

```
JMP  0
```

   Location 0 of the Program Segment Prefix contains an INT 20 H instruction, so this technique is simply one step removed from the first.

4. Call Function Request 00H:

```
MOV  AH,00H
INT  21H
```

   This causes a jump to location 0, so it is simply one step removed from technique 2, or two steps removed from technique 1.

1.2.4 Console And Printer Input/Output Calls

The console and printer system calls let you read from and write to the console device and print on the printer without using any machine-specific codes. You can still take advantage of specific capabilities (display attributes such as positioning the cursor or erasing the screen, printer attributes such as double-strike or underline, etc.) by using constants for these codes and reassembling once with the correct constant values for the attributes.

1.2.5 Disk I/O System Calls

Many of the system calls that perform disk input and output require placing values into or reading values from two system control blocks: the File Control Block (FCB) and directory entry.

1.3 FILE CONTROL BLOCK (FCB)

The Program Segment Prefix includes room for two FCBs at offsets 5CH and 6CH. The system call descriptions refer to unopened and opened FCBs. An **unopened** FCB is one that contains only a drive specifier and filename, which can contain wild card characters (* and ?). An **opened** FCB contains all fields filled by the Open File system call (Function 0FH). Table 1.1 describes the fields of the FCB.

Table 1.1 Fields of File Control Block (FCB)

Name	Size (bytes)	Offset Hex	Offset Decimal
Drive number	1	00H	0
Filename	8	01-08H	1-8
Extension	3	09-0BH	9-11
Current block	2	0CH,0DH	12,13
Record size	2	0EH,0FH	14,15
File size	4	10-13H	16-19
Date of last write	2	14H,15H	20,21
Time of last write	2	16H,17H	22,23
Reserved	8	18-1FH	24-31
Current record	1	20H	32
Relative record	4	21-24H	33-36

1.3.1 Fields Of The FCB

Drive Number (offset 00H): Specifies the disk drive; 1 means drive A: and 2 means drive B:. If the FCB is to be used to create or open a file, this field can be set to 0 to specify the default drive; the Open File system call Function (0FH) sets the field to the number of the default drive.

Filename (offset 01H): Eight characters, left-aligned and padded (if necessary) with blanks. If you specify a reserved device name (such as LPT1), do not put a colon at the end.

Extension (offset 09H): Three characters, left-aligned and padded (if necessary) with blanks. This field can be all blanks (no extension).

Current Block (offset 0CH): Points to the block (group of 128 records) that contains the current record. This field and the Current Record field (offset 20H) make up the record pointer. This field is set to 0 by the Open File system call.

Record Size (offset 0EH): The size of a logical record, in bytes. Set to 128 by the Open File system call. If the record size is not 128 bytes, you must set this field after opening the file.

File Size (offset 10H): The size of the file, in bytes. The first word of this 4-byte field is the low-order part of the size.

Date of Last Write (offset 14H): The date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

```
Offset 15H
| Y | Y | Y | Y | Y | Y | Y | M |
 15                          9 8

Offset 14H
| M | M | M | D | D | D | D | D |
          5 4                 0
```

Time of Last Write (offset 16H): The time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

```
Offset 17H
| H | H | H | H | H | M | M | M |
 15              11 10

Offset 16H
| M | M | M | S | S | S | S | S |
          5 4                 0
```

Reserved (offset 18H): These fields are reserved for use by MS-DOS.

Current Record (offset 20H): Points to one of the 128 records in the current block. This field and the Current Block field (offset 0CH) make up the record pointer. This field is **not** initialized by the Open File system call. You must set it before doing a sequential read or write to the file.

Relative Record (offset 21H): Points to the currently selected record, counting from the beginning of the file (starting with 0). This field is **not** initialized by the Open File system call. You must set it before doing a random read or write to the file. If the record size is less than 64 bytes, both words of this field are used; if the record size ist 64 bytes or more, only the first three bytes are used.

NOTE

If you use the FCB at offset 5CH of the Program Segment Prefix, the last byte of the Relative Record field is the first byte of the unformatted parameter area that starts at offset 80H. This is the default Disk Transfer Address.

1.3.2 Extended FCB

The Extended File Control Block is used to create or search for directory entries of files with special attributes. It adds the following 7-byte prefix to the FCB:

Name	Size (bytes)	Offset (Decimal)
Flag byte (255, or FFH)	1	-7
Reserved	5	-6
Attribute byte: 02H = Hidden file 04H = System file	1	-1

1.3.3 Directory Entry

A directory contains one entry for each file on the disk. Each entry is 32 bytes; Table 1.2 describes the fields of an entry.

Table 1.2 Fields of Directory Entry

Name	Size (bytes)	Offset Hex	Offset Decimal
Filename	8	00-07H	0-7
Extension	3	08-0AH	8-10
Attributes	1	0BH	11
Reserved	10	0C-15H	12-21
Time of last write	2	16H,17H	22,23
Date of last read	2	18H,19H	24,25
Reserved	2	1AH,1BH	26,27
File size	4	1C-1FH	28-31

1.3.4 Fields Of The FCB

Filename (offset 00H): Eight characters, left-aligned and padded (if necessary) with blanks. MS-DOS uses the first byte of this field for two special codes:

00H	(0)	End of allocated directory
E5H	(229)	Free directory entry

Extension (offset 08H): Three characters, left-aligned and padded (if necessary) with blanks. This field can be all blanks (no extension).

Attributes (offset 0BH): Attributes of the file:

Value			
Hex	Binary	Dec	Meaning
01H	0000 0001	1	Read-only
02H	0000 0010	2	Hidden
04H	0000 0100	4	System
07H	0000 0111	7	Changeable with CHGMOD
08H	0000 1000	8	Volume-ID
0AH	0001 0000	10	Directory
16H	0001 0110	22	Hard attributes for FINDENTRY
20H	0010 0000	32	Archive

Reserved (offset 0CH): Reserved for MS-DOS.

Time of Last Write (offset 16H): The time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

Offset 17H

H	H	H	H	H	M	M	M
15				11	10		

Offset 16H

M	M	M	S	S	S	S	S
		5	4				0

Date of Last Write (offset 18H): The date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

Offset 19H

Offset 18H

File Size (offset 1CH): The size of the file, in bytes. The first word of this 4-byte field is the low-order part of the size.

1.4 SYSTEM CALL DESCRIPTIONS

Many system calls require that parameters be loaded into one or more registers before the call is issued; most calls return information in the registers (usually a code that describes the success or failure of the operation). The description of system calls 00H-2EH includes the following:

A drawing of the 8088 registers that shows their contents before and after the system call.

A more complete description of the register contents required before the system call.

A description of the processing performed.

A more complete description of the register contents after the system call.

An example of its use.

The description of system calls 2FH-57H includes the following:

A drawing of the 8088 registers that shows their contents before and after the system call.

A more complete description of the register contents repuired before the system call.

A description of the processing performed.

Error returns from the system call.

An example of its use.

Figure 1 is an example of how each system call is described. Function 27H, Random Block Read, is shown.

Call
AH = 27H
DS:DX
  Opened FCB
CX
  Number of blocks to read

Return
AL
  0 = Read completed successfully
  1 = EOF
  2 = End of segment
  3 = EOF, partial record
CX
  Number of blocks read

Figure 1. Example of System Call Description

1.4.1 Programming Examples

A macro is defined for each system call, then used in some examples. In addition, a few other macros are defined for use in the examples. The use of macros allows the examples to be more complete programs, rather than isolated uses of the system calls. All macro definitions are listed at the end of the chapter.
The examples are not intended to represent good programming practice. In particular, error checking and good human interface design have been sacrificed to conserve space. You may, however, find the macros a convenient way to include system calls in your assembly language programs.
A detailed description of each system call follows. They are listed in numeric order; the interrupts are described first, then the function requests.

NOTE

Unless otherwise stated, all numbers in the system call descriptions - both text and code - are in hex.

1.5 XENIX COMPATIBLE CALLS

MS-DOS 2.0 supports hierarchical (i.e., tree-structured) directories, similar to those found in the Xenix operating system. (For information on tree-structured directories, refer to the **MS-DOS User's Guide.**)

The following system calls are compatible with the Xenix system:

Function 39H	Create Sub-Directory
Function 3AH	Remove a Directory Entry
Function 3BH	Change the Current Directory
Function 3CH	Create a File
Function 3DH	Open a File
Function 3FH	Read From File/Device
Function 40H	Write to a File or Device
Function 41H	Delete a Directory Entry
Function 42H	Move a File Pointer
Function 43H	Change Attributes
Function 44H	I/O Control for Devices
Function 45H	Duplicate a File Handle
Function 46H	Force a Duplicate of a Handle
Function 4BH	Load and Execute a Program
Function 4CH	Terminate a Process
Function 4DH	Retrieve Return Code of a Child

There is no restriction in MS-DOS 2.0 on the depth of a tree (the length of the longest path from root to leaf) except in the number of allocation units available. The root directory will have a fixed number of entries (64 for the single sided disk). For non-root directories, the number of files per directory is only limited by the number of allocation units available.

Pre-2.0 disks will appear to MS-DOS 2.0 as having only a root directory with files in it and no subdirectories.

Implementation of the tree structure is simple. The root directory is the pre-2.0 directory. Subdirectories of the root have a special attribute set indicating that they are directories. The subdirectories themselves are files, linked through the FAT as usual. Their contents are identical in character to the contents of the root directory.

Pre-2.0 programs that use system calls not described in this chapter will be unable to make use of files in other directories. Those files not necessary for the current task will be placed in other directories.

Attributes apply to the tree-structured directories in the following manner:

Attribute	Meaning/Function for files	Meaning/Function for directories
volume-id	Present at the root. Only one file may have this set.	Meaningless.
directory	Meaningless.	Indicates that the directory entry is a directory. Cannot be changed with 43H.
read-only	Old fcb-create, new Create, new open (for write or read/write) will fail.	Meaningless.
archive	Set when file is written. Set/reset via Function 43H.	Meaningless.
hidden/ system	Prevents file from being found in search first/search next. Old open will fail.	Prevents directory entry from being found. Function 3BH will still work.

1.6 INTERRUPTS

MS-DOS reserves interrupts 20H through 3FH for its own use. The table of interrupt routine addresses (vectors) is maintained in locations 80H-FCH. Table 1.3 lists the interrupts in numeric order; Table 1.4 lists the interrupts in alphabetic order (of the description). User programs should only issue Interrupts 20H, 21H, 25H, 26H, and 27H. (Function Requests 4CH and 31H are the preferred method for Interrupts 20H and 27H for versions of MS-DOS that are 2.0 and higher.)

NOTE

Interrupts 22H, 23H, and 24H are not interrupts that can be issued by user programs; they are simply locations where a segment and offset address are stored.

Table 1.3 MS-DOS Interrupts, Numeric Order

Interrupt Hex	Interrupt Dec	Description
20H	32	Program Terminate
21H	33	Function Request
22H	34	Terminate Address
23H	35	<CTRL-C> Exit Address
24H	36	Fatal Error Abort Address
25H	37	Absolute Disk Read
26H	38	Absolute Disk Write
27H	39	Terminate But Stay Resident
28-40H	40-64	RESERVED - DO NOT USE

Table 1.4 MS-DOS Interrupts, Alphabetic Order

Description	Interrupt Hex	Interrupt Dec
Absolute Disk Read	25H	37
Absolute Disk Write	26H	38
<CTRL-C>Exit Address	23H	35
Fatal Error Abort Address	24H	36
Function Request	21H	33
Program Terminate	20H	32
RESERVED - DO NOT USE	28-40H	40-64
Terminate Address	22H	34
Terminate But Stay Resident	27H	39

Program Terminate (Interrupt 20H)

Call
CS
  Segment address of Program Segment Prefix

Return
None

Interrupt 20H causes the current process to terminate and returns control to its parent process. All open file handles are closed and the disk cache is cleaned. This interrupt is almost always used in old .COM files for termination.
The CS register must contain the segment address of the Program Segment Prefix before you call this interrupt.
The following exit addresses are restored from the Program Segment Prefix:

Exit Address	Offset
Program Terminate	0AH
CONTROL-C	0EH
Critical Error	12H

All file buffers are flushed to disk.

NOTE

Close all files that have changed in length before issuing this interrupt. If a changed file is not closed, its length is not recorded correctly in the directory. See Functions 10H and 3EH for a description of the Close File system calls.

Interrupt 20H is provided for compatibility with versions of MS-DOS prior to 2.0. New programs should use Function Request 4CH, Terminate a Process.

```
Macro Definition: terminate macro
                  int 20H
                  endm
```

Example

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
      INT 20H
;There is no return from this interrupt
```

Function Request (Interrupt 21H)

Call
AH
  Function number
Other registers as specified in individual function

Return
As specified in individual function

The AH register must contain the number of the system function. See Section 1.7. "Function Requests", for a description of the MS-DOS system functions.

NOTE

No macro is defined for this interrupt, because all function descriptions in this chapter that define a macro include Interrupt 21H.

Example
To call the Get Time function:

```
mov  ah,2CH     ;Get Time is Function 2CH
int  21H        ;THIS INTERRUPT
```

Terminate Address (Interrupt 22H)
CONTROL-C Exit Address (Interrupt 23H)
Fatal Error Abort Address (Interrupt 24H)

These are not true interrupts, but rather storage locations for a segment and offset address. The interrupts are issued by MS-DOS under the specified circumstance. You can change any of these addresses with Function Request 25H (Set Vector) if you prefer to write your own interrupt handlers.

Interrupt 22H -- Terminate Address
When a program terminates, control transfers to the address at offset 0AH of the Program Segment Prefix. This address is copied into the Program Segment Prefix, from the Interrupt 22H vector, when the segment is created.

Interrupt 23H - CONTROL-C Exit Address
If the user types CONTROL-C during keyboard input or display output, control transfers to the INT 23H vector in the interrupt table. This address is copied into the Program Segment Prefix, from the Interrupt 23H vector, when the segment is created.
If the CONTROL-C routine preserves all registers, it can end with an IRET instruction (return from interrupt) to continue program execution. When the interrupt occurs, all registers are set to the value they had when the original call to MS-DOS was made. There are no restrictions on what a CONTROL-C handler can do - including MS-DOS function calls - so long as the registers are unchanged if IRET is used.
If Function 09H or 0AH (Display String of Buffered Keyboard Input) is interrupted by CONTROL-C, the three-byte sequence 03H-0DH-0AH (ETX-CR-LF) is sent to the display and the function resumes at the beginning of the next line.
If the program creates a new segment and loads a second program that changes the CONTROL-C address, termination of the second program restores the CONTROL-C address to its value before execution of the second program.

Interrupt 24H - Fatal Error Abort Address

If a fatal disk error occurs during execution of one of the disk I/O function calls, control transfers to the INT 24H vector in the vector table. This address is copied into the Program Segment Prefix, from the Interrupt 24H vector, when the segment is created.

BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved.

NOTE

Interrupt 24H is not issued if the failure occurs during execution of Interrupt 25H (Absolute Disk Read) or Interrupt 26H (Absolute Disk Write). These errors are usually handled by the MS-DOS error routine in COMMAND.COM that retries the disk operation, then gives the user the choice of aborting, retrying the operation, or ignoring the error. The following topics give you the information you need about interpreting the error codes, managing the registers and stack, and controlling the system's response to the error in order to write your own error-handling routines.

Error Codes

When an error-handling program gains control from Interrupt 24H, the AX and DI registers can contain codes that describe the error. If Bit 7 of AH is 1, the error is either a bad image of the File Allocation Table or an error occurred on a character device. The device header passed in BP:SI can be examined to determine which case exists. If the attribute byte high order bit indicates a block device, then the error was a bad FAT. Otherwise, the error is on a character device.

The following are error codes for Interrupt 24H:

Error Code	Description
0	Attempt to write on write-protected disk
1	Unknown unit
2	Drive not ready
3	Unknown command
4	Data error
5	Bad request structure length
6	Seek error
7	Unknown media type
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure

The user stack will be in effect (the first item described below is at the top of the stack), and will contain the following from top to bottom:

```
IP        MS-DOS registers from
CS        issuing INT 24H
FLAGS

AX        User registers at time of original
BX        INT 21H request
CX
DX
SI
DI
BP
DS
ES

IP        From the original INT 21H
CS        from the user to MS-DOS
FLAGS
```

The registers are set such that if an IRET is executed, MS-DOS will respond according to (AL) as follows:

```
(AL) = 0   ignore the error
     = 1   retry the operation
     = 2   terminate the program via INT 23H
```

Notes:

1. Before giving this routine control for disk errors, MS-DOS performs five retries.
2. For disk errors, this exit is taken only for errors occurring during an Interrupt 21H. It is not used for errors during Interrupts 25H or 26H.
3. This routine is entered in a disabled state.
4. The SS, SP, DS, ES, BX, CX, and DX registers must be preserved.
5. This interrupt handler should refrain from using MS-DOS function calls. If necessary, it may use calls 01H through 0CH. Use of any other call will destroy the MS-DOS stack and will leave MS-DOS in an unpredictable state.
6. The interrupt handler must not change the contents of the device header.
7. If the interrupt handler will handle errors rather than returning to MS-DOS, it should restore the application program's registers from the stack, remove all but the last three words on the stack, then issue an IRET. This will return to the program immediately after the INT 21H that experienced the error. Note that if this is done, MS-DOS will be in an unstable state until a function call higher than 0CH is issued.

Absolute Disk Read (Interrupt 25H)

Call
AL
  Drive number
DS:BX
  Disk Transfer Address
CX
  Number of sectors
DX
  Beginning relative sector

Return
AL
  Error code if CF = 1
FlagsL
  CF = 0 if successful
     = 1 if not successful

The registers must contain the following:

AL	Drive number (0 = A, 1 = B, etc.).
BX	Offset of Disk Transfer Address (from segment address in DS).
CX	Number of sectors to read.
DX	Beginning relative sector.

This interrupt transfers control to the MS-DOS BIOS. The number of sectors specified in CX is read from the disk to the Disk Transfer Address. Its requirements and processing are identical to Interrupt 26H, except data is read rather than written.

NOTE

All registers except the segment registers are destroyed by this call. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still there upon return. (This is necessary because data is passed back in the flags.) Be sure to pop the stack upon return to prevent uncontrolled growth.

If the disk operation was successful, the Carry Flag (CF) is 0. If the disk operation was not successful, CF is 1 and AL contains the MS-DOS error code (see Interrupt 24H earlier in this section for the codes and their meaning).

Macro Definition:

```
abs-disk-read   macro   disk,buffer,num-sectors,start
                mov     al, disk
                mov     bx,offset buffer
                mov     cx,num-sectors
                mov     dh,start
                int     25H
                endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:. It uses a buffer of 32K bytes:

```
prompt      db      "Source in A, target in B",13,10
            db      "Any Key to start. $"
start       dw      0
buffer      db      64 dup (512 dup (?))   ;64 sectors
            .
            .
int-25H:    display prompt ;see Function 09H
            read-kbd       ;see Function 08H
            mov   cx,5     ;copy 5 groups of
                           ;64 sectors
copy:       push  cx       ;save the loop counter
            abs-disk-read 0,buffer,64,start   ;THIS INTERRUPT
            abs-disk-write 1,buffer,64,start   ;see INT 26H
            add start,64   ;do the next 64 sectors
            pop cx         ;restore the loop counter
            loop copy
```

Absolute Disk Write (Interrupt 26H)

```
Call
AL
  Drive number
DS:BX
  Disk Transfer Address
CX
  Number of sectors
DX
  Beginning relative sector

Return
AL
  Error code if CF = 1
FLAGSL
  CF = 0 if successful
     = 1 if not successful
```

The registers must contain the following:

AL	Drive number (0 = A, 1 = B, etc.).
BX	Offset of Disk Transfer Address (from segment address in DS).
CX	Number of sectors to write.
DX	Beginning relative sector.

This interrupt transfers control to the MS-DOS BIOS. The number of sectors specified in CX is written from the Disk Transfer Address to the disk. Its requirements and processing are identical to Interrupt 25H, except data is written to the disk rather than read from it.

NOTE

All registers except the segment registers are destroyed by this call. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still there upon return. (This is necessary because data is passed back in the flags.) Be sure to pop the stack upon return to prevent uncontrolled growth.

If the disk operation was successful, the Carry Flag (CF) is 0. If the disk operation was not successful, CF is 1 and AL contains the MS-DOS error code (see Interrupt 24H for the codes and their meaning).

Macro Definition:

```
abs-disk-write  macro   disk,buffer,num-sectors,start
                mov     al,disk
                mov     bx,offset buffer
                mov     cx,num-sectors
                mov     dh,start
                int     26H
                endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:, verifying each write. It uses a buffer of 32K bytes:

```
off             equ     0
on              equ     1
                .
                .
prompt          db      "Source in A, target in B",13,10
                db      "Any key to start. $"
start           dw      0
buffer          db      64 dup (512 dup (?))   ;64 sectors
                .
                .
int-26H:        display prompt ;see Function 09H
                read-kbd       ;see Function 08H
                verify on      ;see Function 2EH
                mov  cx,5      ;copy 5 groups of 64 sectors
copy:           push  cx       ;save the loop counter
                abs-disk-read  0,buffer,64,start   ;see INT 25H
                abs-disk-write 1,buffer,64,start   ;THIS INTERRUPT
                add start,64   ;do the next 64 sectors
                pop cx         ;restore the loop counter
                loop copy
                verify off     ;see Function 2EH
```

Terminate But Stay Resident (Interrupt 27H)

Call
CS:DX
  First byte following
  last byte of code

Return
None

The Terminate But Stay Resident call is used to make a piece of code remain resident in the system after its termination. Typically, this call is used in .COM files to allow some device-specific interrupt handler to remain resident to process asynchronous interrupts.
DX must contain the offset (from the segment address in CS) of the first byte following the last byte of code in the program. When Interrupt 27H is executed, the program terminates but is treated as an extension of MS-DOS; it remains resident and is not overlaid by other programs when it terminates.
This interrupt is provided for compatibility with versions of MS-DOS prior to 2.0. New programs should use Function 31H, Keep Process.

Macro Definition:

```
stay-resident   macro   last-instruc
                mov     dx,offset last-instruc
                inc     dx
                int     27H
                endm
```

Example

```
;CS must be equal to PSP values given at program start
; (ES and DS values)
  mov     DX,LastAddress
  int     27H
;There is no return from this interrupt
```

1.7 FUNCTION REQUESTS

Most of the MS-DOS function calls require input to be passed to them in registers. After setting the proper register values, the function may be invoked in one of the following ways:

1. Place the function number in AH and execute a long call to offset 50H in your Program Segment Prefix. Note that programs using this method will not operate correctly on versions of MS-DOS that are lower than 2.0.
2. Place the function number in AH and issue Interrupt 21H. All of the examples in this chapter use this method.
3. An additional method exists for programs that were written with different calling conventions. This method should be avoided for all new programs. The function number is placed in the CL register and other registers are set according to the function specification. Then, an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the MS-DOS function dispatcher. Register AX is always destroyed if this method is used; otherwise, it is the same as normal function calls. Note that this method is valid only for Function Requests 00H through 024H.

1.7.1 CP/M(R)-Compatible Calling Sequence

A different sequence can be used for programs that must conform to CP/M calling conventions:

1. Move any required data into the appropriate registers (just as in the standard sequence).
2. Move the function number into the CL register.
3. Execute an intrasegment call to location 5 in the current code segment.

This method can only be used with functions 00H through 24H that do not pass a parameter in AL. Register AX is always destroyed when a function is called in this manner.

1.7.2 Treatment Of Registers

When MS-DOS takes control after a function call, it switches to an internal stack. Registers not used to return information (except AX) are preserved. The calling program's stack must be large enough to accommodate the interrupt system - at least 128 bytes in addition to other needs.

IMPORTANT NOTE

The macro definitions and extended example for MS-DOS system calls 00H through 2EH can be found at the end of this chapter.

Table 1.5 lists the function requests in numeric order; Table 1.6 lists the function requests in alphabetic order (of the description).

Table 1.5 MS-DOS Function Requests, Numeric Order

Function Number	Function Name
00H	Terminate Program
01H	Read Keyboard and Echo
02H	Display Character
03H	Auxiliary Input
04H	Auxiliary Output
05H	Print Character
06H	Direct Console I/O
07H	Direct Console Input
08H	Read Keyboard
09H	Display String
0AH	Buffered Keyboard Input
0BH	Check Keyboard Status
0CH	Flush Buffer, Read Keyboard
0DH	Disk Reset
0EH	Select Disk
0FH	Open File
10H	Close File
11H	Search for First Entry
12H	Search for Next Entry
13H	Delete File
14H	Sequential Read
15H	Sequential Write

16H	Create File
17H	Rename File
19H	Current Disk
1AH	Set Disk Transfer Address
21H	Random Read
22H	Random Write
23H	File Size
24H	Set Relative Record
25H	Set Vector
27H	Random Block Read
28H	Random Block Write
29H	Parse File Name
2AH	Get Date
2BH	Set Date
2CH	Get Time
2DH	Set Time
2EH	Set/Reset Verify Flag
2FH	Get Disk Transfer Address
30H	Get DOS Version Number
31H	Keep Process
33H	CONTROL-C Check
35H	Get Interrupt Vector
36H	Get Disk Free Space
38H	Return Country-Dependent Info.
39H	Create Sub-Directory
3AH	Remove a Directory Entry
3BH	Change the Current Directory
3CH	Create a File
3DH	Open a File
3EH	Close a File Handle
3FH	Read From File/Device
40H	Write to a File/Device
41H	Delete a Directory Entry
42H	Move a File Pointer
43H	Change Attributes
44H	I/O Control for Devices
45H	Duplicate a File Handle
46H	Force a Duplicate of a Handle
47H	Return Text of Current Directory
48H	Allocate Memory
49H	Free Allocated Memory
4AH	Modify Allocated Memory Blocks
4BH	Load and Execute a Program
4CH	Terminate a Process

4DH	Retrieve the Return Code of a Child
4EH	Find Match File
4FH	Step Through a Directory Matching Files
54H	Return Current Setting of Verify
56H	Move a Directory Entry
57H	Get/Set Date/Time of File

Table 1.6 MS-DOS Function Requests, Alphabetic Order

Function Name	Number
Allocate Memory	48H
Auxiliary Input	03H
Auxiliary Output	04H
Buffered Keyboard Input	0AH
Change Attributes	43H
Change the Current Directory	3BH
Check Keyboard Status	0BH
Close a File Handle	3EH
Close File	10H
CONTROL-C Check	33H
Create a File	3CH
Create File	16H
Create Sub-Directory	39H
Current Disk	19H
Delete a Directory Entry	41H
Delete File	13H
Direct Console Input	07H
Direct Console I/O	06H
Disk Reset	0DH
Display Character	02H
Display String	09H
Duplicate a File Handle	45H
File Size	23H
Find Match File	4EH
Flush Buffer, Read Keyboard	0CH
Force a Duplicate of a Handle	46H
Free Allocated Memory	49H
Get Date	2AH
Get Disk Free Space	36H
Get Disk Transfer Address	2FH
Get DOS Version Number	30H
Get Interrupt Vector	35H

Get Time	2CH
Get/Set Date/Time of File	57H
I/D Control for Devices	44H
Keep Process	31H
Load and Execute a Program	4BH
Modify Allocated Memory Blocks	4AH
Move a Directory Entry	56H
Move a File Pointer	42H
Open a File	3DH
Open File	0FH
Parse File Name	29H
Print Character	05H
Random Block Read	27H
Random Block Write	28H
Random Read	21H
Random Write	22H
Read From File/Device	3FH
Read Keyboard	08H
Read Keyboard and Echo	01H
Remove a Directory Entry	3AH
Rename File	17H
Retrieve the Return Code of a Child	4DH
Return Current Setting of Verify	54H
Return Country-Dependent Info.	38H
Return Text of Current Directory	47H
Search for First Entry	11H
Search for Next Entry	12H
Select Disk	0EH
Sequential Read	14H
Sequential Write	15H
Set Date	2BH
Set Disk Transfer Address	1AH
Set Relative Record	24H
Set Time	2DH
Set Vector	25H
Set/Reset Verify Flag	2EH
Step Through a Directory Matching	4FH
Terminate a Process	4CH
Terminate Program	00H
Write to a File/Device	40H

Terminate Program (Function 00H)

Call
AH = 00H
CS
  Segment address of
  Program Segment Prefix

Return
None

Function 00H is called by Interrupt 20H; it performs the same processing.
The CS register must contain the segment address of the Program Segment Prefix before you call this interrupt.
The following exit addresses are restored from the specified offsets in the Program Segment Prefix:

Program terminate	0AH
CONTROL-C	0EH
Critical error	12H

All file buffers are flushed to disk.

Warning: Close all files that have changed in length before calling this function. If a changed file is not closed, its length is not recorded correctly in the directory. See Function 10H for a description of the Close File system call.

```
Macro Definition:  terminate-program  macro
                                      xor     ah,ah
                                      int     21H
                                      endm
```

Example

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
      mov   ah,0
      int   21H
;There are no returns from this interrupt
```

Read Keyboard and Echo (Function 01H)

Call
AH = 01H

Return
AL
  Character typed

Function 01H waits for a character to be typed at the keyboard, then echoes the character to the display and returns it in AL. If the character is CONTROL-C, Interrupt 23H is executed.

```
Macro Definition:  read-kbd-and-echo  macro
                                      mov     ah, 01H
                                      int     21H
                                      endm
```

Example

The following program both displays and prints characters as they are typed. If RETURN is pressed, the program sends Line Feed-Carriage Return to both the display and the printer:

```
func-01H:  read-kbd-and-echo          ;THIS FUNCTION
           print-char   al            ;see Function 05H
           cmp          al,0DH        ;is it a CR?
           jne          func-01H      ;no, print it
           print-char   10            ;see Function 05H
           display-char 10            ;see Function 02H
           jmp          func-01H      ;get another character
```

Display Character (Function 02H)

Call
AH = 02H
DL
  Character to be displayed

Return
None

Function 02H displays the character in DL. If CONTROL-C is typed, Interrupt 23H is issued.

```
Macro Definition:  display-char  macro   character
                                 mov     dl,character
                                 mov     ah, 02H
                                 int     21H
                                 endm
```

Example

The following program converts lowercase characters to uppercase before displaying them:

```
func-02H:   read-kbd                ;see Function 08H
            cmp     al,"a"
            jl      uppercase       ;don't convert
            cmp     al,"z"
            jg      uppercase       ;don't convert
            sub     al,20H          ;convert to ASCII code
                                    ;for uppercase
uppercase:  display-char al         ;THIS FUNCTION
            jmp     func-02H:       ;get another character
```

Auxiliary Input (Function 03H)

Call
AH = 03H

Return
AL
Character from auxiliary device

Function 03H waits for a character from the auxiliary input device, then returns the character in AL. This system call does not return a status or error code.
If a CONTROL-C has been typed at console input, Interrupt 23H is issued.

```
Macro Definition:  aux-input    macro
                                mov     ah,03H
                                int     21H
                                endm
```

Example

The following program prints characters as they are received from the auxiliary device. It stops printing when an end-of-file character (ASCII 1AH, or CONTROL-Z) is received:

```
func-03H:   aux-input               ;THIS FUNCTION
            cmp     al,1AH          ;end of file?
            je      continue        ;yes, all done
            print-char  al          ;see Function 05H
            jmp     func-03H        ;get another character
continue:   .
```

.

Auxiliary Output (Function 04H)

Call
AH = 04H
DL
  Character for auxiliary device

Return
None

Function 04H sends the character in DL to the auxiliary output device. This system call does not return a status or error code.
If a CONTROL-C has been typed at console input, Interrupt 23H is issued.

```
Macro Definition:  aux-output   macro   character
                                mov     dl,character
                                mov     ah,04H
                                int     21H
                                endm
```

Example

The following program gets a series of strings of up to 80 bytes from the keyboard, sending each to the auxiliary device. It stops when a null string (CR only) is typed:

```
string      db      81 dup(?) ;see Function 0AH
            .
func-04H:   get-string  80,string           ;see Function 0AH
            cmp     string[1],0             ;null string?
            je      continue                ;yes, all done
            mov     cx, word ptr string[1]  ;get string length
            mov     bx,0                    ;set index to 0
send-it:    aux-output string[bx+2]         ;THIS FUNCTION
            inc     bx                      ;bump index
            loop    send-it                 ;send another character
            jmp     func-04H                ;get another string
continue:   .
            .
```

Print Character (Function 05H)

Call
AH = 05H
DL
  Character for printer

Return
None

Function 05H prints the character in DL on the standard printer device. If CONTROL-C has been typed at console input, Interrupt 23H is issued.

```
Macro Definition:  print-char   macro   character
                                mov     dl,character
                                mov     ah,05H
                                int     21H
                                endm
```

Example

The following program prints a walking test pattern on the printer. It stops if CONTROL-C is pressed.

```
line-num    db      0
            .
func-05H:   mov     cx,60           ;print 60 lines
start-line: mov     bl,33           ;first printable ASCII
                                    ;character (!)
            add     bl,line-num     ;to offset ne character
            push    cx              ;save number-of-lines counter
            mov     cx,80           ;loop counter for line
print-it:   print-char bl           ;THIS FUNCTION
            inc     bl              ;move to next ASCII character
            cmp     bl,126          ;last printable ASCII
                                    ;character ( ~)
            jl      no-reset        ;not there yet
            mov     bl,33           ;start over with (!)
```

```
no-reset:   loop    print-it        ;print another character
            print-char 13           ;carriage return
            print-char 10           ;line feed
            inc     line-num        ;to offset 1st char. of line
            pop     cx              ;restore #-of-lines counter
            loop    start-line;     ;print another line
```

Direct Console I/O (Function 06H)

Call
AH = 06H
DL
  See text

Return
AL
  If DL = FFH (255) before call, then Zero flag not set means AL has character from keyboard.
  Zero flag set means there was not a character to get, and AL = 0

The processing depends on the value in DL when the function is called:

DL is FFH (255) - If a character has been typed at the keyboard, it is returned in AL and the Zero flag is 0; if a character has not been typed, the Zero flag is 1.
DL is not FFH - The character in DL is displayed.

This function does **not** check for CONTROL-C.

```
Macro Definition:  dir-console-io macro   switch
                                  mov     dl,switch
                                  mov     ah,06H
                                  int     21H
                                  endm
```

Example

The following program sets the system clock to 0 and continuously displays the time. When any character is typed, the display stops changing; when any character is typed again, the clock is reset to 0 and the display starts again:

```
time        db   "00:00:00.00",13,10,"$"   ;see Function 09H
;                                          ;for explanation of $
ten         db   10
            .
            .
func-06H:   set-time  0,0,0,0              ;see Function 2DH
read-clock: get-time                       ;see Function 2CH
            convert  ch,ten,time           ;see end of chapter
            convert  cl,ten,time[3]        ;see end of chapter
            convert  dh,ten,time[6]        ;see end of chapter
            convert  dl,ten,time[9]        ;see end of chapter
            display  time                  ;see Function 09H
            dir-console-io   FFH           ;THIS FUNCTION
            jne      stop                  ;yes, stop timer
            jmp      read-clock            ;no, keep timer
                                           ;running
stop:       read-kbd                       ;see Function 08H
            jmp      func-06H              ;start over
```

Direct Console Input (Function 07H)

Call
AH = 07H

Return
AL
  Character from keyboard

Function 07H waits for a character to be typed, then returns it in AL. This function does not echo the character or check for CONTROL-C. (For a keyboard input function that echoes or checks for CONTROL-C, see Functions 01H or 08H.)

```
Macro Definition:  dir-console-input  macro
                                      mov     ah,07H
                                      int     21H
                                      endm
```

Example

The following program prompts for a password (8 characters maximum) and places the characters into a string without echoing them:

```
password   db       8 dup(?)
prompt     db       "Password: $"  ;see Function 09H for
                                   ;explanation of $
           .
           .
func-07H:  display prompt          ;see Function 09H
           mov      cx,8           ;maximum length of password
           xor      bx,bx          ;so BL can be used as index
get-pass:  dir-console-input       ;THIS FUNCTION
           cmp      al,0DH         ;was it a CR?
           je       continue       ;yes, all done
           mov      password[bx],al ;no, put character in string
           inc      bx             ;bump index
           loop     get-pass       ;get another character
continue:  .                       ;BX has length of password+1
           .
```

Read Keyboard (Function 08H)

Call
AH = 08H

Return
AL
Character from keyboard

Function 08H waits for a character to be typed, then returns it in AL. If CONTROL-C is pressed, Interrupt 23H is executed. This function does not echo the character. (For a keyboard input function that echoes the character or does not check for CONTROL-C, see Functions 01H or 07H.)

```
Macro Definition:  read-kbd    macro
                               mov     ah,08H
                               int     21H
                               endm
```

Example

The following program prompts for a password (8 characters maximum) and places the characters into a string without echoing them:

```
password   db       8 dup(?)
prompt     db       "Password: $"  ;see Function 09H
                                   ;for explanation of $
           .
           .
func-08H:  display prompt          ;see Function 09H
           mov      cx,8           ;maximum length of password
           xor      bx,bx          ;BL can be an index
get-pass:  read-kbd                ;THIS FUNCTION
           cmp      al,0DH         ;was it a CR?
           je       continue       ;yes, all done
           mov      password[bx],al ;no, put char. in string
           inc      bx             ;bump index
           loop     get-pass       ;get another character
continue:  .                       ;BX has length of password+1
           .
```

Display String (Function 09H)

Call
AH = 09H
DS:DX
  String to be displayed

Return
None

DX must contain the offset (from the segment address in DS) of a string that ends with "$". The string is displayed (the $ is not displayed).

```
Macro Definition:  display  macro   string
                            mov     dx,offset string
                            mov     ah,09H
                            int     21H
                            endm
```

Example

The following program displays the hexadecimal code of the key that is typed:

```
table       db      "0123456789ABCDEF"
sixteen     db      16
result      db      " - 00H",13,10,"$"      ;see text for
                                            ;explanation of $
            .
            .
func-09H:   read-kbd-and-echo               ;see Function 01H
            convert al, sixteen, result[3]  ;see end of chapter
            display result                  ;THIS FUNCTION
            jmp     func-09H                ;do it again
```

Buffered Keyboard Input (Function 0AH)

Call
AH = 0AH
DS:DX
  Input buffer

Return
None

DX must contain the offset (from the segment address in DS) of an input buffer of the following form:

Byte	Contents
1	Maximum number of characters in buffer, including the CR (you must set this value).
2	Actual number of characters typed, not counting the CR (the function sets this value).
3-h	Buffer; must be at least as long as the number in byte 1.

This function waits for characters to be typed. Characters are read from the keyboard and placed in the buffer beginning at the third byte until RETURN is typed. If the buffer fills to one less than the maximum, additional characters typed are ignored and ASCII 7 (BEL) is sent to the display until RETURN is pressed. The string can be edited as it is being entered. If CONTROL-C is typed, Interrupt 23H is issued.
The second byte of the buffer is set to the number of characters entered (not counting the CR).

Macro Definition:

```
get-string   macro   limit,string
             mov     dx,offset string
             mov     string,limit
             mov     ah,0AH
             int     21H
             endm
```

Example

The following program gets a 16-byte (maximum) string from the keyboard and fills a 24-line by 80-character screen with it:

```
buffer          label   byte
max-length      db      ?               ;maximum length
chars-entered   db      ?               ;number of chars.
string          db      17 dup (?)      ;16 chars + CR
strings-per-line dw     0               ;how many strings
                                        ;fit on line
crlf            db      13,10,"$"
                .
                .
func-0AH:       get-string 17,buffer    ;THIS FUNCTION
                xor     bx,bx           ;so byte can be
                                        ;used as index
                mov     bl,chars-entered ;get string length
                mov     buffer[bx+2],"$" ;see Function 09H
                mov     al,50H          ;columns per line
                cbw
                div     chars-entered   ;times string fits
                                        ;on line
                xor     ah,ah           ;clear remainder
                mov     strings-per-line,ax ;save col. counter
                mov     cx,24           ;row counter
display-screen: push    cx              ;save it
                mov     cx, strings-per-line ;get col. counter
display-line:   display string          ;see Function 09H
                loop    display-line
                display crlf            ;see Function 09H
                pop     cx              ;get line counter
                loop    display-screen  ;display 1 more line
```

Check Keyboard Status (Function 0BH)

Call
AH = 0BH

Return
AL
255 (FFH) = characters in type-ahead buffer
0 = no characters in type-ahead buffer

Checks whether there are characters in the type-ahead buffer. If so, AL returns FFH (255); if not, AL returns 0. If CONTROL-C is in the buffer, Interrupt 23H is executed.

```
Macro Definition:  check-kbd-status  macro
                                     mov     ah,0BH
                                     int     21H
                                     endm
```

Example
The following program continuously displays the time until any key is pressed.

```
time        db      "00:00:00.00",13,10,"$"
ten         db      10
            .
            .
func-0BH:   get-time                    ;see Function 2CH
            convert ch,ten,time         ;see end of chapter
            convert cl,ten,time[3]      ;see end of chapter
            convert dh,ten,time[6]      ;see end of chapter
            convert dl,ten,time[9]      ;see end of chapter
            display time                ;see Function 09H
            check-kbd-status            ;THIS FUNCTION
            cmp     al, FFH             ;has a key been typed?
            je      all-done            ;yes, go home
            jmp     func-0BH            ;no, keep displaying
                                        ;time
```

Flush Buffer, Read Keyboard (Function 0CH)

Call
AH = 0CH
AL
1, 6, 7, 8, or 0AH = The corresponding function is called.
Any other value = no further processing.

Return
AL
0 = Type-ahead buffer was flushed; no other
processing performed.

The keyboard type-ahead buffer is emptied. Further processing depends on the value in AL when the function is called:

1, 6, 7, 8, or 0AH - The corresponding MS-DOS function is executed.

Any other value - No further processing; AL returns 0.

```
Macro Definition:  flush-and-read-kbd  macro   switch
                                       mov     al,switch
                                       mov     ah,0CH
                                       int     21H
                                       endm
```

Example
The following program both displays and prints characters as they are typed. If RETURN is pressed, the program sends Carriage Return-Line Feed to both the display and the printer.

```
func-0CH:  flush-and-read-kbd 1      ;THIS FUNCTION
           print-char  al            ;see Function 05H
           cmp         al,0DH        ;is it a CR?
           jne         func-0CH      ;no, print it
           print-char  10            ;see Function 05H
           display-char 10           ;see Function 02H
           jmp         func-0CH      ;get another character
```

Disk Reset (Function 0DH)

Call
AH = 0DH

Return
None

Function 0DH is used to ensure that the internal buffer cache matches the disks in the drives. This function writes out dirty buffers (buffers that have been modified), and marks all buffers in the internal cache as free.
Function 0DH flushes all file buffers. It does not update directory entries; you must close files that have changed to update their directory entries (see Function 10H, Close File). This function need not be called before a disk change if all files that changed were closed. It is generally used to force a known state of the system; CONTROL-C interrupt handlers should call this function.

```
Macro Definition:  disk-reset   macro   disk
                                mov     ah,0DH
                                int     21H
                                endm
```

Example

```
        mov     ah,0DH
        int     21H
;There are no errors returned by this call.
```

Select Disk (Function 0EH)

Call
AH = 0EH
DL
  Drive number
  (0 = A:, 1 = B:, etc.)

Return
AL
  Number of logical drives

The drive specified in DL (0 = A:, 1 = B:, etc.) is selected as the default disk. The number of drives is returned in AL.

```
Macro Definition:  select-disk   macro   disk
                                 mov     dl,disk[-64]
                                 mov     ah, 0EH
                                 int     21H
                                 endm
```

Example
The following program selects the drive not currently selected in a 2-drive system:

```
func-0EH:  current-disk              ;see Function 19H
           cmp      al,00H           ;drive A: selected?
           je       select-b         ;yes, select B
           select-disk "A"           ;THIS FUNCTION
           jmp      continue
select-b:  select-disk "B"           ;THIS FUNCTION
Continue:  .
           .
```

Open File (Function 0FH)

Call
AH = 0FH
DS:DX
  Unopened FCB

Return
AL
  0 = Directory entry found
  255 (FFH) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened File Control Block (FCB). The disk directory is searched for the named file.
If a directory entry for the file is found, AL returns 0 and the FCB is filled as follows:

If the drive code was 0 (default disk), it is changed to the actual disk used (1 = A:, 2 = B:, etc.). This lets you change the default disk without interfering with subsequent operations on this file.
The Current Block field (offset 0CH) is set to zero.
The Record Size (offset 0EH) is set to the system default of 128.
The File Size (offset 10H), Date of Last Write (offset 14H), and Time of Last Write (offset 16H) are set from the directory entry.

Before performing a sequential disk operation on the file, you must set the Current Record field (offset 20H). Before performing a random disk operation on the file, you must set the Relative Record field (offset 21H). If the default record size (128 bytes) is not correct, set it to the correct length.

If a directory entry for the file is not found, AL returns FFH (255).

Macro Definition:

```
open    macro   fcb
        mov     dx,offset fcb
        mov     ah,0FH
        int     21H
        endm
```

Example

The following program prints the file named TEXTFILE.ASC that is on the disk in drive B:. If a partial record is in the buffer at end-of-file, the routine that prints the partial record prints characters until it encounters an end-of-file mark (ASCII 26, or CONTROL-Z):

```
fcb             db      2,"TEXTFILEASC"
                db      25 dup (?)
buffer          db      128 dup (?)
                .
func-0FH:       set-dta buffer          ;see Function 1AH
                open    fcb             ;THIS FUNCTION
read-line:      read-seq fcb            ;see Function 14H
                cmp     al,02H          ;end of file?
                je      all-done        ;yes, go home
                cmp     al,00H          ;more to come?
                jg      check-more      ;no, check for partial
                                        ;record
                mov     cx,128          ;yes, print the buffer
                xor     si,si           ;set index to 0
print-it:       print-char buffer[si]   ;see Function 05H
                inc     si              ;bump index
                loop    print-it        ;print next character
                jmp     read-line       ;read another record
check-more:     cmp     al,03H          ;part. record to print?
                jne     all-done        ;no
                mov     cx,128          ;yes, print it
                xor     si,si           ;set index to 0
find-eof:       cmp     buffer[si],26   ;end-of-file mark?
                je      all-done        ;yes
                print-char buffer[si]   ;see Function 05H
                inc     si              ;bump index to next
                                        ;character
                loop    find-eof
all-done:       close   fcb             ;see Function 10H
```

Close File (Function 10H)

Call
AH = 10 H
DS:DX
  Opened FCB

Return
AL
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (to the segment address in DS) of an opened FCB. The disk directory is searched for the file named in the FCB. This function must be called after a file is changed to update the directory entry.
If a directory entry for the file is found, the location of the file is compared with the corresponding entries in the FCB. The directory entry is updated, if necessary, to match the FCB, and AL returns 0.
If a directory entry for the file is not found, AL returns FFH (255).

```
Macro Definition:  close     macro   fcb
                             mov     dx,offset fcb
                             mov     ah,10H
                             int     21H
                             endm
```

Example
The following program checks the first byte of the file named MOD1.-BAS in drive B: to see if it is FFH, and prints a message if it is:

```
message    db       "Not saved in ASCII format",13,10,"$"
fcb        db       2,"MOD1   BAS"
           db       25 dup (?)
buffer     db       128 dup (?)
           .
           .
func-10H:  set-dta buffer          ;see Function 1AH
           open    fcb             ;see Function 0FH
           read-seq fcb            ;see Function 14H
```

```
            cmp     buffer,FFH      ;is first byte FFH?
            jne     all-done        ;no
            display message         ;see Function 09H
all-done:   close   fcb             ;THIS FUNCTION
```

Search for First Entry (Function 11H)

Call
AH = 11H
DS:DX
  Unopened FCB

Return
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened FCB. The disk directory is searched for the first matching name. The name can have the ? wild card character to match any character. To search for hidden or system files, DX must point to the first byte of the extended FCB prefix.
If a directory entry for the filename in the FCB is found, AL returns 0 and an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.
If a directory entry for the filename in the FCB is not found, AL returns FFH (255).

Notes:
If an extended FCB is used, the following search pattern is used:

1. If the FCB attribute is zero, only normal file entries are found. Entries for volume label, sub-directories, hidden, and system files will not be returned.
2. If the attribute field is set for hidden or system files, or directory entries, it is to be considered as an inclusive search. All normal file entries plus all entries matching the specified attributes are returned. To look at all directory entries except the volume label, the attribute byte may be set to hidden + system + directory (all 3 bits on).

3. If the attribute field is set for the volume label, it is considered an exclusive search, and only the volume label entry is returned.

```
Macro Definition:  search-first  macro   fcb
                                 mov     dx,offset fcb
                                 mov     ah,11H
                                 int     21H
                                 endm
```

Example
The following program verifies the existence of a file named REPORT.ASM on the disk in drive B::

```
yes         db      "FILE EXISTS.$"
no          db      "FILE DOES NOT EXIST.$"
fcb         db      2,"REPORT ASM"
            db      25 dup (?)
buffer      db      128 dup (?)
            .
func-11H:   set-dta buffer          ;see Function 1AH
            search-first fcb        ;THIS FUNCTION
            cmp     al,FFH          ;directory entry found?
            je      not-there       ;no
            display yes             ;see Function 09H
            jmp     continue
not-there:  display no              ;see Function 09H
continue:   display crlf            ;see Function 09H
            .
            .
```

Search for Next Entry (Function 12H)

Call
AH = 12H
DS:DX
  Unopened FCB

Return
AL
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an FCB previously specified in a call to Function 11H. Function 12H is used after Function 11H (Search for First Entry) to find additional directory entries that match a filename that contains wild card characters. The disk directory is searched for the next matching name. The name can have the ? wild card character to match any character. To search for hidden or system files, DX must point to the first byte of the extended FCB prefix.
If a directory entry for the filename in the FCB is found, AL returns 0 and an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.
If a directory entry for the filename in the FCB is not found, AL returns FFH (255).

```
Macro Definition:  search-next  macro   fcb
                                mov     dx,offset fcb
                                mov     ah,12H
                                int     21H
                                endm
```

Example
The following program displays the number of files on the disk in drive B:

```
message   db     "No files",10,13,"$"
files     db     0
ten       db     10
fcb       db     2,"???????????"
          db     25 dup (?)
buffer    db     128 dup (?)
```

```
                .
                .
func-12H:       set-dta buffer              ;see Function 1AH
                search-first fcb            ;see Function 11H
                cmp     al,FFH              ;directory entry found?
                je      all-done            ;no, no files on disk
                inc     files               ;yes, increment file
                                            ;counter
search-dir:     search-next fcb             ;THIS FUNCTION
                cmp     al,FFH              ;directory entry found?
                je      done                ;no
                inc     files               ;yes, increment file
                                            ;counter
                jmp     search-dir          ;check again
done:           convert files,ten,message   ;see end of chapter
all-done:       display message             ;see Function 09H
```

Delete File (Function 13H)

Call
AH = 13H
DS:DX
  Unopened FCB

Return
  0 = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened FCB. The directory is searched for a matching filename. The filename in the FCB can contain the ? wild card character to match any character.
If a matching directory entry is found, it is deleted from the directory. If the ? wild card character is used in the filename, all matching directory entries are deleted. AL returns 0.
If no matching directory entry is found, AL returns FFH (255).

```
Macro Definition:  delete    macro   fcb
                             mov     dx,offset fcb
                             mov     ah,13H
                             int     21H
                             endm
```

Example
The following program deletes each file on the disk in drive B: that was last written before December 31, 1982:

```
year      dw      1982
month     db      12
day       db      31
files     db      0
ten       db      10
message   db      "NO FILES DELETED.",13,10,"$"
                                ;see Function 09H for
                                ;explanation of $
fcb       db      2,"???????????"
          db      25 dup (?)
```

```
buffer      db      128 dup (?)
            .
            .
func-13H:   set-dta buffer          ;see Function 1AH
            search-first fcb        ;see Function 11H
            cmp     al,FFH          ;directory entry found?
            je      all-done        ;no, no files on disk
compare:    convert-date buffer     ;see end of chapter
            cmp     cx,year         ;next several lines
            jg      next            ;check date in directory
            cmp     dl,month        ;entry against date
            jg      next            ;above & check next file
            cmp     dh,day          ;if date in directory
            jge     next            ;entry isn't earlier.
            delete  buffer          ;THIS FUNCTION
            inc     files           ;bump deleted-files
                                    ;counter
next:       search-next fcb         ;see Function 12H
            cmp     al,00H          ;directory entry found?
            je      compare         ;yes, check date
            cmp     files,0         ;any files deleted?
            je      all-done        ;no, display NO FILES
                                    ;message.
            convert files,ten,message ;see end of chapter
all-done:   display message         ;see Function 09H
```

Sequential Read (Function 14H)

Call
AH = 14H
DS:DX
  Opened FCB

Return
Al
  0 = Read completed successfully
  1 = EOF
  2 = DTA too small
  3 = EOF, partial record

DX must contain the offset (from the segment address in DS) of an opened FCB. The record pointed to by the current block (offset 0CH) and Current Record (offset 20H) fields is loaded at the Disk Transfer Address, then the Current Block and Current Record fields are incremented.
The record size is set to the value at offset 0EH in the FCB.
AL returns a code that describes the processing:

Code	Meaning
0	Read completed successfully.
1	End-of-file, no data in the record.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.
3	End-of-file; a partial record was read and padded to the record length with zeros.

```
Macro Definition:  read-seq    macro   fcb
                               mov     dx,offset fcb
                               mov     ah,14H
                               int     21H
                               endm
```

Example
The following program displays the file named TEXTFILE.ASC that is on the disk in drive B:; its function is similar to the MS-DOS TYPE command. If a partial record is in the buffer at end of file, the routine that displays the partial record displays characters until it encounters an end-of-file mark (ASCII 26, or CONTROL-Z):

```
fcb         db      2,"TEXTFILEASC"
            db      25 dup (?)
buffer      db      128 dup (?),"$"
            .
            .
func-14H:   set-dta buffer          ;see Function 1AH
            open    fcb             ;see Function 0FH
read-line:  read-seq fc             ;THIS FUNCTION
            cmp     al,02H          ;end-of-file?
            je      all-done        ;yes
            cmp     al,02H          ;end-of-file with partial
                                    ;record?
            jg      check-more      ;yes
            display buffer          ;see Function 09H
            jmp     read-line       ;get another record
check-more: cmp     al,03H          ;partial record in buffer?
            jne     all-done        ;no, go home
            xor     si,si           ;set index to 0
find-eof:   cmp     buffer[si],26   ;is character EOF?
            je      all-done        ;yes, no more to display
            display-char buffer[si] ;see Function 02H
            inc     si              ;bump index to next
                                    ;character
            jmp     find-eof        ;check next character
all-done    close   fcb             ;see Function 10H
```

Sequential Write (Function 15H)

Call
AH = 15H
DS:DX
  Opened FCB

Return
AL
  00H = Write completed successfully
  01H = Disk full
  02H = DTA too small

DX must contain the offset (from the segment address in DS) of an opened FCB. The record pointed to by Current Block (offset 0CH) and Current Record (offset 20H) fields is written from the Disk Transfer Address, then the current block and current record fields are incremented.
The record size is set to the Value at offset 0EH in the FCB. If the Record Size is less than a sector, the data at the Disk Transfer Address is written to a buffer; the buffer is written to disk when it contains a full sector of data, or the file is closed, or a Reset Disk system call (Function 0DH) is issued.
AL returns a code that describes the processing:

Code	Meaning
0	Transfer completed successfully.
1	Disk full; write canceled.
2	Not enough room at the Disk Transfer Address to write one record; write canceled

Macro Definition:

```
write-seq   macro   fcb
            mov     dx,offset fcb
            mov     ah,15H
            int     21H
            endm
```

Example

The following program creates a file named DIR.TMP on the disk in drive B: that contains the disk number (0 = A:, 1 = B:, etc.) and filename from each directory entry on the disk:

```
record-size  equ         14              ;offset of Record Size
                                         ;field in FCB
             .
             .
fcb1         db          2,"DIR  TMP"
             db          25 dup (?)
fcb2         db          2,"???????????"
             db          25 dup (?)
buffer       db          128 dup (?)
             .
             .
func-15H:    set-dta     buffer          ;see Function 1AH
             search-first fcb2           ;see Function 11H
             cmp         al,FFH          ;directory entry found?
             je          all-done        ;no, no files on disk
             create      fcb1            ;see Function 16H
             mov         fcb1[record-size],12
                                         ;set record size to 12
write-it:    write-seq   fcb1            ;THIS FUNCTION
             search-next fcb2            ;see Function 12H
             cmp         al,FFH          ;directory entry found?
             je          all-done        ;no, go home
             jmp         write-it        ;yes, write the record
all-done:    close       fcb1            ;see Function 10H
```

Create File (Function 16H)

Call
AH = 16H
DS:DX
  Unopened FCB

Return
AL
  00H = Empty directory found
  FFH (255) = No empty directory
              available

DX must contain the offset (from the segment address in DS) of an unopened FCB. The directory is searched for an empty entry or an existing entry for the specified filename.
If an empty directory entry is found, it is initialized to a zero-length file, the Open File system call (Function 0FH) is called, and AL returns 0. You can create a hidden file by using an extended FCB with the attribute byte (offset FCB-1) set to 2.
If an entry is found for the specified filename, all data in the file is released, making a zero-length file, and the Open File system call (Function 0FH) is issued for the filename (in other words, if you try to create a file that already exists, the existing file is erased, and a new, empty file is created).
If an empty directory entry is not found and there is no entry for the specified filename, AL returns FFH (255).

```
Macro Definition:  create    macro   fcb
                             mov     dx,offset fcb
                             mov     ah,16H
                             int     21H
                             endm
```

Example
The following program creates a file named DIR.TMP on the disk in drive B: that contains the disk number (0 = A:, 1 = B:, etc.) and filename from each directory entry on the disk:

```
record-size  equ     14              ;offset of Record Size
                                     ;field of FCB
             .
             .
fcb1         db      2,"DIR  TMP"
             db      25 dup (?)
fcb2         db      2,"???????????"
             db      25 dup (?)
buffer       db      128 dup (?)
             .
             .
func-16H:    set-dta buffer          ;see Function 1AH
             search-first fcb2       ;see Function 11H
             cmp     al,FFH          ;directory entry found?
             je      all-done        ;no, no files on disk
             create  fcb1            ;THIS FUNCTION
             mov     fcb1[record-size],12
                                     ;set record size to 12
write-it:    write-seq fcb1          ;see Function 15H
             search-next  fcb2       ;see Function 12H
             cmp     al,FFH          ;directory entry found?
             je      all-done        ;no, go home
             jmp     write-it        ;yes, write the record
all-done:    close   fcb1            ;see Function 10H
```

Rename File (Function 17H)

Call
AH = 17H
DS:DX
  Modified FCB

Return
AL
  00H = Directory entry found
  EFH (255) = No directory entry
  found or destination already exists

DX must contain the offset (from the segment address in DS) of an FCB with the drive number and filename filled in, followed by a second filename at offset 11H. The disk directory is searched for an entry that matches the first filename, which can contain the ? wild card character.
If a matching directory entry is found, the filename in the directory entry is changed to match the second filename in the modified FCB (the two filenames cannot be the same name). If the ? wild card character is used in the second filename, the corresponding characters in the filename of the directory entry are not changed. AL returns 0.
If a matching directory entry is not found or an entry is found for the second filename, AL returns FFH (255).

```
Macro Definition:  rename    macro   fcb,newname
                             mov     dx,offset fcb
                             mov     ah,17H
                             int     21H
                             endm
```

Example
The following program prompts for the name of a file and a new name, then renames the file:

```
fcb         db   37 dup (?)
prompt1     db   "Filename: $"
prompt2     db   "New name: $"
reply       db   17 dup(?)
crlf        db   13,10,"$"
                 .
                 .
```

```
func-17H:  display   prompt1          ;see Function 09H
           get-string 15,reply        ;see Function 0AH
           display   crlf             ;see Function 09H
           parse     reply[2],fcb     ;see Function 29H
           display   prompt2          ;see Function 09H
           get-string 15,reply        ;see Function 0AH
           display   crlf             ;see Function 09 H
           parse     reply[2],fcb[16]
                                      ;see Function 29H
           rename    fcb              ;THIS FUNCTION
```

Current Disk (Function 19H)

Call
AH = 19H

Return
AL
Currently selected drive
(0 = A, 1 = B, etc.)

AL returns the currently selected drive (0 = A:, 1 = B:, etc.).

```
Macro Definition:  current-disk  macro
                                 mov     ah,19H
                                 int     21H
                                 endm
```

Example
The following program displays the currently selected (default) drive in a 2-drive system:

```
message     db "Current disk is $"   ;see Function 09H
                                     ;for explanation of $
crlf        db      13,10,"$"
            .
            .
func-19H:   display  message         ;see Function 09H
            current-disk             ;THIS FUNCTION
            cmp      al,00H          ;is it disk A?
            jne      disk-b          ;no, it's disk B:
            display-char "A"         ;see Function 02H
            jmp      all-done
disk-b:     display-char "B"         ;see Function 02H
all-done:   display  crlf            ;see Function 09H
```

Set Disk Transfer Address (Function 1AH)

Call
AH = 1AH
DS:DX
  Disk Transfer Address

Return
None

DX must contain the offset (from the segment address in DS) of the Disk Transfer Address. Disk transfers cannot wrap around from the end of the segment to the beginning, nor can they overflow into another segment.

NOTE

If you do not set the Disk Transfer Address, MS-DOS defaults to offset 80H in the Program Segment Prefix.

```
Macro Definition:  set-dta    macro   buffer
                              mov     dx,offset buffer
                              mov     ah,1AH
                              int     21H
                              endm
```

Example
The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. The file contains 26 records; each record is 28 bytes long:

```
record-size      equ    14          ;offset of Record Size
                                    ;field of FCB
relative-record  equ    33          ;offset of Relative Record
                                    ;field of FCB
                 .
                 .
```

```
fcb         db      2,"ALPHABETDAT"
            db      25 dup (?)
buffer      db      34 dup (?),"$"
prompt      db      "Enter letter: $"
crlf        db      13,10,"$"
            .
            .
func-1AH:   set-dta buffer              ;THIS FUNCTION
            open    fcb                 ;see Function 0FH
            mov     fcb[record-size],28 ;set record size
get-char:   display prompt              ;see Function 09H
            read-kbd-and-echo           ;see Function 01H
            cmp     al,0DH              ;just a CR?
            je      all-done            ;yes, go home
            sub     al,41H              ;convert ASCII
                                        ;code to record #
            mov     fcb[relative-record],al
                                        ;set relative record
            display crlf                ;see Function 09H
            read-ran fcb                ;see Function 21H
            display buffer              ;see Function 09H
            display crlf                ;see Function 09H
            jmp     get-char            ;get another character
all-done:   close   fcb                 ;see Function 10H
```

Random Read (Function 21H)

Call
AH = 21H
DS:DX
  Opened FCB

Return
AL
  00H = Read completed successfully
  01H = EOF
  02H = DTA too small
  03H = EOF, partial record

DX must contain the offset (from the segment address in DS) of an opened FCB. The Current Block (offset 0CH) and Current Record (offset 20H) fields are set to agree with the Relative Record field (offset 21H), then the record addressed by these fields is loaded at the Disk Transfer Address.
AL returns a code that describes the processing:

Code	Meaning
0	Read completed successfully.
1	End-of-file; no data in the record.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.
3	End-of-file; a partial record was read and padded to the record length with zeros.

```
Macro Definition:  read-ran  macro   fcb
                             mov     dx,offset fcb
                             mov     ah,21H
                             int     21H
                             endm
```

Example

The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. The file contains 26 records; each record is 28 bytes long:

```
record-size     equ     14              ;offset of Record Size
                                        ;field of FCB
relative-record equ     33              ;offset of Relative Record
                                        ;field of FCB
                .
                .
fcb             db      2,"ALPHABETDAT"
                db      25 dup (?)
buffer          db      34 dup (?),"$"
prompt          db      "Enter letter: $"
crlf            db      13,10,"$"
                .
                .
func-21H:       set-dta buffer          ;see Function 1AH
                open    fcb             ;see Function 0FH
                mov     fcb[record-size],28 ;set record size
get-char:       display prompt          ;see Function 09H
                read-kbd-and-echo       ;see Function 01H
                cmp     al,0DH          ;just a CR?
                je      all-done        ;yes, go home
                sub     al,41H          ;convert ASCII code
                                        ;to record #
                mov     fcb[relative-record],al ;set relative
                                        ;record
                display crlf            ;see Function 09H
                read-ran fcb            ;THIS FUNCTION
                display buffer          ;see Function 09H
                display crlf            ;see Function 09H
                jmp     get-char        ;get another char.
all-done:       close   fcb             ;see Function 10H
```

Random Write (Function 22H)

Call
AH = 22H
DS:DX
  Opened FCB

Return
AL
  00H = Write completed successfully
  01H = Disk full
  02H = DTA too small

DX must contain the offset from the segment address in DS of an opened FCB. The Current Block (offset 0CH) and Current Record (offset 20H) fields are set to agree with the Relative Record field (offset 21H), then the record addressed by these fields is written from the Disk Transfer Address. If the record size is smaller than a sector (512 bytes), the records are buffered until a sector is ready to write. AL returns a code that describes the processing:

Code	Meaning
0	Write completed successfully.
1	Disk is full.
2	Not enough room at the Disk Transfer Address to write one record; write canceled.

```
Macro Definition:  write-ran   macro   fcb
                               mov     dx,offset fcb
                               mov     ah,22H
                               int     21H
                               endm
```

Example

The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in drive B:. After displaying the record, it prompts the user to enter a changed record. If the user types a new record, it is written to the file; if the user just presses RETURN, the record is not replaced. The file contains 26 records; each record is 28 bytes long:

```
record-size     equ     14              ;offset of Record Size
                                        ;field of FCB
relative-record equ     33              ;offset of Relative Record
                                        ;field of FCB
                .
                .
fcb             db      2,"ALPHABETDAT"
                db      25 dup (?)
buffer          db      26 dup (?),13,10,"$"
prompt1         db      "Enter letter: $"
prompt2         db      "New record (RETURN for no change): $"
crlf            db      13,10,"$"
reply           db      28 dup (32)
blanks          db      26 dup (32)
                .
                .
func-22H:       set-dta buffer          ;see Function 1AH
                open    fcb             ;see Function 0FH
                mov     fcb[record-size],32 ;set record size
get-char:       display prompt1         ;see Function 09H
                read-kbd-and-echo       ;see Function 01H
                cmp     al,0DH          ;just a CR?
                je      all-done        ;yes, go home
                sub     al,41H          ;convert ASCII
                                        ;code to record #
                mov     fcb[relative-record],al
                                        ;set relative record
                display crlf            ;see Function 09H
                read-ran fcb            ;THIS FUNCTION
                display buffer          ;see Function 09H
                display crlf            ;see Function 09H
                display prompt2         ;see Function 09H
                get-string 27,reply     ;see Function 0AH
                display crlf            ;see Function 09H
                cmp     reply[1],0      ;was anything typed
                                        ;besides CR?
                je      get-char        ;no
                                        ;get another char.
                xor     bx,bx           ;to load a byte
                mov     bl,reply[1]     ;use reply length as
                                        ;counter
                move-string blanks,buffer,26 ;see chapter end
                move-string reply[2],buffer,bx ;see chapter end
                write-ran fcb           ;THIS FUNCTION
                jmp     get-char        ;get another character
all-done:       close   fcb             ;see Function 10H
```

File Size (Function 23H)

Call
AH = 23H
DS:DX
  Unopened FCB

Return
AL
  00H = Directory entry found
  FFH (255) = No directory entry found

DX must contain the offset (from the segment address in DS) of an unopened FCB. You must set the Record Size field (offset 0EH) to the proper value before calling this function. The disk directory is searched for the first matching entry.
If a matching directory entry is found, the Relative Record field (offset 21H) is set to the number of records in the file, calculated from the total file size in the directory entry (offset 1CH) and the Record Size field of the FCB (offset 0EH). AL returns 00.
If no matching directory is found, AL returns FFH (255).

NOTE

If the value of the Record Size field of the FCB (offset 0EH) doesn't match the actual number of characters in a record, this function does not return the correct file size. If the default record size (128) is not correct, you must set the Record Size field to the correct value before using this function.

```
Macro Definition:  file-size    macro   fcb
                                mov     dx,offset fcb
                                mov     ah,23H
                                int     21H
                                endm
```

Example

The following program prompts for the name of a file, opens the file to fill in the Record Size field of the FCB, issues a File Size system call, and displays the file size and number of records in hexadecimal:

```
fcb         db      37 dup (?)
prompt      db      "File name: $"
msg1        db      "Record length:  ",13,10,"$"
msg2        db      "Records:  ",13,10,"$"
crlf        db      13,10,"$"
reply       db      17 dup (?)
sixteen     db      16
            .
            .
func-23H:   display prompt          ;see Function 09H
            get-string 17,reply     ;see Function 0AH
            cmp     reply[1],0      ;just a CR?
            jne     get-length      ;no, keep going
            jmp     all-done        ;yes, go home
get-length: display crlf            ;see Function 09H
            parse   reply[2],fcb    ;see Function 29H
            open    fcb             ;see Function 0FH
            file-size fcb           ;THIS FUNCTION
            mov     si,33           ;offset to Relative
                                    ;Record field
            mov     di,9            ;reply in msg-2
convert-it: cmp     fcb[si],0       ;digit to convert?
            je      show-it         ;no, prepare message
            convert fcb[si],sixteen,msg-2[di]
            inc     si              ;bump n-o-r index
            inc     di              ;bump message index
            jmp     convert-it      ;check for a digit
show-it:    convert fcb[14],sixteen,msg-1[15]
            display msg-1           ;see Function 09H
            display msg-2           ;see Function 09H
            jmp     func-23H        ;get a filename
all-done:   close   fcb             ;see Function 10H
```

Set Relative Record (Function 24H)

Call
AH = 24H
DS:DX
  Opened FCB

Return
None

DX must contain the offset (from the segment address in DS) of an opened FCB. The Relative Record field (offset 21H) is set to the same file address as the Current Block (offset 0CH) and Current Record (offset 20H) fields.

```
Macro Definition:  set-relative-record  macro   fcb
                                        mov     dx,offset fcb
                                        mov     ah,24H
                                        int     21H
                                        endm
```

Example

The following program copies a file using the Random Block Read and Random Block Write system calls. It speeds the copy by setting the record length equal to the file size and the record count to 1, and using a buffer of 32K bytes. It positions the file pointer by setting the Current Record field (offset 20H) to 1 and using Set Relative Record to make the Relative Record field (offset 21H) point to the same record as the combination of the Current Block (offset 0CH) and Current Record (offset 20H) fields:

```
current-record  equ     32              ;offset of Current Record
                                        ;field of FCB
file-size       equ     16              ;offset of File Size
                                        ;field of FCB
                .
                .
fcb             db      37 dup (?)
filename        db      17 dup (?)
prompt1         db      "File to copy: $"   ;see Function 09H for
prompt2         db      "Name of copy: $"   ;explanation of $
crlf            db      13,10,"$"
```

```
file-length  dw      ?
buffer       db      32767 dup (?)
             .
             .
func-24H:    set-dta   buffer          ;see Function 1AH
             display   prompt1         ;see Function 09H
             get-string 15, filename   ;see Function 0AH
             display   crlf            ;see Function 09H
             parse     filename[2],fcb ;see Function 29H
             open      fcb             ;see Function 0FH
             mov       fcb[current-record],0 ;set Current Record
                                       ;field
             set-relative-record fcb   ;THIS FUNCTION
             mov       ax,word ptr fcb[file-size] ;get file size
             mov       file-length,ax  ;save it for
                                       ;ran-block-write
             ran-block-read fcb,1,ax   ;see Function 27H
             display   prompt2         ;see Function 09H
             get-string 15,filename    ;see Function 0AH
             display   crlf            ;see Function 09H
             parse     filename[2],fcb ;see Function 29H
             create    fcb             ;see Function 16H
             mov       fcb[current-record],0 ;set Current Record
                                       ;field
             set-relative-record fcb   ;THIS FUNCTION
             mov       ax,file-length  ;get original file
                                       ;length
             ran-block-write fcb,1,ax  ;see Function 28H
             close     fcb             ;see Function 10H
```

Set Vector (Function 25H)

Call
AH = 25H
AL
  Interrupt number
DS:DX
  Interrupt-handling routine

Return
None

Function 25H should be used to set a particular interrupt vector. The operating system can then manage the interrupts on a per-process basis. Note that programs should **never** set interrupt vectors by writing them directly in the low memory vector table.
DX must contain the offset (to the segment address in DS) of an interrupt-handling routine. AL must contain the number of the interrupt handled by the routine. The address in the vector table for the specified interrupt is set to DS:DX.

```
Macro Definition:  set-vector  macro   interrupt,seg-addr,off-addr
                               mov     al,interrupt
                               push    ds
                               mov     ax,seg-addr
                               mov     ds,ax
                               mov     dx,off-addr
                               mov     ah,25H
                               int     21H
                               pop     ds
                               endm
```

Example

```
lds     dx,intvector
mov     ah,25H
mov     al,intnumber
int     21H
;There are no errors returned
```

Random Block Read (Function 27H)

Call
AH = 27H
DS:DX
  Opened FCB
CX
  Number of blocks to read

Return
AL
  00H = Read completed successfully
  01H = EOF
  02H = End of segment
  03H = EOF, partial record
CX
  Number of blocks read

DX must contain the offset (to the segment address in DS) of an opened FCB. CX must contain the number of records to read; if it contains 0, the function returns without reading any records (no operation). The specified number of records - calculated from the Record Size field (offset 0EH) - is read starting at the record specified by the Relative Record field (offset 21H). The records are placed at the Disk Transfer Address.
AL returns a code that describes the processing:

Code	Meaning
0	Read completed successfully.
1	End-of-file; no data in the record.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.
3	End-of-file; a partial record was read and padded to the record length with zeros.

CX returns the number of records read; the Current Block (offset 0CH), Current Record (offset 20H), and Relative Record (offset 21H) fields are set to address the next record.

```
Macro Definition: ran-block-read  macro   fcb,count,rec-size
                                  mov     dx,offset fcb
                                  mov     cx,count
                                  mov     word ptr fcb[14],rec-size
                                  mov     ah,27H
                                  int     21H
                                  endm
```

Example

The following program copies a file using the Random Block Read system call. It speeds the copy by specifying a record count of 1 and a record length equal to the file size, and using a buffer of 32 K bytes; the file is read as a single record (compare to the sample program for Function 28H that specifies a record **length** of 1 and a record **count** equal to the file size):

```
current-record  equ  32  ;offset of Current Record field
file-size       equ  16  ;offset of File Size field

                .
                .
fcb             db       37 dup (?)
filename        db       17 dup(?)
prompt1         db       "File to copy: $"    ;see Function 09H for
prompt2         db       "Name of copy: $"    ;explanation of $
crlf            db       13,10,"$"
file-length     dw       ?
buffer          db       32767 dup(?)
                .
                .
func-27H:       set-dta  buffer               ;see Function 1AH
                display  prompt1              ;see Function 09H
                get-string15,filename         ;see Function 0AH
                display  crlf                 ;see Function 09H
                parse    filename[2],fcb      ;see Function 29H
                open     fcb                  ;see Function 0FH
                mov      fcb[current-record],0  ;set Current
                                              ;Record field
                set-relative-record fcb       ;see Function 24H
                mov      ax,word ptr fcb[file-size]
                                              ;get file size
                mov      file-length,ax       ;save it for
                                              ;ran-block-write
                ran-block-read   fcb,1,ax     ;THIS FUNCTION
```

```
display   prompt2                ;see Function 09H
get-string15,filename            ;see Function 0AH
display   crlf                   ;see Function 09H
parse     filename[2],fcb        ;see Function 29H
create    fcb                    ;see Function 16H
mov       fcb[current-record],0
                                 ;set Current Record
                                 ;field
set-relative-record fcb          ;see Function 24H
mov       ax, file-length        ;get original file
                                 ;size
ran-block-write   fcb,1,ax       ;see Function 28H
close     fcb                    ;see Function 10H
```

Random Block Write (Function 28H)

Call
AH = 28H
DS:DX
  Opened FCB
CX
  Number of blocks to write
  (0 = set File Size field)

Return
AL
  00H = Write completed successfully
  01H = Disk full
  02H = End of segment
CX
  Number of blocks written

DX must contain the offset (to the segment address in DS) of an opened FCB; CX must contain either the number of records to write or 0. The specified number of records (calculated from the Record Size field, offset 0EH) is written from the Disk Transfer Address. The records are written to the file starting at the record specified in the Relative Record field (offset 21H) of the FCB. If CX is 0, no records are written, but the File Size field of the directory entry (offset 1CH) is set to the number of records specified by the Relative Record field of the FCB (offset 21H); allocation units are allocated or released, as required.
AL returns a code that describes the processing:

Code	Meaning
0	Write completed successfully.
1	Disk full. No records written.
2	Not enough room at the Disk Transfer Address to read one record; read canceled.

CX returns the number of records written; the current block (offset 0CH), Current Record (offset 20H), and Relative Record (offset 21H) fields are set to address the next record.

```
Macro Definition:  ran-block-write  macro   fcb,count,rec-size
                                    mov     dx,offset fcb
                                    mov     cx,count
                                    mov     word ptr fcb[14],
                                            rec-size
                                    mov     ah,28H
                                    int     21H
                                    endm
```

Example

The following program copies a file using the Random Block Read and Random Block Write system calls. It speeds the copy by specifying a record count equal to the file size and a record length of 1, and using a buffer of 32K bytes; the file is copied quickly with one disk access each to read and write (compare to the sample program of Function 27H, that specifies a record **count** of 1 and a record **length** equal to file size):

```
current-record  equ     32              ;offset of Current Record field
file-size       equ     16              ;offset of File Size field
                .
                .
fcb             db      37 dup (?)
filename        db      17 dup(?)
prompt1         db      "File to copy: $"   ;see Function 09H for
prompt2         db      "Name of copy: $"   ;explanation of $
crlf            db      13,10,"$"
num-recs        dw      ?
buffer          db      32767 dup(?)
                .
                .
func-28H:       set-dta     buffer          ;see Function 1AH
                display     prompt1         ;see Function 09H
                get-string  15, filename    ;see Function 0AH
                display     crlf            ;see Function 09H
                parse       filename[2],fcb ;see Function 29H
                open        fcb             ;see Function 0FH
                mov         fcb[current-record],0
                                            ;set Current Record
                                            ;field
                set-relative-record fcb     ;see Function 24H
                mov         ax, word ptr fcb[file-size]
                                            ;get file size
```

```
mov       num-recs,ax          ;save it for
                               ;ran-block-write
ran-block-read fcb,num-recs,1 ;THIS FUNCTION
display   prompt2              ;see Function 09H
get-string15,filename          ;see Function 0AH
display   crlf                 ;see Function 09H
parse     filename[2],fcb      ;see Function 29H
create    fcb                  ;see Function 16H
mov       fcb[current-record],0 ;set Current
                               ;Record field
set-relative-record fcb        ;see Function 24H
mov       ax, file-length      ;get size of original
ran-block-write fcb,num-recs,1 ;see Function 28H
close     fcb                  ;see Function 10H
```

Parse File Name (Function 29H)

Call
AH = 29H
AL
  Controls parsing (see text)
DS:SI
  String to parse
ES:DI
  Unopened FCB

Return
AL
  00H = No wild card characters
  01H = Wild-card characters used
  FFH (255) = Drive letter invalid
DS:SI
  First byte past string that was parsed
ES:DI
  Unopened FCB

SI must contain the offset (to the segment address in DS) of a string (command line) to parse; DI must contain the offset (to the segment address in ES) of an unopened FCB. The string is parsed for a filename of the form d:filename.ext; if one is found, a corresponding unopened FCB is created at ES:DI.
Bits 0-3 of AL control the parsing and processing. Bits 4-7 are ignored:

Bit	Value	Meaning
0	0	All parsing stops if a file separator is encountered.
	1	Leading separators are ignored.
1	0	The drive number in the FCB is set to 0 (default drive) if the string does not contain a drive number.
	1	The drive number in the FCB is not changed if the string does not contain a drive number.
2	1	The filename in the FCB is not changed if the string does not contain a filename.
	0	The filename in the FCB is set to 8 blanks if the string does not contain a filename.
3	1	The extension in the FCB is not changed if the string does not contain an extension.
	0	The extension in the FCB is set to 3 blanks if the string does not contain an extension.

If the filename or extension includes an asterisk (*), all remaining characters in the name or extension are set to question mark (?).

Filename separators:

: . ; , = + / " [ ] \ < > | space tab

Filename terminators include all the filename separators plus any control character. A filename cannot contain a filename terminator; if one is encountered, parsing stops.

If the string contains a valid filename:

1. AL returns 1 if the filename or extension contains a wild card character (* or ?); AL returns 0 if neither the filename nor extension contains a wild card character.
2. DS:SI point to the first character following the string that was parsed.
   ES:DI point to the first byte of the unopened FCB.

If the drive letter is invalid, AL returns FFH (255). If the string does not contain a valid filename, ES:DI+1 points to a blank (ASCII 20H).

Macro Definition: parse

```
macro   string,fcb
mov     si,offset string
mov     di,offset fcb
push    es
push    ds
pop     es
mov     al,0FH ;bits 0, 1, 2, 3 on
mov     ah,29H
int     21H
pop     es
endm
```

Example

The following program verifies the existence of the file named in reply to the prompt:

```
fcb     db      37 dup (?)
prompt  db      "Filename: $"
reply   db      17 dup(?)
yes     db      "FILE EXISTS",13,10,"$"
```

```
no          db      "FILE DOES NOT EXIST",13,10,"$"
            .
            .
func-29H:   display prompt          ;see Function 09H
            get-string15,reply      ;see Function 0AH
            parse   reply[2],fcb    ;THIS FUNCTION
            search-first fcb        ;see Function 11H
            cmp     al,FFH          ;dir. entry found?
            je      not-there       ;no
            display yes             ;see Function 09H
            jmp     continue
not-there:  display no
continue:   .
            .
```

Get Date (Function 2AH)

Call
AH = 2AH

Return
CX
  Year (1980 - 2099)
DH
  Month (1 - 12)
DL
  Day (1 - 31)
AL
  Day of week (0=Sun., 6=Sat.)

This function returns the current date set in the operating system as binary numbers in CX and DX:

CX Year (1980-2099)
DH Month (1 = January, 2 = February, etc.)
DL Day (1-31)
AL Day of week (0 = Sunday, 1 = Monday, etc.)

```
Macro Definition:  get-date    macro
                               mov     ah,2AH
                               int     21H
                               endm
```

Example

The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date:

```
month       db      31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func-2AH:   get-date                ;see above
            inc     dl              ;increment day
            xor     bx,bx           ;so BL can be used as index
            mov     bl,dh           ;move month to index register
            dec     bx              ;month table starts with 0
            cmp     dl,month[bx]    ;past end of month?
            jle     month-ok        ;no, set the new date
            mov     dl,1            ;yes, set day to 1
```

```
            inc     dh          ;and increment month
            cmp     dh,12       ;past end of year?

            jle     month-ok    ;no, set the new date
            mov     dh,1        ;yes, set the month to 1
            inc     cx          ;increment year
month-ok:   set-date cx,dh,dl   ;THIS FUNCTION
```

Set Date (Function 2BH)

Call
AH = 2BH
CX
  Year (1980 - 2099)
DH
  Month (1 - 12)
DL
  Day (1 - 31)

Return
AL
  00H = Date was valid
  FFH (255) = Date was invalid

Registers CX and DX must contain a valid date in binary:

CX Year (1980-2099)
DH Month (1 = January, 2 = February, etc.)
DL Day (1-31)

If the date is valid, the date is set and AL returns 0. If the date is not valid, the function is canceled and AL returns FFH (255).

```
Macro Definition:  set-date    macro   year,month,day
                               mov     cx,year
                               mov     dh,month
                               mov     dl,day
                               mov     ah,2BH
                               int     21H
                               endm
```

Example

The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date:

```
month       db      31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func-2BH:   get-date                ;see Function 2AH
            inc     dl              ;increment day
            xor     bx,bx           ;so BL can be used as index
```

```
            mov     bl,dh           ;move month to index register
            dec     bx              ;month table starts with 0
            cmp     dl,month[bx]    ;past end of month?
            jle     month-ok        ;no, set the new date
            mov     dl,1            ;yes, set day to 1
            inc     dh              ;and increment month
            cmp     dh,12           ;past end of year?
            jle     month-ok        ;no, set the new date
            mov     dh,1            ;yes, set the month to 1
            inc     cx              ;increment year
month-ok:   set-date cx,dh,dl       ;THIS FUNCTION
```

Get Time (Function 2CH)

Call
AH = 2CH

Return
CH
  Hour (0 - 23)
CL
  Minutes (0 - 59)
DH
  Seconds (0 - 59)
DL
  Hundredths (0 - 99)

This function returns the current time set in the operating system as binary numbers in CX and DX:

CH Hour (0-23)
CL Minutes (0-59)
DH Seconds (0-59)
DL Hundredths of a second (0-99)

Macro Definition: get-time

```
macro
mov     ah,2CH
int     21H
endm
```

Example

The following program continuously displays the time until any key is pressed:

```
time        db      "00:00:00.00",13,10,"$"
ten         db      10
            .
            .
func-2CH:   get-time                ;THIS FUNCTION
            convert ch,ten,time     ;see end of chapter
            convert cl,ten,time[3]  ;see end of chapter
            convert dh,ten,time[6]  ;see end of chapter
            convert dl,ten,time[9]  ;see end of chapter
            display time            ;see Function 09H
            check-kbd-status        ;see Function 0BH
            cmp     al,FFH          ;has a key been pressed?
            je      all-done        ;yes, terminate
            jmp     func-2CH        ;no, display time
```

Set Time (Function 2DH)

```
Call
AH = 2DH
CH
  Hour (0 - 23)
CL
  Minutes (0 - 59)
DH
  Seconds (0 - 59)
DL
  Hundredths (0 - 99)

Return
AL
  00H = Time was valid
  FFH (255) = Time was invalid
```

Registers CX and DX must contain a valid time in binary:

CH Hour (0-23)
CL Minutes (0-59)
DH Seconds (0-59)
DL Hundredths of a second (0-99)

If the time is valid, the time is set and AL returns 0. If the time is not valid, the function is canceled and AL returns FFH (255).

```
Macro Definition: set-time macro  hour,minutes,seconds,hundredths
                  mov     ch,hour
                  mov     cl,minutes
                  mov     dh,seconds
                  mov     dl,hundredths
                  mov     ah,2DH
                  int     21H
                  endm
```

Example

The following program sets the system clock to 0 and continuously displays the time. When a character is typed, the display freezes; when another character is typed, the clock is reset to 0 and the display starts again:

```
time        db      "00:00:00.00",13,10,"$"
ten         db      10
            .
            .
func-2DH:   set-time 0,0,0,0          ;THIS FUNCTION
read-clock: get-time                  ;see Function 2CH
            convert ch,ten,time       ;see end of chapter
            convert cl,ten,time[3]    ;see end of chapter
            convert dh,ten,time[6]    ;see end of chapter
            convert dl,ten,time[9]    ;see end of chapter
            display time              ;see Function 09H
            dir-console-io FFH        ;see Function 06H
            cmp     al,00H            ;was a char. typed?
            jne     stop              ;yes, stop the timer
            jmp     read-clock        ;no keep timer on
stop:       read-kbd                  ;see Function 08H
            jmp     func-2DH          ;keep displaying time
```

Set/Reset Verify Flag (Function 2EH)

Call
AH = 2EH
AL
  00H = Do not verify
  01H = Verify

Return
None

AL must be either 1 (verify after each disk write) or 0 (write without verifying). MS-DOS checks this flag each time it writes to a disk. The flag is normally off; you may wish to turn it on when writing critical data to disk. Because disk errors are rare and verification slows writing, you will probably want to leave it off at other times.

```
Macro Definition:  verify    macro   switch
                             mov     al,switch
                             mov     ah,2EH
                             int     21H
                             endm
```

Example

The following program copies the contents of a single-sided disk in drive A: to the disk in drive B:, verifying each write. It uses a buffer of 32K bytes:

```
on          equ     1
off         equ     0
            .
            .
prompt      db      "Source in A, target in B",13,10
            db      "Any key to start. $"
start       dw      0
buffer      db      64 dup (512 dup(?)) ;64 sectors
            .
            .
func-2DH:   display prompt          ;see Function 09H
            read-kbd                ;see Function 08H
            verify on               ;THIS FUNCTION
            mov     cx,5            ;coby 64 sectors
                                    ;5 times
```

```
copy:       push    cx                  ;save counter
            abs-disk-read 0,buffer,64,start
                                        ;see Interrupt 25H
            abs-disk-write 1,buffer,64,start
                                        ;see Interrupt 26H
            add     start,64            ;do next 64 sectors
            pop     cx                  ;restore counter
            loop    copy                ;do it again
            verify  off                 ;THIS FUNCTION

disk-read 0,buffer,64,start             ;see Interrupt 25H
            abs-disk-write 1,buffer,64,start
                                        ;see Interrupt 26H
            add     start,64            ;do next 64 sectors
            pop     cx                  ;restore counter
            loop    copy                ;do it again
            verify  off
```

Get Disk Transfer Address (Function 2FH)

Call
AH = 2FH

Return
ES:BX
Points to Disk Transfer Address

Function 2FH returns the DMA transfer address.

Error returns:
None.

Example

```
mov     ah,2FH
int     21H
        ;es:bx has current DMA transfer address
```

Get DOS Version Number (Function 30H)

Call
AH = 30H

Return
AL
  Major version number
AH
  Minor version number

This function returns the MS-DOS version number. On return, AL.AH will be the two-part version designation; i.e., for MS-DOS 1.28, AL would be 1 and AH would be 28. For pre-1.28, DOS AL = 0. Note that version 1.1 is the same as 1.10, not the same as 1.01.

Error returns:
None.

Example

```
mov     ah,30
int     21H
        ; al is the major version number
        ; ah is the minor version number
        ; bh is the OEM number
        ; bl:cx is the (24 bit) user number
```

Keep Process (Function 31H)

Call
AH = 31H
AL
  Exit code
DX
  Memory size, in paragraphs

Return
None

This call terminates the current process and attempts to set the initial allocation block to a specific size in paragraphs. It will not free up any other allocation blocks belonging to that process. The exit code passed in AX is retrievable by the parent via Function 4DH.
This method is preferred over Interrupt 27H and has the advantage of allowing more than 64K to be kept.

Error returns:
None.

Example

```
mov     al, exitcode
mov     dx, parasize
mov     ah, 31H
int     21H
```

CONTROL-C Check (Function 33H)

Call
AH = 33H
AL
  Function
  00H =Request current state
  01H = Set state
DL (if setting)
  00H = Off
  01H = On

Return
DL
  00H = Off
  01H = On

MS-DOS ordinarily checks for a CONTROL-C on the controlling device only when doing function call operations 01H-0CH to that device. Function 33H allows the user to expand this checking to include any system call. For example, with the CONTROL-C trapping off, all disk I/O will proceed without interruption; with CONTROL-C trapping on, the CONTROL-C interrupt is given at the system call that initiates the disk operation.

NOTE

Programs that wish to use calls 06H or 07H to read CONTROL-Cs as data must ensure that the CONTROL-C check is off.

Error return:
AL = FF
  The function passed in AL was not in the range 0:1.

Example

```
mov     dl,val
mov     ah,33H
mov     al,func
```

```
int         21H
            ; If al was 0, then dl has the current value
            ;of the CONTROL-C check
```

Get Interrupt Vector (Function 35H)

Call
AH = 35H
AL
  Interrupt number

Return
ES:BX
  Pointer to interrupt routine

This function returns the interrupt vector associated with an interrupt. Note that programs should **never** get an interrupt vector by reading the low memory vector table directly.

Error returns:
None.

Example

```
mov     ah,35H
mov     al,interrupt
int     21H
   ; es:bx now has long pointer to interrupt routine
```

Get Disk Free Space (Function 36H)

Call
AH = 36H
DL
  Drive (0 = Default,
  1 = A, etc.)

Return
BX
  Available clusters
DX
  Clusters per drive
CX
  Bytes per sector
AX
  FFFF if drive number is invalid;
  otherwise sectors per cluster

This function returns free space on disk along with additional information about the disk.

Error returns:
AX = FFFF
  The drive number given in DL was invalid.

Example

```
mov     ah,36H
mov     dl,Drive        ;0 = default, A = 1
int     21H
   ; bx = Number of free allocation units on drive
   ; dx = Total number of allocation units on drive
   ; cx = Bytes per sector
   ; ax = Sectors per allocation unit
```

Return Country-Dependent Information (Function 38H)

Call
AH = 38H
DS:DX
  Pointer to 32-byte memory area
AL
  Function code. In MS-DOS 2.0, must be 0

Return
Carry set:
AX
  2 = file not found
Carry not set:
  DX:DS filled in with country data

The value passed in AL is either 0 (for current country) or a country code. Country codes are typically the international telephone prefix code for the country.
If DX = -1, then the call sets the current country (as returned by the AL = 0 call) to the country code in AL. If the country code is not found, the current country is not changed.

NOTE

Applications must assume 32 bytes of information. This means the buffer pointed to by DS:DX must be able to accommodate 32 bytes.

This function is fully supported only in versions of MS-DOS 2.01 and higher. It exists in MS-DOS 2.0, but is not fully implemented.
This function returns, in the block of memory pointed to by DS:DX, the following information pertinent to international applications:

WORD Date/time format
5 BYTE ASCIZ string currency symbol
2 BYTE ASCIZ string thousands separator
2 BYTE ASCIZ string decimal separator
2 BYTE ASCIZ string date separator
2 BYTE ASCIZ string time separator
1 BYTE Bit field
1 BYTE Currency places
1 BYTE time format
DWORD Case Mapping call
2 BYTE ASCIZ string data list separator

The format of most of these entries is ASCIZ (a NUL terminated ASCII string), but a fixed size is allocated for each field for easy indexing into the table.
The date/time format has the following values:

0 - USA standard	h:m:s m/d/y
1 - Europe standard	h:m:s d/m/y
2 - Japan standard	y/m/d h:m:s

The bit field contains 8 bit values. Any bit not currently defined must be assumed to have a random value.

Bit 0 = 0 If currency symbol precedes the currency amount.
= 1 If currency symbol comes after the currency amount.
Bit 1 = 0 If the currency symbol immediately precedes the currency amount.
= 1 If there is a space between the currency symbol and the amount.

The time format has the following values:

0 - 12 hour time
1 - 24 hour time

The currency places field indicates the number of places which appear after the decimal point on currency amounts.
The Case Mapping call is a FAR procedure which will perform country specific lower-to-uppercase mapping on character values from 80H to FFH. It is called with the character to be mapped in AL. It returns the correct upper case code for that character, if any, in AL. AL and the FLAGS are the only registers altered. It is allowable to pass this routine codes below 80H; however nothing is done to characters in this range. In the case where there is no mapping, AL is not altered.

Error returns:
AX
2 = file not found
The country passed in AL was not found (no table for specified country).

Example

```
lds     dx, blk
mov     ah, 38H
mov     al, Country-code
int     21H
   ;AX = Country code of country returned
```

Create Sub-Directory (Function 39H)

Call
AH = 39H
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  3 = path not found
  5 = access denied
Carry not set:
  No error

Given a pointer to an ASCIZ name, this function creates a new directory entry at the end.

Error returns:
AX
  3 = path not found
      The path specified was invalid or not found.
  5 = access denied
      The directory could not be created (no room in parent directory), the directory/file already existed or a device name was specified.

Example

```
lds     dx, name
mov     ah, 39H
int     21H
```

Remove a Directory Entry (Function 3AH)

Call
AH = 3AH
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  3 = path not found
  5 = access denied
  16 = current directory
Carry not set:
  No error

Function 3AH is given an ASCIZ name of a directory. That directory is removed from its parent directory.

Error returns:
AX
3 = path not found
    The path specified was invalid or not found.
5 = access denied
    The path specified was not empty, not a directory, the root directory, or contained invalid information.
16 = current directory
    The path specified was the current directory on a drive.

Example

```
lds     dx, name
mov     ah, 3AH
int     21H
```

Change the Current Directory (Function 3BH)

Call
AH = 3BH
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  3 = path not found
Carry not set:
  No error

Function 3BH is given the ASCIZ name of the directory which is to become the current directory. If any member of the specified pathname does not exist, then the current directory is unchanged. Otherwise, the current directory is set to the string.

Error returns:
AX
  3 = path not found
    The path specified in DS:DX either indicated a file or the path was invalid.

Example

```
lds     dx, name
mov     ah, 3BH
int     21H
```

Create a File (Function 3CH)

Call
AH = 3CH
DS:DX
  Pointer to pathname
CX
  File attribute

Return
Carry set:
AX
  5 = access denied
  3 = path not found
  4 = too many open files
Carry not set:
  AX is handle number

Function 3CH creates a new file or truncates an old file to zero length in preparation for writing. If the file did not exist, then the file is created in the appropriate directory and the file is given the attribute found in CX. The file handle returned has been opened for read/write access.

Error returns:
AX
5 = access denied
    The attributes specified in CX contained one that could not be created (directory, volume ID), a file already existed with a more inclusive set of attributes, a directory existed with the same name, or the path was not found.
3 = path not found
    The path specified had a syntax error.
4 = too many open files
    The file was created with the specified attributes, but there were no free handles available for the process, or the internal system tables were full.

Example

```
lds     dx, name
mov     ah, 3CH
mov     cx, attribute
int     21H
  ; ax now has the handle
```

Open a File (Function 3DH)

Call
AH = 3DH
AL
  Access
  0 = File opened for reading
  1 = File opened for writing
  2 = File opened for both
  reading and writing

Return
Carry set:
AX
  12 = invalid access
  2 = file not found
  5 = access denied
  4 = too many open files
Carry not set:
  AX is handle number

Function 3DH associates a 16-bit file handle with a file. The following values are allowed:

ACCESS	Function
0	file is opened for reading
1	file is opened for writing
2	file is opened for both reading and writing.

DS:DX point to an ASCIZ name of the file to be opened.

The read/write pointer is set at the first byte of the file and the record size of the file is 1 byte. The returned file handle must be used for subsequent I/O to the file.

Error returns:

AX

12 = invalid access

The access specified in AL was not in the range 0:2.

2 = file not found

The path specified was invalid or not found.

5 = access denied

The user attempted to open a directory or volume-id, or open a read-only file for writing.

4 = too many open files

There were no free handles available in the current process or the internal system tables were full.

Example

```
lds     dx, name
mov     ah, 3DH
mov     al, access
int     21H
  ; ax has error or file handle
  ; If successful open
```

Close a File Handle (Function 3EH)

Call
AH = 3EH
BX
  File handle

Return
Carry set:
AX
  6 = invalid handle
Carry not set:
  No error

In BX is passed a file handle (like that returned by Functions 3DH, 3CH, or 45H), Function 3EH closes the associated file. Internal buffers are flushed.

Error return:
AX
  6 = invalid handle
      The handle passed in BX was not currently open.

Example

```
mov     bx, handle
mov     ah, 3EH
int     21H
```

Call
AH = 3FH
DS:DX
  Pointer to buffer
CX
  Bytes to read
BX
  File handle

Return
Carry set:
AX
  Number of bytes read
  6 = invalid handle
  5 = error set:
Carry not set:
  AX = number of bytes read

Function 3FH transfers count bytes from a file into a buffer location. It is not guaranteed that all count bytes will be read; for example, reading from the keyboard will read at most one line of text. If the returned value is zero, then the program has tried to read from the end of file.
All I/O is done using normalized pointers; no segment wraparound will occur.

Error returns:
AX
  6 = invalid handle
      The handle passed in BX was not currently open.
  5 = access denied
      The handle passed in BX was opened in a mode that did not allow reading.

Example

```
lds     dx, buf
mov     cx, count
mov     bx, handle
mov     ah, 3FH
int     21H
   ; ax has number of bytes read
```

Write to a File or Device (Function 40H)

Call
AH = 40H
DS:DX
  Pointer to buffer
CX
  Bytes to write
BX
  File handle

Return
Carry set:
AX
  Number of bytes written
  6 = invalid handle
  5 = access denied
Carry not set:
AX = number of bytes written

Function 40H transfers count bytes from a buffer into a file. It should be regarded as an error if the number of bytes written is not the same as the number requested.
The write system call with a count of zero (CX = 0) will set the file size to the current position. Allocation units are allocated or released as required.
All I/O is done using normalized pointers; no segment wraparound will occur.

Error returns:
AX
  6 = invalid handle
    The handle passed in BX was not currently open.
  5 = access denied
    The handle was not opened in a mode that allowed writing.

Example

```
lds     dx, buf
mov     cx, count
mov     bx, handle
mov     ah, 40H
int     21H
  ;ax has number of bytes written
```

Delete a Directory Entry (Function 41H)

Call
AH = 41H
DS:DX
  Pointer to pathname

Return
Carry set:
AX
  2 = file not found
  5 = access denied
Carry not set:
  No error

Function 41H removes a directory entry associated with a filename.

Error returns:
AX
  2 = file not found
      The path specified was invalid or not found.
  5 = access denied
      The path specified was a directory or read-only.

Example

```
lds     dx, name
mov     ah, 41H
int     21H
```

Move File Pointer (Function 42H)

Call
AH = 42H
CX:DX
  Distance to move, in bytes
AL
  Method of moving:
  (see text)
BX
  File handle

Return
Carry set:
AX
  6 = invalid handle
  1 = invalid function
Carry not set:
  DX:AX = new pointer location

Function 42H moves the read/write pointer according to one of the following methods:

Method	Function
0	The pointer is moved to offset bytes from the beginning of the file.
1	The pointer is moved to the current location plus offset.
2	The pointer is moved to the end of file plus offset.

Offset should be regarded as a 32-bit integer with CX occupying the most significant 16 bits.

Error returns:
AX
  6 = invalid handle
      The handle passed in BX was not currently open.
  1 = invalid function
      The function passed in AL was not in the range 0:2.

Example

```
mov     dx, offsetlow
mov     cx, offsethigh
mov     al, method
mov     bx, handle
mov     ah, 42H
int     21H
  ; dx:ax has the new location of the pointer
```

Change Attributes (Function 43H)

Call
AH = 43H
DS:DX
  Pointer to pathname
CX (if AL = 01)
  Attribute to be set
AL
  Function
  01 Set to CX
  00 Return in CX

Return
Carry set:
AX
  3 = path not found
  5 = access denied
  1 = invalid function
Carry not set:
  CX attributes (if AL = 00)

Given an ASCIZ name, Function 42H will set/get the attributes of the file to those given in CX.
A function code is passed in AL:

AL	Function
0	Return the attributes of the file in CX.
1	Set the attributes of the file to those in CX.

Error returns:
AX
  3 = path not found
      The path specified was invalid.
  5 = access denied
      The attributes specified in CX contained one that could not be changed (directory, volume ID).
  1 = invalid function
      The function passed in AL was not in the range 0:1.

Example

```
lds     dx, name
mov     cx, attribute
mov     al, func
int     ah, 43H
int     21H
```

I/O Control for Devices (Function 44H)

Call
AH = 44H
BX
  Handle
BL
  Drive (for calls AL = 4, 5
  0 = default, 1 = A, etc.)
DS:DX
  Data or buffer
CX
  Bytes to read or write
AL
  Function code; see text
Return
Carry set:
AX
  6 = invalid handle
  1 = invalid function
 13 = invalid data
  5 = access denied
Carry not set:
AL = 2,3,4,5
AX = Count transferred
AL = 6,7
  00 = Not ready
  FF = Ready

Function 44H sets or gets device information associated with an open handle, or send/receives a control string to a device handle or device. The following values are allowed for function:

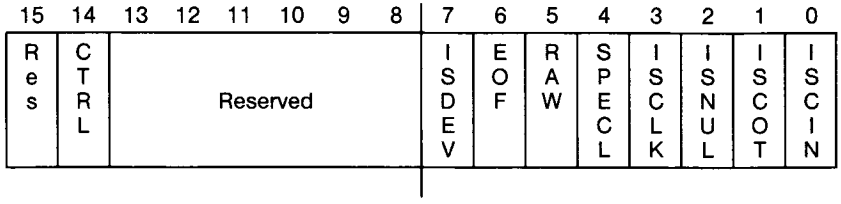
Request Function

0 Get device information (returned in DX)
1 Set device information (as determined by DX)
2 Read CX number of bytes into DS:DX from device control channel.
3 Write CX number of bytes from DS:DX to device control channel.
4 Same as 2 only drive number in BL 0=default,A:=1,B:=2,...
5 Same as 3 only drive number in BL 0=default,A:=1,B:=2,...
6 Get input status
7 Get output status

This function can be used to get information about device channels. Calls can be made on regular files, but only calls 0,6 and 7 are defined in that case (AL=0,6,7). All other calls return an invalid function error.

Calls AL=0 and AL=1

The bits of DX are defined as follows for calls AL=0 and AL=1. Note that the upper byte MUST be zero on a set call.



ISDEV = 1 if this channel is a device
= 0 if this channel is a disk file (Bits 8-15 = 0 in this case)

If ISDEV = 1

EOF = 0 if End Of File on input
RAW = 1 if this device is in Raw mode
= 0 if this device is cooked
ISCLK = 1 if this device is the clock device
ISNUL = 1 if this device is the null device
ISCOT = 1 if this device is the console output
ISCIN = 1 if this device is the console input
SPECL = 1 if this device is special

CTRL = 0 if this device can not do control strings via calls AL=2 and AL=3.
CTRL = 1 if this device can process control strings via calls AL=2 and AL=3.
NOTE that this bit cannot be set.

If ISDEV = 0

EOF = 0 if channel has been written
Bits 0-5 are the block device number for the channel (0 = A:, 1 = B:, ...)
Bits 15,8-13,4 are reserved and should not be altered.

Calls 2..5:

These four calls allow arbitrary control strings to be sent or received from a device. The call syntax is the same as the read and write calls, except for 4 and 5, which take a drive number in BL instead of a handle in BX.

An invalid function error is returned if the CTRL bit (see above) is 0.
An access denied is returned by calls AL=4,5 if the drive number is invalid.

Calls 6,7:

These two calls allow the user to check if a file handle is ready for input or output. Status of handles open to a device is the intended use of these calls, but status of a handle open to a disk file is allowed, and is defined as follows:

Input:

Always ready (AL=FF) until EOF reached, then always not ready (AL=0) unless current position changed via LSEEK.

Output:

Always ready (even if disk full).

IMPORTANT

The status is defined at the time the system is CALLED. On future versions, by the time control is returned to the user from the system, the status returned may NOT correctly reflect the true current state of the device or file.

Error returns:
AX
6 = invalid handle
The handle passed in BX was not currently open.
1 = invalid function
The function passed in AL was not in the range 0:7.
13 = invalid data
5 = access denied (calls AL=4..7)

Example

```
    mov         bx, Handle
(or mov         bl, drive       for calls AL=4,5
                                0=default,A:=1...)
    mov         dx, Data
(or lds         dx, buf         and
    mov         cx, count       for calls AL=2,3,4,5)
    mov         ah, 44H
    mov         al, func
    int         21H
  ; For calls AL=2,3,4,5 AX is the number of bytes
  ; transferred (same as READ and WRITE).
  ; For calls AL=6,7 AL is status returned, AL=0 if
  ; status is not ready, AL=0FFH otherwise.
```

Duplicate a File Handle (Function 45H)

Call
AH = 45H
BX
  File handle

Return
Carry set:
AX
  6 = invalid handle
  4 = too many open files
Carry not set:
AX = new file handle

Function 45H takes an already opened file handle and returns a new handle that refers to the same file at the same position.

Error returns:
AX
  6 = Invalid handle
      The handle passed in BX was not currently open.
  4 = too many open files
      There were no free handles available in the current process or the internal system tables were full.

Example

```
mov     bx, fh
mov     ah, 45H
int     21H
   ; ax has the returned handle
```

Force a Duplicat of a Handle (Function 46H)

Call
AH = 46H
BX
  Existing file handle
CX
  New file handle

Return
Carry set:
AX
  6 = invalid handle
  4 = too many open files
Carry not set:
  No error

Function 46H takes an already opened file handle and returns a new handle that refers to the same file at the same position. If there was already a file open on handle CX, it is closed first.

Error returns:
AX
  6 = invalid handle
      The handle passed in BX was not currently open.
  4 = too many open files
      There were no free handles available in the current process or the internal system tables were full.

Example

```
mov     bx, fh
mov     cx, newfh
mov     ah, 46H
int     21H
```

Return Text of Current Directory (Function 47H)

Call
AH = 47 H
DS:SI
  Pointer to 64-byte memory area
DL
  Drive number

Return
Carry set:
AX
  15 = invalid drive
Carry not set:
  No error

Function 47H returns the current directory for a particular drive. The directory is root-relative and does not contain the drive specifier or leading path separator. The drive code passed in DL is 0=default, 1=A:, 2=B:, etc.

Error returns:
AX
15 = invalid drive
    The drive specified in DL was invalid.

Example

```
mov     ah, 47H
lds     si,area
mov     dl,drive
int     21H
   ; ds:si is a pointer to 64 byte area that
   ; contains drive current directory.
```

Allocate Memory (Function 48H)

Call
AH = 48H
BX
  Size of memory to be allocated

Return
Carry set:
AX
  8 = not enough memory
  7 = arena trashed
BX
  Maximum size that could be allocated
Carry not set:
AX:0
  Pointer to the allocated memory

Function 48H returns a pointer to a free block of memory that has the requested size in paragraphs.

Error return:
AX
  8 = not enough memory
      The largest available free block is smaller than that requested or there is no free block.
  7 = arena trashed
      The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.

Example

```
mov     bx,size
mov     ah,48H
int     21H
   ; ax:0 is pointer to allocated memory
   ; if alloc fails, bx is the largest block available
```

Free Allocated Memory (Function 49H)

Call
AH = 49H
ES
  Segment address of memory
  area to be freed

Return
Carry set:
AX
  9 = invalid block
  7 = arena trashed
Carry not set:
  No error

Function 49H returns a piece of memory to the system pool that was allocated by Function Request 48H.

Error return:
AX
  9 = invalid block
      The block passed in ES is not one allocated via Function Request 48H.
  7 = arena trashed
      The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.

Example

```
mov     es,block
mov     ah,49H
int     21H
```

Call
AH = 4AH
ES
  Segment address of memory area
BX
  Requested memory area size

Return
Carry set:
AX
  9 = invalid block
  7 = arena trashed
  8 = not enough memory
BX
  Maximum size possible
Carry not set:
  No error

Function 4AH will attempt to grow/shrink an allocated block of memory.

Error return:
AX
  9 = invalid block
      The block passed in ES is not one allocated via this function.
  7 = arena trashed
      The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.
  8 = not enough memory
      There was not enough free memory after the specified block to satisfy the grow request.

Example

```
mov     es,block
mov     bx,newsize
mov     ah,4AH
int     21H
   ; if setblock fails for growing, BX will have the
   ; maximum size possible
```

Load and Execute a program (Function 4BH)

Call
AH = 4BH
DS:DX
  Pointer to pathname
ES:BX
  Pointer to parameter block
AL
  00 = Load and execute program
  03 = Load program

Return
Carry set:
AX
  1 = invalid function
  10 = bad environment
  11 = bad format
  8 = not enough memory
  2 = file not found
Carry not set:
  No error

This function allows a program to load another program into memory and (default) begin execution of it. DS:DX points to the ASCIZ name of the file to be loaded. ES:BX points to a parameter block for the load.
A function code is passed in AL:

AL Function

0 Load and execute the program. A program header is established for the program and the terminate and CONTROL-C addresses are set to the instruction after the EXEC system call.

3 Load (do not create) the program header, and do not begin execution. This is useful in loading program overlays.

For each value of AL, the block has the following format:

AL = 0 –> load/execute program

WORD segment address of environment.
DWORD pointer to command line at 80H
DWORD pointer to default FCB to be passed at 5CH
DWORD pointer to default FCB to be passed at 6CH

AL = 3 –> load overlay

WORD segment address where file will be loaded.
WORD relocation factor to be applied to the image.

Note that all open files of a process are duplicated in the child process after an EXEC. This is extremely powerful; the parent process has control over the meanings of stdin, stdout, stderr, stdaux and stdprn. The parent could, for example, write a series of records to a file, open the file as standard input, open a listing file as standard output and then EXEC a sort program that takes its input from stdin and writes to stdout.

Also inherited (or passed from the parent) is an "environment". This is a block of text strings (less than 32K bytes total) that convey various configurations parameters. The format of the environment is as follows:

(paragraph boundary)

BYTE ASCIZ string 1
BYTE ASCIZ string 2
. . .
BYTE ASCIZ string n
BYTE of zero

Typically the environment strings have the form:

parameter = value

For example, COMMAND.COM might pass its execution search path as:

PATH = A:XBIN;B:XBASICXLIB

A zero value of the environment address causes the child process to inherit the parent's environment unchanged.

Error returns:

AX

1 = invalid function
The function passed in AL was not 0, 1 or 3.

10 = bad environment
The environment was larger than 32Kb.

11 = bad format
The file pointed to by DS:DX was an EXE format file and contained information that was internally inconsistent.

8 = not enough memory
There was not enough memory for the process to be created.

2 = file not found
The path specified was invalid or not found.

Example

```
lds     dx, name
les     bx, blk
mov     ah, 4BH
mov     al, func
int     21H
```

Terminate a Process (Function 4CH)

Call
AH = 4CH
AL
  Return code

Return
None

Function 4CH terminates the current process and transfers control to the invoking process. In addition, a return code may be sent. All files open at the time are closed.
This method is preferred over all others (Interrupt 20H, JMP 0) and has the advantage that CS:0 does not have to point to the Program Header Prefix.

Error returns:
  None.

Example

```
mov     al, code
mov     ah, 4CH
int     21H
```

Retrieve the Return Code of a Child (Function 4DH)

Call
AH = 4DH

Return
AX
Exit Code

Function 4DH returns the Exit code specified by a child process. It returns this Exit code only once. The low byte of this code is that sent by the Exit routine. The high byte is one of the following:

0 = Terminate/abort
1 = CONTROL-C
2 = Hard error
3 = Terminate and stay resident

Error returns:
None.

Example

```
mov     ah, 4DH
int     21H
  ; ax has the exit code
```

Call
AH = 4EH
DS:DX
  Pointer to pathname
CX
  Search attributes

Return
Carry set:
AX
  2 = file not found
  18 = no more files
Carry not set:
  no error

Function 4EH takes a pathname with wild card characters in the last component (passed in DS:DX), a set of attributes (passed in CX) and attempts to find all files that match the pathname and have a subset of the required attributes. A datablock at the current DMA is written that contains information in the following form:

```
find-buf-reserved  DB   21  DUP (?); Reserved*
find-buf-attr      DB   ?   ; attribute found
find-buf-time      DW   ?   ; time
find-buf-date      DW   ?   ; date
find-buf-size-l    DW   ?   ; low(size)
find-buf-size-h    DW   ?   ; high(size)
find-buf-pname     DB   13  DUP (?) ; packed name
find-buf    ENDS
```

*Reserved for MS-DOS use on subsequent find-nexts

To obtain the subsequent matches of the pathname, see the description of Function 4FH.

Error returns:
AX
 2 = file not found
      The path specified in DS:DX was an invalid path.
18 = no more files
      There were no files matching this specification.

Example

```
        mov     ah, 4EH
        lds     dx, pathname
        mov     cx, attr
        int     21H
          ; dma address has datablock
```

Call
AH = 4FH

Return
Carry set:
AX
  18 = no more files
Carry not set:
  No error

Function 4FH finds the next matching entry in a directory. The current DMA address must point at a block returned by Function 4EH (see Function 4EH).

Error returns:
AX
18 = no more files
  There are no more files matching this pattern.

Example

```
    ; dma points at area returned by Function 4FH
mov     ah, 4FH
int     21H
    ; next entry is at dma
```

Return Current Setting of Verify After Write Flag (Function 54H)

Call
AH = 54H

Return
AL
Current verify flag value

The current value of the verify flag is returned in AL.

Error returns:
None.

Example

```
mov     ah, 54H
int     21H
  ; al is the current verify flag value
```

Move a Directory Entry (Function 56H)

Call
AH = 56H
DS:DX
  Pointer to pathname of
  existing file
ES:DI
  Pointer to new pathname

Return
Carry set:
AX
  2 = file not found
 17 = not same device
  5 = access denied
Carry not set:
  No error

Function 56H attempts to rename a file into another path. The paths must be on the same device.

Error returns:
AX
 2 = file not found
     The file name specifed by DS:DX was not found.
17 = not same device
     The source and destination are on different drives.
 5 = access denied
     The path specified in DS:DX was a directory or the file specified by ES:DI exists or the destination directory entry could not be created.

Example

```
lds     dx, source
les     di, dest
mov     ah, 56H
int     21H
```

Get/Set Date/Time of File (Function 57H)

Call
AH = 57H
AL
  00 = get date and time
  01 = set date and time
BX
  File handle
CX (if AL = 01)
  Time to be set
DX (if AL = 01)
  Date to be set

Return
Carry set:
AX
  1 = invalid function
  6 = invalid handle
Carry not set:
No error
CX/DX set if function 0

Function 57H returns or sets the last-write time for a handle. These times are not recorded until the file is closed.
A function code is passed in AL:

AL	Function
0	Return the time/date of the handle in CX/DX
1	Set the time/date of the handle to CX/DX

Error returns:
AX
 1 = invalid function
     The function passed in AL was not in the range 0:1.
 6 = invalid handle
     The handle passed in BX was not currently open.

Example

```
        mov     ah, 57H
        mov     al, func
        mov     bx, handle
            ; if al = 1 then next two are mandatory
        mov     cx, time
        mov     dx, date
        int     21H
          ;  if al = 0 then cx/dx has the last write time/date
          ; for the handle.
```

1.8 MACRO DEFINITIONS FOR MS-DOS SYSTEM CALL EXAMPLES

NOTE

These macro definitions apply to system call examples 00H through 57H.

```
.xlist
;
;*******************
; Interrupts
;*******************
;
                                        ;ABS-DISK-READ
;abs-disk-read macro disk,buffer,num-sectors,first-sector
        mov     al,disk
        mov     bx,offset buffer
        mov     cx,num-sectors
        mov     dx,first-sector
        int     37              ;interrupt 37
        popf
        endm
;
                                        ;ABS-DISK-WRITE
abs-disk-write macro disk,buffer,num-sectors,first-sector
        mov     al,disk
        mov     bx,offset buffer
        mov     cx,num-sectors
        mov     dx,first-sector
        int     38              ;interrupt 38
        popf
        endm
;
stay-resident macro last-instruc        ;STAY-RESIDENT
        mov     dx,offset last-instruc
        inc     dx
        int     39              ;interrupt 39
        endm
;
;*******************
; Functions
;*******************
;
;
read-kbd-and-echo macro                 ;READ-KBD-AND-ECHO
        mov     ah,1            ;function 1
        int     33
        endm
;
display-char macro character            ;DISPLAY-CHAR
        mov     dl,character
        mov     ah,2            ;function 2
```

```
                int     33
                endm
;
aux-input macro                         ;AUX-INPUT
                mov     ah,3            ;function 3
                int     33
                endm
;
aux-output macro                        ;AUX-OUTPUT
                mov     ah,4            ;function 4
                int     33
                endm
;;page
print-char macro        character       ;PRINT-CHAR
                mov     dl,character
                mov     ah,5            ;function 5
                int     33
                endm
;
dir-console-io macro switch             ;DIR-CONSOLE-IO
                mov     dl,switch
                mov     ah,6            ;function 6
                int     33
                endm
;
dir-console-input macro                 ;DIR-CONSOLE-INPUT
                mov     ah,7            ;function 7
                int     33
                endm
;
read-kbd        macro                   ;READ-KBD
                mov     ah,8            ;function 8
                int     33
                endm
;
display         macro   string          ;DISPLAY
                mov     dx,offset string
                mov     ah,9            ;function 9
                int     33
                endm
;
get-string macro        limit,string    ;GET-STRING
                mov     String,limit
                mov     dx,offset string
                mov     ah,10           ;function 10
                int     33
                endm
;
check-kbd-status macro                  ;CHECK-KBD-STATUS
                mov     ah,11           ;function 11
                int     33
                endm
;
```

```
flush-and-read-kbd  macro switch    ;FLUSH-AND-READ-KBD
                    mov   al,switch
                    mov   ah,12     ;function 12
                    int   33
                    endm
;
reset-disk macro                    ;RESET DISK
                    mov   ah,13     ;function 13
                    int   33
                    endm
;;page
select-disk macro   disk            ;SELECT-DISK
                    mov   dl,disk[-65]
                    mov   ah,14     ;function 14
                    int   33
                    endm
;
open        macro   fcb             ;OPEN
                    mov   dx,offset fcb
                    mov   ah,15     ;function 15
                    int   33
                    endm
;
close       macro   fcb             ;CLOSE
                    mov   dx,offset fcb
                    mov   ah,16     ;function 16
                    int   33
                    endm
;
search-first macro  fcb             ;SEARCH-FIRST
                    mov   dx,offset fcb
                    mov   ah,17     ;Function 17
                    int   33
                    endm
;
search-next macro   fcb             ;SEARCH-NEXT
                    mov   dx,offset fcb
                    mov   ah,18     ;function 18
                    int   33
                    endm
;
delete      macro   fcb             ;DELETE
                    mov   dx,offset fcb
                    mov   ah,19     ;function 19
                    int   33
                    endm
;
read-seq    macro   fcb             ;READ-SEQ
                    mov   dx,offset fcb
                    mov   ah,20     ;function 20
                    int   33
                    endm
;
```

```
write-seq macro     fcb             ;WRITE-SEQ
          mov       dx,offset fcb
          mov       ah,21           ;function 21
          int       33
          endm
;
create    macro     fcb             ;CREATE
          mov       dx,offset fcb
          mov       ah,22           ;function 22
          int       33
          endm
;
rename    macro     fcb,newname     ;RENAME
          mov       dx,offset fcb
          mov       ah,23           ;function 23
          int       33
          endm
;
current-disk macro                  ;CURRENT-DISK
          mov       ah,25           ;function 25
          int       33
          endm
;
set-dta   macro     buffer          ;SET-DTA
          mov       dx,offset buffer
          mov       ah,26           ;function 26
          int       33
          endm
;
alloc-table macro                   ;ALLOC-TABLE
          mov       ah,27           ;function 27
          int       33
          endm
;
read-ran  macro     fcb             ;READ-RAN
          mov       dx,offset fcb
          mov       ah,33           ;function 33
          int       33
          endm
;
write-ran macro     fcb             ;WRITE-RAN
          mov       dx,offset fcb
          mov       ah,34           ;function 34
          int       33
          endm
;
file-size macro     fcb             ;FILE-SIZE
          mov       dx,offset fcb
          mov       ah,35           ;function 35
          int       33
          endm
;
```

```
set-relative-record macro fcb              ;SET-RELATIVE-RECORD
                mov     dx,offset fcb
                mov     ah,36              ;function 36
                int     33
                endm
;;page
set-vector macro interrupt,seg-addr,off-addr ;SET-VECTOR
                push
                mov     ax,seg-addr
                mov     ds,ax
                mov     dx,off-addr
                mov     al,interrupt
                mov     ah,37              ;function 37
                int     33
                endm
;
create-prog-seg macro seg-addr             ;CREATE-PROG-SEG
                mov     dx,seg-addr
                mov     ah,38              ;function 38
                int     33
                endm
;
ran-block-read macro fcb,count,rec-size ;RAN-BLOCK-READ
                mov     dx,offset fcb
                mov     cx,count
                mov     word ptr fcb[14],rec-size
                mov     ah,39              ;function 39
                int     33
                endm
;
ran-block-write macro fcb,count,rec-size ;RAN-BLOCK-WRITE
                mov     dx,offset fcb
                mov     cx,count
                mov     word ptr fcb[14],rec-size
                mov     ah,40              ;function 40
                int     33
                endm
;
parse           macro   filename,fcb       ;PARSE
                mov     si,offset filename
                mov     di,offset fcb
                push    es
                push    ds
                pop     es
                mov     al,15
                mov     ah,41              ;function 41
                int     33
                pop     es
                endm
;
get-date        macro                      ;GET-DATE
                mov     ah,42              ;function 42
                int     33
```

```
            endm
;;page
set-date    macro   year,month,day  ;SET-DATE
            mov     cx,year
            mov     dh,month
            mov     dl,day
            mov     ah,43           ;function 43
            int     33
            endm
;
get-time    macro                   ;GET-TIME
            mov     ah,44           ;function 44
            int     33
            endm
;

set-time    macro   hour,minutes,seconds,hundredths ;SET-TIME
            mov     ch,hour
            mov     cl,minutes
            mov     dh,seconds
            mov     dl,hundredths
            mov     ah,45           ;function 45
            int     33
            endm
;
verify      macro   switch          ;VERIFY
            mov     al,switch
            mov     ah,46           ;function 46
            int     33
            endm
;
;********************
;
; General
;********************
;
;
move-string macro source,destination,num-bytes ;MOVE-STRING
            push    es
            mov     ax,ds
            mov     es,ax
            assume  es:data
            mov     si,offset source
            mov     di,offset  destina-
                    tion
            mov     cx,num-bytes
        rep movs    es:destination,source
            assume  es:nothing
            pop     es
            endm
;
;
convert     macro   value,base,destination ;CONVERT
            local   table,start
            jmp     start
```

```
table           db      "0123456789ABCDEF"
start:          mov     al,value
                xor     ah,ah
                xor     bx,bx
                div     base
                mov     bl,al
                mov     al,cs:table[bx]
                mov     destination,al
                mov     bl,ah
                mov     al,cs:table[bx]
                mov     destination[1],al
                endm
;;page
convert-to-binary macro string,number,value ;CONVERT-TO-BINARY
                local   ten,start,calc,mult,no-mult
                jmp     start
ten             db      10
start:          mov     value,0
                xor     cx,cx
                mov     cl,number
                xor     si,si
calc:           xor     ax,ax
                mov     al,string[si]
                sub     al,48
                cmp     cx,2
                jl      no-mult
                push    cx
                dec     cx
mult:           mul     cs:ten
                loop    mult
                pop     cx
no-mult:        add     value,ax
                inc     si
                loop    calc
                endm
;
convert-date macro      dir-entry
                mov     dx,word ptr dir-entry[25]
                mov     cl,5
                shr     dl,cl
                mov     dh,dir-entry[25]
                and     dh,1fh
                xor     cx,cx
                mov     cl,dir-entry[26]
                shr     cl,1
                add     cx,1980
                endm
;
```

1.9 EXTENDED EXAMPLE OF MS-DOS SYSTEM CALLS

```
title DISK DUMP
zero                    equ     0
disk-B                  equ     1
sectors-per-read        equ     9
cr                      equ     13
blank                   equ     32
period                  equ     46
tilde                   equ     126
    INCLUDE B:CALLS.EQU
;
subttl DATA SEGMENT
page +
data                    segment
;
input-buffer            db      9 dup(512 dup(?))
output-buffer           db      77 dup(" ")
                        db      0DH,0AH,"$"
start-prompt            db      "Start at sector: $"
sectors-prompt          db      "Number of sectors: $"
continue-prompt         db      "RETURN to continue $"
header                  db      "Relative sector $"
end-string              db      0DH,0AH,0AH,07H,"ALL DONE$"
                                ;DELETE THIS
crlf                    db      0DH,0AH,"$"
table                   db      "0123456789ABCDEF$"
;
ten                     db      10
sixteen                 db      16
;
start-sector            dw      1
sector-num              label   byte
sector-number           dw      0
sectors-to-dump         dw      sectors-per-read
sectors-read            dw      0
;
buffer                  label   byte
max-length              db      0
current-length          db      0
digits                  db      5 dup(?)
;
data                    ends
;
subttl STACK SEGMENT
page +
stack                   seg-
                        ment    stack
                        dw      100 dup(?)
stack-top               label   word
stack                   ends
;
subttl MACROS
page +
;
```

```
     INCLUDE B:CALLS.MAC
;BLANK LINE
blank-line          macro   number
                    local   print-it
                    push    cx
                    call    clear-line
                    mov     cx,number
print-it:           display output-buffer
                    loop    print-it
                    pop     cx
                    endm
;
subttl ADDRESSABILITY
page +
code                segment
                    assume  cs:code,ds:data,ss:stack
start:              mov     ax,data
                    mov     ds,ax
                    mov     ax,stack
                    mov     ss,ax
                    mov     sp,offset stack-top
;
                    jmp     main-procedure
subttl PROCEDURES
page +
;
; PROCEDURES
; READ-DISK
read-disk           proc;
                    cmp     sectors-to-dump-zero
                    jle     done
                    mov     bx,offset input-buffer
                    mov     dx,start-sector
                    mov     al,disk-b
                    mov     cx,sectors-per-read
                    cmp     cx,sectors-to-dump
                    jle     get-sector
                    mov     cx,sectors-to-dump
get-sector:         push    cx
                    int     disk-read
                    popf
                    pop     cx
                    sub     sectors-to-dump,cx
                    add     start-sector,cx
                    mov     sectors-read,cx
                    xor     si,si
done:               ret
read-disk           endp
;CLEAR-LINE
clear-line          proc;
                    push    cx
                    mov     cx,77
                    xor     bx,bx
move-blank:         mov     output-buffer[bx],"  "
                    inc     bx
```

```
                        loop    move-blank
                        pop     cx
                        ret
clear-line              endp
;
;PUT-BLANK
put-blank               proc;
                        mov     output-buffer[di], "  "
                        inc     di
                        ret
put-blank               endp
;
;
setup                   proc;
                        display start-prompt
                        get-string 4,buffer
                        display crlf
                        convert-to-binary digits,
                        current-length,start-sector
                        mov     ax,start-sector
                        mov     sector-number,ax
                        display sectors-prompt
                        get-string 4,buffer
                        convert-to-binary digits,
                        current-length,sectors-to-dump
                        ret
setup                   endp
;
;CONVERT-LINE
convert-line            proc;
                        push    cx
                        mov     di,9
                        mov     cx,16
convert-it              convert input-buffer[si],sixteen,
                        output-buffer[di]
                        inc     si
                        add     di,2
                        call    put-blank
                        loop    convert-it
                        sub     si,16
                        mov     cx,16
                        add     di,4
display-ascii:          mov     output-buffer[di],period
                        cmp     input-buffer[si],blank
                        jl      non-printable
                        cmp     input-buffer[si],tilde
                        jg      non-printable
printable:              mov     dl,input-buffer[si]
                        mov     output-buffer[di],dl
non-printable:          inc     si
                        inc     di
                        loop    display-ascii
                        pop     cx
                        ret
convert-line            endp
```

```
;
;DISPLAY-SCREEN
display-screen          proc;
                        push    cx
                        call    clear-line
;
                        mov     cx,17
;I WANT length header
                        dec     cx
;minus 1 in cx
                        xor     di,di
move-header:            mov     al,header[di]
                        mov     output-buffer[di],al
                        inc     di
                        loop    move-header   ;FIX THIS!
;
                        convert sector-num[1],sixteen,
                        output-buffer[di]
                        add     di,2
                        convert sector-num,sixteen,
                        output-buffer[di]
                        display output-buffer
                        blank-line 2
                        mov     cx,16
dump-it:                call    clear-line
                        call    convert-line
                        display output-buffer
                        loop    dump-it
                        blank-line 3
                        display continue-prompt
                        get-char-no-echo
                        display crlf
                        pop     cx
                        ret
display-screen          endp
;
;
; END PROCEDURES
subttl MAIN PROCEDURE
page +
main-procedure:         call    setup
check-done:             cmp     sectors-to-dump,zero
                        jng     all-done
                        call    read-disk
                        mov     cx,sectors-read
display-it:             call    display-screen
                        call    display-screen
                        inc     sector-number
                        loop    display-it
                        jmp     check-done
all-done:               display end-string
                        get-char-no-echo
code                    ends
                        end     start
```

CHAPTER 2
MS-DOS 2.0 DEVICE DRIVERS

2.1 WHAT IS A DEVICE DRIVER?

A device driver is a binary file with all of the code in it to manipulate the hardware and provide a consistent interface to MS-DOS. In addition, it has a special header at the beginning that identifies it as a device, defines the strategy and interrupt entry points, and describes various attributes of the device.

NOTE

For device drivers, the file must not use the ORG 100H (like .COM files). Because it does not use the Program Segment Prefix, the device driver is simply loaded; therefore, the file must have an origin of zero (ORG 0 or no ORG statement).

There are two kinds of device drivers.

1. Character device drivers
2. Block device drivers

Character devices are designed to perform serial character I/O like CON, AUX, and PRN. These devices are named (i.e., CON, AUX, CLOCK, etc.), and users may open channels (handles or FCBs) to do I/O to them.
Block devices are the "disk drives" on the system. They can perform random I/O in pieces called blocks (usually the physical sector size). These devices are not named as the character devices are, and therefore cannot be opened directly. Instead they are identified via the drive letters (A:,B:,C:, etc.).

Block devices also have units. A single driver may be responsible for one or more disk drives. For example, block device driver ALPHA

may be responsible for drives A:,B:,C: and D:. This means that it has four units (0-3) defined and, therefore, takes up four drive letters. The position of the driver in the list of all drivers determines which units correspond to which driver letters. If driver ALPHA is the first block driver in the device list, and it defines 4 units (0-3), then they will be A:,B:,C: and D:. If Beta is the second block driver and defines three units (0-2), then they will be E:,F: and G:, and so on. MS-DOS 2.0 is not limited to 16 block device units, as previous versions were. The theoretical limit is 63 (26 - 1), but it should be noted that after 26 the drive letters are unconventional (such as], \, and ^).

NOTE

Character devices cannot define multiple units because they have only one name.

2.2 DEVICE HEADERS

A device header is required at the beginning of a device driver. A device header looks like this:

DWORD pointer to next device (Must be set to -1)
WORD attributes Bit 15 = 1 if char device 0 is blk if bit 15 is 1 Bit 0 = 1 if current sti device Bit 1 = 1 if current sto output Bit 2 = 1 if current NUL device Bit 3 = 1 if current CLOCK dev Bit 4 = 1 if special Bits 5 - 12 Reserved; must be set to 0 Bit 14 is the IOCTL bit Bit 13 is the NON IBM FORMAT bit
WORD pointer to device strategy entry point
WORD pointer to device interrupt entry point
8-BYTE character device name field Character devices set a device name. For block devices the first byte is the number of units.

Figure 2. Sample Device Header

Note that the device entry points are words. They must be offsets from the same segment number used to point to this table. For example, if XXX:YYY points to the start of this table, then XXX:strategy and XXX:interrupt are the entry points.

2.2.1 Pointer To Next Device Field

The pointer to the next device header field is a double word field (offset followed by segment) that is set by MS-DOS to point at the next driver in the system list at the time the device driver is loaded. It is important that this field be set to -1 prior to load (when it is on the disk as a file) unless there is more than one device driver in the file. If there is more than one driver in the file, the first word of the double word pointer should be the offset of the next driver's Device Header.

NOTE

If there is more than one device driver in the .COM file, the **last** driver in the file must have the pointer to the next Device Header field set to -1.

2.2.2 Attribute Field

The attribute field is used to tell the system whether this device is a block or character device (bit 15). Most other bits are used to give selected character devices certain special treatment. (Note that these bits mean nothing on a block device). For example, assume that a user has a new device driver that he wants to be the standard input and output. Besides installing the driver, he must tell MS-DOS that he wants his new driver to override the current standard input and standard output (the CON device). This is accomplished by setting the attributes to the desired characteristics, so he would set bits 0 and 1 to 1 (note that they are separate!) Similarly, a new CLOCK device could be installed by setting that attribute. (Refer to section 2.7, "The CLOCK Device", in this chapter for more information.) Although there is a NUL device attribute, the NUL device cannot be reassigned. This attribute exists so that MS-DOS can determine if the NUL device is being used.

The NON IBM FORMAT bit applies only to block devices and affects the operation of the BUILD BPB (Bios Parameter Block) device call. (Refer to section 2.5.3 for further information on this call).

The other bit of interest is the IOCTL bit, which has meaning on character and block devices. This bit tells MS-DOS whether the device can handle control strings (via the IOCTL system call, Function 44H).

If a driver cannot process control strings, it should initially set this bit to 0. This tells MS-DOS to return an error if an attempt is made (via Function 44H) to send or receive control strings to this device. A device which can process control strings should initialize the IOCTL bit to 1. For drivers of this type, MS-DOS will make calls to the IOCTL INPUT and OUTPUT device functions to send and receive IOCTL strings.

The IOCTL functions allow data to be sent and received by the device for its own use (for example, to set baud rate, stop bits, and form length), instead of passing data over the device channel as does a normal read or write. The interpretation of the passed information is up to the device, but it **must not** be treated as a normal I/O request.

2.2.3 Strategy And Interrupt Routines

These two fields are the pointers to the entry points of the strategy and interrupt routines. They are word values, so they must be in the same segment as the Device Header.

2.2.4 Name Field

This is an 8-byte field that contains the name of a character device or the number of units of a block device. If it is a block device, the number of units can be put in the first byte. This is optional, because MS-DOS will fill in this location with the value returned by the driver's INIT code. Refer to Section 2.4, "Installation of Device Drivers" in this chapter for more information.

2.3 HOW TO CREATE A DEVICE DRIVER

In order to create a device driver that MS-DOS can install, you must write a binary file with a Device Header at the beginning of the file. Note that for device drivers, the code should not be originated at 100H, but rather at 0. The link field (pointer to next Device Header) should be -1, unless there is more than one device driver in the file. The attribute field and entry points must be set correctly.
If it is a character device, the name field should be filled in with the name of that character device. The name can be any legal 8-character filename.
MS-DOS always processes installable device drivers before handling the default devices, so to install a new CON device, simply name the device CON. Remember to set the standard input device and standard output device bits in the attribute word on a new CON device. The scan of the device list stops on the first match, so the installable device driver takes precedence.

NOTE

Because MS-DOS can install the driver anywhere in memory, care must be taken in any far memory references. You should not expect that your driver will always be loaded in the same place every time.

2.4 INSTALLATION OF DEVICE DRIVERS

MS-DOS 2.0 allows new device drivers to be installed dynamically at boot time. This is accomplished by INIT code in the BIOS, which reads and processes the CONFIG.SYS file.
MS-DOS calls upon the device drivers to perform their function in the following manner:

MS-DOS makes a far call to strategy entry, and passes (in a Request Header) the information describing the functions of the device driver.

This structure allows you to program an interrupt-driven device driver. For example, you may want to perform local buffering in a printer.

2.5 REQUEST HEADER

When MS-DOS calls a device driver to perform a function, it passes a Request Header in ES:BX to the strategy entry point. This is a fixed length header, followed by data pertinent to the operation being performed. Note that it is the device driver's responsibility to preserve the machine state (for example, save all registers on entry and restore them on exit). There is enough room on the stack when strategy or interrupt is called to do about 20 pushes. If more stack is needed, the driver should set up its own stack.
The following figure illustrates a Request Header.

REQUEST HEADER – >

BYTE length of record Length in bytes of this Request Header
BYTE unit code The subunit the operation is for (minor device) (no meaning on character devices)
BYTE command code
WORD status
8 bytes RESERVED

Figure 3. Request Header

2.5.1 Unit Code

The unit code field identifies which unit in your device driver the request is for. For example, if your device driver has 3 units defined, then the possible values of the unit code field would be 0, 1, and 2.

2.5.2 Command Code Field

The command code field in the Request header can have the following values:

Command Code	Function
0	INIT
1	MEDIA CHECK (Block only, NOP for character)
2	BUILD BPB " " " " "
3	IOCTL INPUT (Only called if device has IOCTL)
4	INPUT (read)
5	NON-DESTRUCTIVE INPUT NO WAIT (Char devs only)
6	INPUT STATUS " " "
7	INPUT FLUSH " " "
8	OUTPUT (write)
9	OUTPUT (write) with verify
10	OUTPUT STATUS " " "
11	OUTPUT FLUSH " " "
12	IOCTL OUTPUT (Only called if device has IOCTL)

2.5.3 MEDIA CHECK AND BUILD BPB

MEDIA CHECK and BUILD BPB are used with block devices only. MS-DOS calls MEDIA CHECK first for a drive unit. MS-DOS passes its current media descriptor byte (refer to the section "Media Descriptor Byte" later in this chapter). MEDIA CHECK returns one of the following results:

> Media Not Changed - current DPB and media byte are OK.
> Media Changed - Current DPB and media are wrong. MS-DOS invalidates any buffers for this unit and calls the device driver to build the BPB with media byte and buffer.
> Not Sure - If there are dirty buffers (buffers with changed data, not yet written to disk) for this unit, MS-DOS assumes the DPB and media byte are OK (media not changed). If nothing is dirty, MS-DOS assumes the media has changed. It invalidates any buffers for the unit, and calls the device driver to build the BPB with media byte and buffer.
> Error - If an error occurs, MS-DOS sets the error code accordingly.

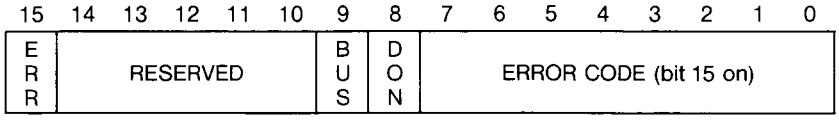
MS-DOS will call BUILD BPB under the following conditions:

> If Media Changed is returned
>
> If Not Sure is returned, and there are no dirty buffers

The BUILD BPB call also gets a pointer to a one-sector buffer. What this buffer contains is determined by the NON IBM FORMAT bit in the attribute field. If the bit is zero (device is IBM format-compatible), then the buffer contains the first sector of the first FAT. The FAT ID byte is the first byte of this buffer. NOTE: The BPB must be the same, as far as location of the FAT is concerned, for all possible media because this first FAT sector must be read **before** the actual BPB is returned. If the NON IBM FORMAT bit is set, then the pointer points to one sector of scratch space (which may be used for anything).

2.5.4 Status Word

The following figure illustrates the status word in the Request Header.



The Status word is zero on entry and is set by the driver interrupt routine on return.
Bit 8 is the done bit. When set, it means the operation is complete. For MS-DOS 2.0, the driver sets it to 1 when it exits.
Bit 15 is the error bit. If it is set, then the low 8 bits indicate the error. The errors are:

0 Write protect violation
1 Unknown Unit
2 Drive not ready
3 Unknown command
4 CRC error
5 Bad drive request structure length
6 Seek error
7 Unknown media
8 Sector not found
9 Printer out of paper
A Write fault
B Read Fault
C General failure

Bit 9 is the busy bit, which is set only by status calls.

> **For output on character devices:** If bit 9 is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request, and a write request (if made) would start immediately.

For input on character devices with a buffer: If bit 9 is 1 on return, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the device buffer and a read would return quickly. It also indicates that something has been typed. MS-DOS assumes all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer should always return busy=0 so that MS-DOS will not continuously wait for something to get into a buffer that does not exist.

One of the functions defined for each device is INIT. This routine is called only once when the device is installed. The INIT routine returns a location (DS:DX), which is a pointer to the first free byte of memory after the device driver (similar to "Keep Process"). This pointer method can be used to delete initialization code that is only needed once, saving on space.
Block devices are installed the same way and also return a first free byte pointer as described above. Additional information is also returned:

The number of units is returned. This determines logical device names. If the current maximum logical device letter is F at the time of the install call, and the INIT routine returns 4 as the number of units, then they will have logical names G, H, I and J. This mapping is determined by the position of the driver in the device list, and by the number of units on the device (stored in the first byte of the device name field).
A pointer to a BPB (BIOS Paramter Block) pointer array is also returned. There is one table for each unit defined. These blocks will be used to build an internal DOS data structure for each of the units. The pointer passed to the DOS from the driver points to an array of n word pointers to BPBs, where n is the number of units defined. In this way, if all units are the same, all of the pointers can point to the same BPB, saving space. Note that this array must be protected (below the free pointer set by the return) since an internal DOS structure will be built starting at the byte pointed to by the free pointer. The sector size defined must be less than or equal to the maximum sector size defined at default BIOS INIT time. If it isn't, the install will fail.
The last thing that INIT of a block device must pass back is the media descriptor byte. This byte means nothing to MS-DOS, but is passed to devices so that they know what parameters MS-DOS is currently using for a particular drive unit.

Block devices may take several approaches; they may be **dumb** or **smart.** A dumb device defines a unit (and therefore an internal DOS structure) for each possible media drive combination. For example, unit 0 = drive 0 single side, unit 1 = drive 0 double side. For this approach, media descriptor bytes do not mean anything. A smart device allows multiple media per unit. In this case, the BPB table returned at INIT must define space large enough to accommodate the largest possible media supported. Smart drivers will use the media descriptor byte to pass information about what media is currently in a unit.

2.6 FUNCTION CALL PARAMETERS

All strategy routines are called with ES:BX pointing to the Request Header. The interrupt routines get the pointers to the Request Header from the queue that the strategy routines store them in. The command code in the Request Header tells the driver which function to perform.

NOTE

All DWORD pointers are stored offset first, then segment.

2.6.1 INIT

Command code = 0

INIT – ES:BX – >

13-BYTE Request Header
BYTE # of units
DWORD break address
DWORD pointer to BPB array (Not set by character devices)

The number of units, break address, and BPB pointer are set by the driver. On entry, the DWORD that is to be set to the BPB array (on block devices) points to the character after the "=" on the line in CONFIG.SYS that loaded this device. This allows drivers to scan the CONFIG.SYS invocation line for arguments.

NOTE

If there are multiple device drivers in a single .COM file, the ending address returned by the last INIT called will be the one MS-DOS uses. It is recommended that all of the device drivers in a single .COM file return the same ending address.

2.6.2 MEDIA CHECK

Command Code = 1
MEDIA CHECK - ES:BX -

13-BYTE Request Header
BYTE media descriptor from DPB
BYTE returned

In addition to setting the status word, the driver must set the return byte to one of the following:

-1 Media has been changed
0 Don't know if media has been changed
1 Media has not been changed

If the driver can return -1 or 1 (by having a door-lock or other interlock mechanism) MS-DOS performance is enhanced because MS-DOS does not need to reread the FAT for each directory access.

2.6.3 BUILD BPB (BIOS Paramter Block)

Command code = 2
BUILD BPB – ES:BX – >

13–BYTE Request Header
BYTE media descriptor from DPB
DWORD transfer address (Points to one sector worth of scratch space or first sector of FAT depending on the value of the NON IBM FORMAT bit)
DWORD pointer to BPB

If the NON IBM FORMAT bit of the device is set, then the DWORD transfer address points to a one sector buffer, which can be used for any purpose. If the NON IBM FORMAT bit is 0, then this buffer contains the first sector of the first FAT and the driver must not alter this buffer.
If IBM compatible format is used (NON IBM FORMAT BIT = 0), then the first sector of the first FAT must be located at the same sector on all possible media. This is because the FAT sector will be read BEFORE the media is actually determined. Use this mode if all you want is to read the FAT ID byte.
In addition to setting status word, the driver must set the Pointer to the BPB on return.

In order to allow for many different OEMs to read each other's disks, the following standard is suggested: The information relating to the BPB for a particular piece of media is kept in the boot sector for the media. In particular, the format of the boot sector is:

	3 BYTE near JUMP to boot code
	8 BYTES OEM name and version
BPB IV	WORD bytes per sector
	BYTE sectors per allocation unit
	WORD reserved sectors
	BYTE number of FATs
	WORD number of root dir entries
	WORD number of sectors in logical image
IBPB	BYTE media descriptor
	WORD number of FAT sectors
	WORD sectors per track
	WORD number of heads
	WORD number of hidden sectors

The three words at the end (sectors per track, number of heads, and number of hidden sectors) are optional. They are intended to help the BIOS understand the media. Sectors per track may be redundant (could be calculated from total size of the disk). Number of heads is useful for supporting different multi-head drives which have the same storage capacity, but different numbers of surfaces. Number of hidden sectors may be used to support drive-partitioning schemes.

2.6.4 Media Descriptor Byte

The last two digits of the FAT ID byte are called the media descriptor byte. Currently, the media descriptor byte has been defined for a few media types, including 5-1/4" and 8" standard disks. For more information, refer to Section 3.6, "MS-DOS Standard Disk Formats." Although these media bytes map directly to FAT ID bytes (which are constrained to the 8 values F8-FF), media bytes can, in general, be any value in the range 0-FF.

2.6.5 READ OR WRITE

Command codes = 3,4,8,9, and 12

READ or WRITE - ES:BX (Including IOCTL) - >

13-BYTE Request Header
BYTE media descriptor from DPB
DWORD transfer address
WORD byte/sector count
WORD starting sector number (Ignored on character devices)

In addition to setting the status word, the driver must set the sector count to the actual number of sectors (or bytes) transferred. No error check is performed on an IOCTL I/O call. The driver **must** correctly set the return sector (byte) count to the actual number of bytes transferred.

THE FOLLOWING APPLIES TO BLOCK DEVICE DRIVERS:

Under certain circumstances the BIOS may be asked to perform a write operation of 64K bytes, which seems to be a "wrap around" of the transfer address in the BIOS I/O packet. This request arises due to an optimization added to the write code in MS-DOS. It will only manifest on user writes that are within a sector size of 64K bytes on files "growing" past the current EOF. **It is allowable for the BIOS to ignore the balance of the write that "wraps around" if it so chooses.** For example, a write of 10000H bytes worth of sectors with a transfer address of XXX:1 could ignore the last two bytes. A user program can never request an I/O of more than FFFFH bytes and cannot wrap around (even to 0) in the transfer segment. Therefore, in this case, the last two bytes can be ignored.

2.6.6 NON DESTRUCTIVE READ NO WAIT

Command code = 5

NON DESRUCTIVE READ NO WAIT - ES:BX - >

13-BYTE Request Header
BYTE read from device

If the character device returns busy bit = 0 (characters in buffer), then the next character that would be read is returned. This character is **not** removed from the input buffer (hence the term "Non Destructive Read"). Basically, this call allows MS-DOS to look ahead one input character.

2.6.7 STATUS

Command codes = 6 and 10

STATUS Calls - ES:BX - >

13-BYTE Request Header

All the driver must do is set the status word and the busy bit as follows:

> **For output on character devices:** If bit 9 is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request and a write request (if made) would start immediately.
> **For input on character devices with a buffer:** A return of 1 means, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the devices buffer and a read would return quickly. A return of 0 also indicates that the user has typed something. MS-DOS assumes that all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer should always return busy = 0 so that the DOS will not hang waiting for something to get into a buffer which doesn't exist.

2.6.8 FLUSH

Command codes = 7 and 11

FLUSH Calls - ES:BX - >

13-Byte Request Header

The FLUSH call tells the driver to flush (terminate) all pending requests. This call is used to flush the input queue on character devices.

2.7 THE CLOCK DEVICE

One of the most popular add-on boards is the real time clock board. To allow this board to be integrated into the system for TIME and DATE, there is a special device (determined by the attribute word), called the CLOCK device. The CLOCK device defines and performs functions like any other character device. Most functions will be: "set done bit, reset error bit, return." When a read or write to this device occurs, exactly 6 bytes are transferred. The first two bytes are a word, which is the count of days since 1-1-80. The third byte is minutes, the fourth, hours, the fifth, hundredths of seconds, and the sixth, seconds. Reading the CLOCK device gets the date and time; writing to it sets the date and time.

2.8 EXAMPLE DEVICE DRIVERS

The following examples illustrate a block device driver and a character device driver program.

2.8.1 Block Device Driver

```
;***************************** A BLOCK DEVICE ******************************

        TITLE 5 1/4" DISK DRIVER FOR SCP DISK-MASTER

;This driver is intended to drive up to four 5 1/4" drives
;hooked to the Seattle Computer Products DISK MASTER disk
;controller. All standard IBM PC formats are supported.

FALSE           EQU             0
TRUE            EQU             NOT FALSE

;The I/O port address of the DISK MASTER
DISK            EQU             0E0H
;DISK+0
;               1793            Command/Status
;DISK+1
;               1793            Track
;DISK+2
;               1793            Sector
;DISK+3
;               1793            Data
;DISK+4
;               Aux Command/Status
;DISK+5
;               Wait Sync

;Back side select bit
BACKBIT         EQU             04H
;5 1/4"                         select bit
SMALBIT         EQU             10H
;Double Density bit
DDBIT           EQU             08H

;Done bit in status register
DONEBIT         EQU             01H

; Use table below to select head step speed.
;Step times for 5" drives
; are double that shown in the table.
;
; Step value                    1771    1793
;
;               0               6ms     3ms
;               1               6ms     6ms
```

```
;               2               10ms    10ms
;               3               20ms    15ms
;
STPSPD          EQU             1

NUMERR          EQU             ERROUT-ERRIN

CR              EQU             0DH
LF              EQU             0AH
CODE            SEGMENT
ASSUME          CS:CODE,DS:NOTHING,ES:NOTHING,SS:NOTHING
;--------------------------------------------------------------
;
;               DEVICE          HEADER
;
DRVDEV          LABEL           WORD
                DW              -1,-1
                DW              0000    ;IBM format-compatible, Block
                DW              STRATEGY
                DW              DRV$IN
DRVMAX          DB              4

DRVTBL          LABEL           WORD
                DW              DRV$INIT
                DW              MEDIA$CHK
                DW              GET$BPB
                DW              CMDERR
                DW              DRV$READ
                DW              EXIT
                DW              EXIT
                DW              EXIT
                DW              DRV$WRIT
                DW              DRV$WRIT
                DW              EXIT
                DW              EXIT
                DW              EXIT

;--------------------------------------------------------------
;
;               STRATEGY
PTRSAV          DD              0

STRATP          PROC            FAR
STRATEGY:
                MOV             WORD PTR [PTRSAV],BX
                MOV             WORD PTR [PTRSAV+2],ES
                RET
STRATP          ENDP

;--------------------------------------------------------------
;
;               MAIN ENTRY
```

```
CMDLEN  =  0        ;LENGTH OF THIS COMMAND
UNIT    =  1        ;SUB UNIT SPECIFIER
CMDC    =  2        ;COMMAND CODE
STATUS  =  3        ;STATUS
MEDIA   = 13        ;MEDIA DESCRIPTOR
TRANS   = 14        ;TRANSFER ADDRESS
COUNT   = 18        ;COUNT OF BLOCKS OR CHARACTERS
START   = 20        ;FIRST BLOCK TO TRANSFER
DRV$IN:
        PUSH  SI
        PUSH  AX
        PUSH  CX
        PUSH  DX
        PUSH  DI
        PUSH  BP
        PUSH  DS
        PUSH  ES
        PUSH  BX

        LDS   BX,[PTRSAV]   ;GET POINTER TO I/O PACKET

        MOV   AL,BYTE PTR [BX].UNIT         ;AL = UNIT CODE
        MOV   AH,BYTE PTR [BX].MEDIA        ;AH = MEDIA DESCRIP
        MOV   CX,WORD PTR [BX].COUNT        ;CX = COUNT
        MOV   DX,WORD PTR [BX].START;       ;DX = START SECTOR
        PUSH  AX
        MOV   AL,BYTE PTR [BX].CMDC         ;Command code
        CMP   AL,11
        JA    CMDERRP                       ;Bad command
        CBW
        SHL   AX,1                          ;2 times command =
                                            ;word table index
        MOV   SI,OFFSET DRVTBL
        ADD   SI,AX                         ;Index into table
        POP   AX                            ;Get back media
                                            ;and unit

        LES   DI,DWORD PTR[BX].TRANS        ;ES:DI = TRANSFER
                                            ;ADDRESS

        PUSH  CS
        POP   DS
ASSUME  DS:CODE
        JMP   WORD PTR [SI]                 ;GO DO COMMAND
;---------------------------------------------------------------------------
;       EXIT - ALL ROUTINES RETURN THROUGH THIS PATH
;
ASSUME  DS:NOTHING
CMDERRP:
```

```
            POP     AX                              ;Clean stack
CMDERR:
            MOV     AL,3                            ;UNKNOWN COMMAND ERROR
            JMP     SHORT ERR$EXIT

ERR$CNT:    LDS     BX,[PTRSAV]
            SUB     WORD PTR [BX].COUNT,CX ;# OF SUCCESS. I/Os

ERR$EXIT:
;AL has error code
            MOV     AH,10000001B                    ;MARK ERROR RETURN
            JMP     SHORT ERR1

EXITP       PROC    FAR

EXIT:       MOV     AH,00000001B
ERR1:       LDS     BX, [PTRSAV]
            MOV     WORD PTR [BX].STATUS,AX ;MARK OPERATION COMPLETE

            POP     BX
            POP     ES
            POP     DS
            POP     BP
            POP     DI
            POP     DX
            POP     CX
            POP     AX
            POP     SI
            RET                                     ;RESTORE REGS AND RETURN
EXITP       ENDP

CURDRV      DB      -1

TRKTAB      DB      -1,-1,-1,-1

SECCNT      DW      0

DRVLIM      =       8                               ;Number of sectors on device
SECLIM      =       13                              ;MAXIMUM SECTOR
HDLIM       =       15                              ;MAXIMUM HEAD

;WARNING - preserve order of drive and curhd!

DRIVE       DB      0                               ;PHYSICAL DRIVE CODE
CURHD       DB      0                               ;CURRENT HEAD
CURSEC      DB      0                               ;CURRENT SECTOR
CURTRK      DW      0                               ;CURRENT TRACK

;
MEDIA$CHK:                                          ;Always indicates Don't know
ASSUME      DS:CODE
            TEST    AH,00000100B ;TEST IF MEDIA REMOVABLE
            JZ      MEDIA$EXT
```

```
            XOR   DI,DI                          ;SAY I DON'T KNOW
MEDIA$EXT:
            LDS   BX, [PTRSAV]
            MOV   WORD PTR [BX].TRANS,DI
            JMP   EXIT

BUILD$BPB:
ASSUME      DS:CODE
            MOV   AH,BYTE PTR ES: [DI]          ;GET FAT ID BYTE
            CALL  GETBP                         ;TRANSLATE
SETBPB:     LDS   BX,[PTRSAV]
            MOV   [BX].MEDIA,AH
            MOV   [BX].COUNT,DI
            MOV   [BX].COUNT+2,CS
            JMP   EXIT

BUILDBP:
ASSUME      DS:NOTHING
;AH is media byte on entry
;DI points to correct BPB on return
            PUSH  AX
            PUSH  CX
            PUSH  DX
            PUSH  BX
            MOV   CL,AH             ;SAVE MEDIA
            AND   CL,0F8H           ;NORMALIZE
            CMP   CL,0F8H ;COMPARE WITH GOOD MEDIA BYTE
            JZ    GOODID
            MOV   AH,0FEH           ;DEFAULT TO 8-SECTOR,
                                    ;SINGLE-SIDED
GOODID:
            MOV   AL,1              ;SET NUMBER OF FAT SECTORS
            MOV   BX,64*256+8       ;SET DIR ENTRIES AND SECTOR MAX
            MOV   CX,40*8           ;SET SIZE OF DRIVE
            MOV   DX,01*256+1       ;SET HEAD LIMIT & SEC/ALL UNIT
            MOV   DI,OFFSET DRVBPB
            TEST  AH,00000010B      ;TEST FOR 8 OR 9 SECTOR
            JNZ   HAS8              ;NZ = HAS 8 SECTORS
            INC   AL                ;INC NUMBER OF FAT SECTORS
            INC   BL                ;INC SECTOR MAX
            ADD   CX,40             ;INCREASE SIZE
HAS8:       TEST  AH,00000001B      ;TEST FOR 1 OR 2 HEADS
            JZ    HAS1              ;Z = 1 HEAD
            ADD   CX,CX             ;DOUBLE SIZE OF DISK
            MOV   BH,112            ;INCREASE # OF DIREC. ENTRIES
            INC   DH                ;INC SEC/ALL UNIT
            INC   DL                ;INC HEAD LIMIT
HAS1:       MOV   BYTE PTR [DI].2,DH
            MOV   BYTE PTR [DI].6,BH
            MOV   WORD PTR [DI].8,CX
            MOV   BYTE PTR [DI].10,AH
            MOV   BYTE PTR [DI].11,AL
            MOV   BYTE PTR [DI].13,BL
            MOV   BYTE PTR [DI].15,DL
            POP   BX
```

```
                POP     DC
                POP     CX
                POP     AX
                RET
;--------------------------------------------------------------------------------
;
;               DISK I/O HANDLERS
;
;ENTRY:
;               AL = DRIVE NUMBER (0-3)
;               AH = MEDIA DESCRIPTOR
;               CX = SECTOR COUNT
;               DX = FIRST SECTOR
;               DS = CS
;               ES:DI = TRANSFER ADDRESS
;EXIT:
;               IF SUCCESSFUL CARRY FLAG = 0
;                 ELSE CF = 1 AND AL CONTAINS (MS-DOS) ERROR CODE,
;                 CX # sectors NOT transferred
DRV$READ:
ASSUME          DS:CODE
                JCXZ            DSKOK
                CALL            SETUP
                JC              DSK$IO
                CALL            DISKRD
                JMP             SHORT DSK$IO

DRV$WRIT:
ASSUME          DS:CODE
                JCXZ            DSKOK
                CALL            SETUP
                JC              DSK$IO
                CALL            DISKWRT
ASSUME          DS:NOTHING
DSK$IO:         JNC             DSKOK
                JMP             ERR$CNT
DSKOK:          JMP             EXIT

SETUP:
ASSUME          DS:CODE
;Input same as above
;On output
; ES:DI = Trans addr
; DS:BX Points to BPB
; Carry set if error (AL is error code (MS-DOS))
; else
;               [DRIVE]         = Drive number (0-3)
;               [SECCNT]        = Sectors to transfer
;               [CURSEC]        = Sector number of start of I/O
;               [CURHD]         = Head number of start of I/O   ;SET
;               [CURTRK]        = Track # of start of I/O ;Seek performed
```

```
; All other registers destroyed
        XCHG  BX,DI                     ;ES:BX = TRANSFER ADDRESS
        CALL  GETBP                     ;DS:DI = PTR TO BPB
        MOV   SI,CX
        ADD   SI,DX
        CMP   SI,WORD PTR [DI].DRVLIM
                                        ;COMPARE AGAINST DRIVE MAX
        JBE   INRANGE
        MOV   AL,8
        STC
        RET

INRANGE:
        MOV   [DRIVE],AL
        MOV   [SECCNT],CX               ;SAVE SECTOR COUNT
        XCHG  AX,DX                     ;SET UP LOGICAL SECTOR
                                        ;FOR DIVIDE
        XOR   DX,DX
        DIV   WORD PTR [DI].SECLIM ;DIVIDE BY SEC PER TRACK
        INC   DL
        MOV   [CURSEC],DL               ;SAVE CURRENT SECTOR
        MOV   CX,WORD PTR [DI].HDLIM ;GET NUMBER OF HEADS
        XOR   DX,DX ;DIVIDE TRACKS BY HEADS PER CYLINDER
        DIV   CX
        MOV   [CURHD],DL                ;SAVE CURRENT HEAD
        MOV   [CURTRK],AX               ;SAVE CURRENT TRACK
SEEK:
        PUSH  BX                        ;Xaddr
        PUSH  DI                        ;BPB pointer
        CALL  CHKNEW                    ;Unload nead if change drives
        CALL  DRIVESEL
        MOV   BL,[DRIVE]
        XOR   BH,BH,                    ;BX drive index
        ADD   BX,OFFSET TRKTAB          ;Get current track
        MOV   AX, [CURTRK]
        MOV   DL,AL                     ;Save desired track
        XCHG  AL,DS:[BX]                ;Make desired track current
        OUT   DISK+1,AL                 ;Tell Controller current track
        CMP   AL,DL                     ;At correct track?
        JZ    SEEKRET                   ;Done if yes
        MOV   BH,2                      ;Seek retry count
        CMP   AL,-1                     ;Position Known?
        JNZ   NOHOME                    ;If not home head
TRYSK:
        CALL  HOME
        JC    SEEKERR
NOHOME:
        MOV   AL,DL
        OUT   DISK+3,AL                 ;Desired track
        MOV   AL,1CH+STPSPD             ;Seek
        CALL  DCOM
        AND   AL,98H                    ;Accept not rdy, seek, & CRC errors
        JZ    SEEKRET
        JS    SEEKERR                   ;No retries if not ready
```

```
            DEC     BH
            JNZ     TRYSK
SEEKERR:
            MOV     BL,[DRIVE]
            XOR     BH,BH                   ;BX drive index
            ADD     BX,OFFSET TRKTAB        ;Get current track
            MOV     BYTE PTR DS:[BX],-1     ;Make current track
                                            ;lunknown
            CALL    GETERRCD
            MOV     CX,[SECCNT]             ;Nothing transferred
            POP     BX                      ;BPB Pointer
            POP     DI                      ;Xaddr
            RET

SEEKRET:
            POP     BX                      ;BPB pointer
            POP     DI                      ;Xaddr
            CLC
            RET

;--------------------------------------------------------------------------------
;
;           READ
;

DISKRD:
ASSUME      DS:CODE
            MOV     CX,[SECCNT]
RDLP:
            CALL    PRESET
            PUSH    BX
            MOV     BL,10                   ;Retry count
            MOV     DX,DISK+3               ;Data port
RDAGN:
            MOV     AL,80H                  ;Read command
            CLI                             ;Disable for 1793
            OUT     DISK,AL                 ;Output read command
            MOV     BP,DI                   ;Save address for retry
            JMP     SHORT RLOOPENTRY
RLOOP:
            STOSB
RLOOPENTRY:
            IN      AL,DISK+5               ;Wait for DRQ or INTRQ
            SHR     AL,1
            IN      AL,DX                   ;Read data
            JNC     RLOOP
            STI                             ;Ints OK now
            CALL    GETSTAT
            AND     AL,9CH
            JZ      RDPOP                   ;Ok
            MOV     DI,BP                   ;Get back transfer
            DEC     BL
            JNZ     RDAGN
            CMP     AL,10H                  ;Record not found?
```

```
            JNZ     GOT-CODE                ;No
            MOV     AL,1                    ;Map it
GOT-CODE:
            CALL    GETERRCD
            POP     BX
            RET
RDPOP:
            POP     BX
            LOOP    RDLP
            CLC
            RET

;---------------------------------------------------------------------------
;
;           WRITE
;

DISKWRT:
ASSUME      DS:CODE
            MOV     CX,[SECCNT]
            MOV     SI,DI
            PUSH    ES
            POP     DS
ASSUME      DS:NOTHING
WRLP:
            CALL    PRESET
            PUSH    BX
            MOV     BL,10                   ;Retry count
            MOV     DX,DISK+3               ;Data port
WRAGN:
            MOV     AL,0A0H                 ;Write command
            CLI                             ;Disable for 1793
            OUT     DISK,AL                 ;Output write command
            MOV     BP,SI                   ;Save address for retry
WRLOOP:
            IN      AL,DISK+5
            SHR     AL,1
            LODSB                           ;Get data
            OUT     DX,AL                   ;Write data
            JNC     WRLOOP
            STI                             ;Ints OK now
            DEC     SI
            CALL    GETSTAT
            AND     AL,0FCH
            JZ      WRPOP                   ;Ok
            MOV     SI,BP                   ;Get back transfer
            DEC     BL
            JNZ     WRAGN
            CALL    GETERRCD
            POP     BX
            RET

WRPOP:
            POP     BX
```

```
        LOOP    WRLP
        CLC
        RET

PRESET:
ASSUME  DS:NOTHING
        MOV     AL, [CURSEC]
        CMP     AL,CS:[BX].SECLIM
        JBE     GOTSEC
        MOV     DH,[CURHD]
        INC     DH
        CMP     DH,CS:[BX].HDLIM
        JB      SETHEAD                 ;Select new head
        CALL    STEP                    ;Go on to next track
        XOR     DH,DH                   ;Select head zero
SETHEAD:
        MOV     [CURHD],DH
        CALL    DRIVESEL
        MOV     AL,1                    ;First sector
        MOV     [CURSEC],AL             ;Reset CURSEC
GOTSEC:
        OUT     DISK+2,AL               ;Tell controller which sector
        INC     [CURSEC]                ;We go on to next sector
        RET

STEP:
ASSUME  DS:NOTHING
        MOV     AL,58H+STPSPD           ;Step in w/ update, no verify
        CALL    DCOM
        PUSH    BX
        MOV     BL,[DRIVE]
        XOR     BH,BH                   ;BX drive index
        ADD     BX,OFFSET TRKTAB        ;Get current track
        INC     BYTE PTR CS:[BX]        ;Next track
        POP     BX
        RET

HOME:
ASSUME  DS:NOTHING
        MOV     BL,3
TRYHOM:
        MOV     AL,0CH+STPSPD           ;Restore with verify
        CALL    DCOM
        AND     AL,98H
        JZ      RET3
        JS      HOMERR                  ;No retries if not ready
        PUSH    AX                      ;Save real error code
        MOV     AL,58H+STPSPD           ;Step in w/ update no verify
        CALL    DCOM
        DEC     BL
        POP     AX                      ;Get back real error code
        JNZ     TRYHOM
HOMERR:
        STC
```

```
RET3:       RET

CHKNEW:
ASSUME      DS:NOTHING
            MOV     AL,[DRIVE]              ;Get disk drive number
            MOV     AH,AL
            XCHG    AL,[CURDRV]             ;Make new drive current.
            CMP     AL,AH                   ;Changing drives?
            JZ      RET1                    ;No
; If changing drives, unload head so the head load delay
;one-shot will fire again. Do it by seeking to the same
;track with the H bit reset.
;
            IN      AL,DISK+1               ;Get current track number
            OUT     DISK+3,AL               ;Make it the track to seek
            MOV     AL,10H                  ;Seek and unload head

DCOM:
ASSUME      DS:NOTHING
            OUT     DISK,AL
            PUSH    AX
            AAM                             ;Delay 10 microseconds
            POP     AX
GETSTAT:
            IN      AL,DISK+4
            TEST    AL,DONEBIT
            JZ      GETSTAT
            IN      AL,DISK
RET1:       RET

DRIVESEL:
ASSUME      DS:NOTHING
;Select the drive based on current info
;Only AL altered
            MOV     AL,[DRIVE]
            OR      AL,SMALBIT + DDBIT  ;5 1/4" IBM PC disks
            CMP     [CURHD],0
            JZ      GOTHEAD
            OR      AL,BACKBIT              ;Select side 1
GOTHEAD:
            OUT     DISK+4,AL               ;Select drive and side
            RET

GETERRCD:
ASSUME      DS:NOTHING
            PUSH    CX
            PUSH    ES
            PUSH    DI
            PUSH    CS
            POP     ES                      ;Make ES the local segment
            MOV     CS:[LSTERR],AL          ;Terminate list w/ error code
            MOV     CX,NUMERR               ;Number of error conditions
            MOV     DI,OFFSET ERRIN         ;Point to error conditions
            REPNE   SCASB
```

```
                MOV     AL,NUMERR-1[DI]         ;Get translation
                STC                             ;Flag error condition
                POP     DI
                POP     ES
                POP     CX
                RET                             ;and return

;*****************************************************************************
;               BPB FOR AN IBM FLOPPY DISK, VARIOUS PARAMETERS ARE
;               PATCHED BY GETBP TO REFLECT THE TYPE OF MEDIA
;               INSERTED
;               This is a nine sector single side BPB
DRVBPB:
                DW      512                     ;Physical sector size in bytes
                DB      1                       ;Sectors/allocation unit
                DW      1                       ;Reserved sectors for DOS
                DB      2                       ;# of allocation tables
                DW      64                      ;Number directory entries
                DW      9*40                    ;Number 512-byte sectors
                DB      11111100B               ;Media descriptor
                DW      2                       ;Number of FAT sectors
                DW      9                       ;Sector limit
                DW      1                       ;Head limit

INITAB          DW      DRVBPB                  ;Up to four units
                DW      DRVBPB
                DW      DRVBPB
                DW      DRVBPB

ERRIN:          ;DISK ERRORS RETURNED FROM THE 1793 CONTROLER
                DB      80H                     ;NO RESPONSE
                DB      40H                     ;Write protect
                DB      20H                     ;Write Fault
                DB      10H                     ;SEEK error
                DB      8                       ;CRC error
                DB      1                       ;Mapped from 10H
                                                ;(record not found) on READ
LSTERR          DB      0                       ;ALL OTHER ERRORS

ERROUT:         ;RETURNED ERROR CODES CORRESPONDING TO ABOVE
                DB      2                       ;NO RESPONSE
                DB      0                       ;WRITE ATTEMPT
                                                ;ON WRITE-PROTECT DISK
                DB      0AH                     ;WRITE FAULT
                DB      6                       ;SEEK FAILURE
                DB      4                       ;BAD CRC
                DB      8                       ;SECTOR NOT FOUND
                DB      12                      ;GENERAL ERROR

DRV$INIT:
;
; Determine number of physical drives by reading CONFIG.SYS
;
```

```
ASSUME  DS:CODE
        PUSH  DS
        LDS   SI,[PTRSAV]
ASSUME  DS:NOTHING
        LDS   SI,DWORD PTR [SI.COUNT]    ;DS:SI points to ;CONFIG.SYS
SCAN-LOOP:
        CALL  SCAN-SWITCH
        MOV   AL,CL
        OR    AL,AL
        JZ    SCAN4
        CMP   AL,"s"
        JZ    SCAN4

WERROR: POP   DS
ASSUME  DS:CODE
        MOV   DX,OFFSET ERRMSG2
WERROR2: MOV  AH,9
        INT   21H
        XOR   AX,AX
        PUSH  AX                         ;No units
        JMP   SHORT ABORT

BADNDRV:
        POP   DS
        MOV   DX,OFFSET ERRMSG1
        JMP   WERROR2

SCAN4:
ASSUME  DS:NOTHING
;BX is number of floppies
        OR    BX,BX
        JZ    BADNDRV                    ;User error
        CMP   BX,4
        JA    BADNDRV                    ;User error
        POP   DS
ASSUME  DS:CODE
        PUSH  BX                         ;Save unit count
ABORT:  LDS   BX,[PTRSAV]
ASSUME  DS:NOTHING
        POP   AX
        MOV   BYTE PTR [BX].MEDIA,AL     ;Unit count
        MOV   [DRVMAX],AL
        MOV   WORD PTR [BX].TRANS,OFFSET DRV$INIT ;SET
                                         ;BREAK ADDRESS
        MOV   [BX].TRANS+2,CS
        MOV   WORD PTR [BX].COUNT,OFFSET INITAB
                                         ;SET POINTER TO BPB ARRAY
        MOV   [BX].COUNT+2,CS
        JMP   EXIT
;
; PUT SWITCH IN CL, VALUE IN BX
;
SCAN-SWITCH:
        XOR   BX,BX
```

```
            MOV   CX,BX
            LODSB
            CMP   AL,10
            JZ    NUMRET
            CMP   AL,"-"
            JZ    GOT-SWITCH
            CMP   AL,"/"
            JNZ   SCAN-SWITCH
GOT-SWITCH:
            CMP   BYTE PTR [SI+1],":"
            JNZ   TERROR
            LODSB
            OR    AL,20H                ; CONVERT TO LOWER CASE
            MOV   CL,AL                 ; GET SWITCH
            LODSB                       ; SKIP ":"
;
; GET NUMBER POINTED TO BY [SI]
;
; WIPES OUT AX,DX ONLY                  BX RETURNS NUMBER
;
GETNUM1:    LODSB
            SUB   AL,"0"
            JB    CHKRET
            CMP   AL,9
            JA    CHKRET
            CBW
            XCHG  AX,BX
            MOV   DX,10
            MUL   DX
            ADD   BX,AX
            JMP   GETNUM1

CHKRET:     ADD   AL,"0"
            CMP   AL," "
            JBE   NUMRET
            CMP   AL,"-"
            JZ    NUMRET
            CMP   AL,"/"
            JZ    NUMRET
TERROR:
            POP   DS                    ; GET RID OF RETURN ADDRESS
            JMP   WERROR
NUMRET:     DEC   SI
            RET

ERRMSG1     DB    "SMLDRV: Bad number of drives",13,10,"$"
ERRMSG2     DB    "SMLDRV: Invalid parameter",13,10,"$"
CODE        ENDS
            END
```

2.8.2 Character Device Driver

The following program illustrates a character device driver program.

```
;****************** A CHARACTER DEVICE ******************

TITLE     VT52 CONSOLE FOR 2.0      (IBM)

;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;
;          IBM ADDRESSES FOR I/O
;
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

          CR=13                ;CARRIAGE RETURN
          BACKSP=8             ;BACKSPACE
          ESC=1BH
          BRKADR=6CH           ;006C BREAK VECTOR ADDRESS
          ASNMAX=200           ;SIZE OF KEY ASSIGNMENT BUFFER

CODE      SEGMENT BYTE

     ASSUME CS:CODE,DS:NOTHING,ES:NOTHING
;-----------------------------------------------------------------------------
;
;          C O N - CONSOLE DEVICE DRIVER
;
CONDEV:                                    ;HEADER FOR DEVICE "CON"
          DW    -1,-1
          DW    1000000000010011B          ;CON IN AND CON OUT
          DW    STRATEGY
          DW    ENTRY
          DB    'CON     '

;-----------------------------------------------------------------------------
;
;          COMMAND JUMP TABLES
CONTBL:
          DW    CON$INIT
          DW    EXIT
          DW    EXIT
          DW    CMDERR
          DW    CON$READ
          DW    CON$RDND
          DW    EXIT
          DW    CON$FLSH
          DW    CON$WRIT
          DW    CON$WRIT
          DW    EXIT
          DW    EXIT

CMDTABL   DB    'A'
```

```
        DW      CUU             ;cursor up
        DB      "B"
        DW      CUD             ;cursor down
        DB      "C"
        DW      CUF             ;cursor forward
        DB      "D"
        DW      CUB             ;cursor back
        DB      "H"
        DW      CUH             ;cursor position
        DB      "J"
        DW      ED              ;erase display
        DB      "K"
        DW      EL              ;erase line
        DB      "Y"
        DW      CUP             ;cursor position
        DB      "j"
        DW      PSCP            ;save cursor position
        DB      "k"
        DW      PRCP            ;restore cursor position
        DB      "y"
        DW      RM              ;reset mode
        DB      "x"
        DW      SM              ;set mode
        DB      00

PAGE
;--------------------------------------------------------
;
;       Device entry pont
;
CMDLEN  =       0               ;LENGTH OF THIS COMMAND
UNIT    =       1               ;SUB UNIT SPECIFIER
CMD     =       2               ;COMMAND CODE
STATUS  =       3               ;STATUS
MEDIA   =       13              ;MEDIA DESCRIPTOR
TRANS   =       14              ;TRANSFER ADDRESS
COUNT   =       18              ;COUNT OF BLOCKS OR CHARACTERS
START   =       20              ;FIRST BLOCK TO TRANSFER

PTRSAV  DD      0
STRATP  PROC    FAR

STRATEGY:
        MOV     WORD PTR CS:[PTRSAV],BX
        MOV     WORD PTR CS:[PTRSAV+2],ES
        RET

STRATP  ENDP

ENTRY:
        PUSH    SI
        PUSH    AX
        PUSH    CX
        PUSH    DX
```

```
        PUSH    DI
        PUSH    BP
        PUSH    DS
        PUSH    ES
        PUSH    BX

        LDS     BX,CS:[PTRSAV] ;GET POINTER TO I/O PACKET
        MOV     CX,WORD PTR DS:[BX].COUNT   ;CX = COUNT

        MOV     AL,BYTE PTR DS:[BX].CMD
        CBW
        MOV     SI,OFFSET CONTBL
        ADD     SI,AX
        ADD     SI,AX
        CMP     AL,11
        JA      CMDERR

        LES     DI,DWORD PTR DS:[BX].TRANS

        PUSH    CS
        POP     DS
        ASSUME  DS:CODE

        JMP     WORD PTR [SI]                       ;GO DO COMMAND

PAGE
;===============================================================================
;=
;=      SUBROUTINES SHARED BY MULTIPLE DEVICES
;=
;===============================================================================
;-------------------------------------------------------------------------------
;
;       EXIT - ALL ROUTINES RETURN THROUGH THIS PATH
;
BUS$EXIT:                                       ;DEVICE BUSY EXIT
        MOV     AH,00000011B
        JMP     SHORT ERR1

CMDERR:
        MOV     AL,3                            ;UNKNOWN COMMAND ERROR

ERR$EXIT:
        MOV     AH,10000001B                    ;MARK ERROR RETURN
        JMP     SHORT ERR1

EXITP   PROC    FAR

EXIT:   MOV     AH,00000001B
ERR1:   LDS     BX,CS:[PTRSAV]
        MOV     WORD PTR [BX].STATUS,AX         ;MARK
                                                ;OPERATION COMPLETE
```

```
        POP     BX
        POP     ES
        POP     DS
        POP     BP
        POP     DI
        POP     DX
        POP     CX
        POP     AX
        POP     SI
        RET                             ;RESTORE REGS AND RETURN
EXITP   ENDP
;--------------------------------------------------------------------------------
;
;       BREAK KEY HANDLING
;
BREAK:
        MOV     CS:ALTAH,3              ;INDICATE BREAK KEY SET
INTRET: IRET

PAGE
;
;       WARNING - Variables are very order dependent,
                  so be careful when adding new ones!
;
WRAP    DB      0                       ; 0 = WRAP, 1 = NO WRAP
STATE   DW      S1
MODE    DB      3
MAXCOL  DB      79
COL     DB      0
ROW     DB      0
SAVCR   DW      0
ALTAH   DB      0                       ;Special key handling

;--------------------------------------------------------------------------------
;
;       CHROUT - WRITE OUT CHAR IN AL USING CURRENT ATTRIBUTE
;
ATTRW   LABEL   WORD
ATTR    DB      00000111B               ;CHARACTER ATTRIBUTE
BPAGE   DB      0                       ;BASE PAGE
base    dw      0b800h

chrout: cmp     al,13
        jnz     trylf
        mov     [col],0
        jmp     short setit

trylf:  cmp     al,10
        jz      lf
        cmp     al,7
        jnz     tryback
torom:
        mov     bx,[attrw]
        and     bl,7
        mov     ah,14
```

```
        int     10h
ret5:   ret
tryback:
        cmp     al,8
        jnz     outchr
        cmp     [col],0
        jz      ret5
        dec     [col]
        jmp     short setit

outchr:
        mov     bx,[attrw]
        mov     cx,1
        mov     ah,9
        int     10h
        inc     [col]
        mov     al,[col]
        cmp     al,[maxcol]
        jbe     setit
        cmp     [wrap],0
        jz      outchr1
        dec     [col]
        ret
outchr1:
        mov     [col],0
lf:     inc     [row]
        cmp     [row],24
        jb      setit
        mov     [row],23
        call    scroll

setit:  mov     dh,row
        mov     dl,col
        xor     bh,bh
        mov     ah,2
        int     10h
        ret

scroll: call    getmod
        cmp     al,2
        jz      myscroll
        cmp     al,3
        jz      myscroll
        mov     al,10
        jmp     torom
myscroll:
        mov     bh,[attr]
        mov     bl," "
        mov     bp,80
        mov     ax,[base]
        mov     es,ax
        mov     ds,ax
        xor     di,di
        mov     si,160
```

```
                mov     cx,23*80
                cld
                cmp     ax,0b800h
                jz      colorcard

                rep     movsw
                mov     ax,bx
                mov     cx,bp
                rep     stosw
sret:           push    cs
                pop     ds
                ret

colorcard:
                mov     dx,3dah
wait2:          in      al,dx
                test    al,8
                jz      wait2
                mov     al,25h
                mov     dx,3d8h
                out     dx,al                   ;turn off video
                rep     movsw
                mov     ax,bx
                mov     cx,bp
                rep     stosw
                mov     al,29h
                mov     dx,3d8h
                out     dx,al                   ;turn on video
                jmp     sret

GETMOD:         MOV     AH,15
                INT     16                      ;get column information
                MOV     BPAGE, BH
                DEC     AH
                MOV     WORD PTR MODE,AX
                RET
;-----------------------------------------------------------------------------
;
;               CONSOLE READ ROUTINE
;
CON$READ:
                JCXZ    CON$EXIT
CON$LOOP:
                PUSH    CX                      ;SAVE COUNT
                CALL    CHRIN                   ;GET CHAR IN AL
                POP     CX
                STOSB                           ;STORE CHAR AT ES:DI
                LOOP    CON$LOOP
CON$EXIT:
                JMP     EXIT
;-----------------------------------------------------------------------------
;
;               INPUT SINGLE CHAR INTO AL
;
CHRIN:          XOR     AX,AX
```

```
            XCHG  AL,ALTAH                ;GET CHARACTER & ZERO ALTAH
            OR    AL,AL
            JNZ   KEYRET

INAGN:      XOR   AH,AH
            INT   22
ALT10:
            OR    AX,AX                   ;Check for non-key after BREAK
            JZ    INAGN
            OR    AL,AL                   ;SPECIAL CASE?
            JNZ   KEYRET
            MOV   ALTAH,AH                  ;STORE SPECIAL KEY

KEYRET:     RET
;--------------------------------------------------------------------------------
;
;           KEYBOARD NON DESTRUCTIVE READ, NO WAIT
;
CON$RDND:
            MOV   AL,[ALTAH]
            OR    AL,AL
            JNZ   RDEXIT

RD1:        MOV   AH,1
            INT   22
            JZ    CONBUS
            OR    AX,AX
            JNZ   RDEXIT
            MOV   AH,0
            INT   22
            JMP   CON$RDND

RDEXIT:     LDS   BX,[PTRSAV]
            MOV   [BX].MEDIA,AL
EXVEC:      JMP   EXIT
CONBUS:     JMP   BUS$EXIT
;--------------------------------------------------------------------------------
;
;           KEYBOARD FLUSH ROUTINE
;
CON$FLSH:
            MOV   [ALTAH],0               ;Clear out holding buffer

            PUSH  DS
            XOR   BP,BP
            MOV   DS,BP                   ;Select segment 0
            MOV   DS:BYTE PTR 41AH,1EH    ;Reset KB queue head
                                          ;pointer
            MOV   DS:BYTE PTR 41CH,1EH    ;Reset tail pointer
            POP   DS
            JMP   EXVEC
;--------------------------------------------------------------------------------
;
;           CONSOLE WRITE ROUTINE
;
CON$WRIT:
```

```
            JCXZ   EXVEC
            PUSH   CX
            MOV    AH,3                    ;SET CURRENT CURSOR POSITION
            XOR    BX,BX
            INT    16
            MOV    WORD PTR [COL],DX
            POP    CX

CON$LP:     MOV    AL,ES:[DI]              ;GET CHAR
            INC    DI
            CALL   OUTC                    ;OUTPUT CHAR
            LOOP   CON$LP                  ;REPEAT UNTIL ALL THROUGH
            JMP    EXVEC

COUT:       STI
            PUSH   DS
            PUSH   CS
            POP    DS
            CALL   OUTC
            POP    DS
            IRET

OUTC:       PUSH   AX
            PUSH   CX
            PUSH   DX
            PUSH   SI
            PUSH   DI
            PUSH   ES
            PUSH   BP
            CALL   VIDEO
            POP    BP
            POP    ES
            POP    DI
            POP    SI
            POP    DX
            POP    CX
            POP    AX
            RET

;-----------------------------------------------------------------------------
;
;           OUTPUT SINGLE CHAR IN AL TO VIDEO DEVICE
;
VIDEO:      MOV    SI,OFFSET STATE
            JMP    [SI]

S1:         CMP    AL,ESC                  ;ESCAPE SEQUENCE?
            JNZ    S1B
            MOV    WORD PTR [SI],OFFSET S2
            RET

S1B:        CALL   CHROUT
S1A:        MOV    WORD PTR [STATE],OFFSET S1
            RET
```

```
S2:     PUSH  AX
        CALL  GETMOD
        POP   AX
        MOV   BX,OFFSET CMDTABL-3
S7A:    ADD   BX,3
        CMP   BYTE PTR [BX],0
        JZ    S1A
        CMP   BYTE PTR [BX],AL
        JNZ   S7A
        JMP   WORD PTR [BX+1]

MOVCUR: CMP   BYTE PTR [BX],AH
        JZ    SETCUR
        ADD   BYTE PTR [BX],AL
SETCUR: MOV   DX,WORD PTR COL
        XOR   BX,BX
        MOV   AH,2
        INT   16
        JMP   S1A

CUP:    MOV   WORD PTR [SI],OFFSET CUP1
        RET
CUP1:   SUB   AL,32
        MOV   BYTE PTR [ROW],AL
        MOV   WORD PTR [SI],OFFSET CUP2
        RET
CUP2:   SUB   AL,32
        MOV   BYTE PTR [COL],AL
        JMP   SETCUR

SM:     MOV   WORD PTR [SI],OFFSET S1A
        RET

CUH:    MOV   WORD PTR COL,0
        JMP   SETCUR

CUF:    MOV   AH,MAXCOL
        MOV   AL,1
CUF1:   MOV   BX,OFFSET COL
        JMP   MOVCUR

CUB:    MOV   AX,00FFH
        JMP   CUF1

CUU:    MOV   AX,00FFH
CUU1:   MOV   BX,OFFSET ROW
        JMP   MOVCUR

CUD:    MOV   AX,23*256+1
        JMP   CUU1
```

```
PSCP:   MOV   AX,WORD PTR COL
        MOV   SAVCR,AX
        JMP   SETCUR

PRCP:   MOV   AX,SAVCR
        MOV   WORD PTR COL,AX
        JMP   SETCUR

ED:     CMP   BYTE PTR [ROW],24
        JAE   EL1

        MOV   CX,WORD PTR COL
        MOV   DH,24
        JMP   ERASE

EL1:    MOV   BYTE PTR [COL],0
EL:     MOV   CX,WORD PTR [COL]
EL2     MOV   DH,CH
ERASE:  MOV   DL,MAXCOL
        MOV   BH,ATTR
        MOV   AX,0600H
        INT   16
ED3:    JMP   SETCUR

RM:     MOV   WORD PTR [SI],OFFSET RM1
        RET
RM1:    XOR   CX,CX
        MOV   CH,24
        JMP   EL2

CON$INIT:
        int     11h
        and     al,00110000b
        cmp     al,00110000b
        jnz     iscolor
        mov     [base],0b000h           ;look for bw card
iscolor:
        cmp     al,00010000b            ;look for 40 col mode
        ja      setbrk
        mov     [mode],0
        mov     [maxcol],39

setbrk:
        XOR   BX,BX
        MOV   DS,BX
        MOV   BX,BRKADR
        MOV   WORD PTR [BX],OFFSET BREAK
        MOV   WORD PTR [BX+2],CS

        MOV   BX,29H*4
        MOV   WORD PTR [BX],OFFSET COUT
        MOV   WORD PTR [BX+2],CS
```

```
        LDS     BX,CS:[PTRSAV]
        MOV     WORD PTR [BX].TRANS,OFFSET CON$INIT
                                                ;SET BREAK ADDRESS
        MOV     [BX].TRANS+2,CS
        JMP     EXIT

CODE    ENDS
        END
```

CHAPTER 3
MS-DOS TECHNICAL INFORMATION

3.1 MS-DOS INITIALIZATION

MS-DOS initialization consists of several steps. Typically, a ROM (Read Only Memory) bootstrap obtains control, and then reads the boot sector off the disk. The boot sector then reads the following files:

IO.SYS
MSDOS.SYS

Once these files are read, the boot process begins.

3.2 THE COMMAND PROCESSOR

The Command processor supplied with MS-DOS (file COMMAND.-COM.) consists of 3 parts:

1. **A resident part** resides in memory immediately following MSDOS.SYS and its data area. This part contains routines to process Interrupts 23H (CONTROL-C Exit Address), and 24H (Fatal Error Abort Address), as well as a routine to reload the transient part, if needed. All standard MS-DOS error handling is done within this part of COMMAND.-COM. This includes displaying error messages and processing the Abort, Retry, or Ignore messages.
2. **An initialization part** follows the resident part. During start-up, the initialization part is given control; in contains the AUTOEXEC file processor setup routine. The initialization part determines the segment address at which programs can be loaded. It is overlaid by the first program COMMAND.-COM loads because it is no longer needed.

3. **A transient part** is loaded at the high end of memory. This part contains all of the internal command processors and the batch file processor.
   The transient part of the command processor produces the system prompt (such as A >), reads the command from keyboard (or batch file) and causes it to be executed. For external commands, this part builds a command line and issues the EXEC system call (Function Request 4BH) to load and transfer control to the program.

3.3 MS-DOS DISK ALLOCATION

The MS-DOS area is formatted as follows:

Reserved area - variable size
First copy of file allocation table - variable size
Second copy of file allocation table - variable size (optional)
Additional copies of file allocation table - variable size (optional)
Root directory - variable size
File data area

Allocation of space for a file in the data area is not pre-allocated. The space is allocated one cluster at a time. A cluster consists of one or more consecutive sectors; all of the clusters for a file are "chained" together in the File Allocation Table (FAT). (Refer to Section 3.5, "File Allocation Table.") There is usually a second copy of the FAT kept, for consistency. Should the disk develop a bad sector in the middle of the first FAT, the second can be used. This avoids loss of data due to an unusable disk.

3.4 MS-DOS DISK DIRECTORY

FORMAT builds the root directory for all disks. Its location on disk and the maximum number of entries are dependent on the media. Since directories other than the root directory are regarded as files by MS-DOS, there is no limit to the number of files they may contain. All directory entries are 32 bytes in length, and are in the following format (note that byte offsets are in hexadecimal):

0-7 Filename. Eight characters, left aligned and padded, if necessary, with blanks. The first byte of this field indicates the file status as follows:

- 00H The directory entry has never been used. This is used to limit the length of directory searches, for performance reasons.
- 2EH The entry is for a directory. If the second byte is also 2EH, then the cluster field contains the cluster number of this directory's parent directory (0000H if the parent directory is the root directory). Otherwise, bytes 01H through 0AH are all spaces, and the cluster field contains the cluster number of this directory.
- E5H The file was used, but it has been erased.

Any other character is the first character of a filename.

8-0A Filename extension.

0B File attribute. The attribute byte is mapped as follows (values are in hexadecimal):

- 01 File is marked read-only. An attempt to open the file for writing using the Open File system call (Function Request 3DH) results in an error code being returned. This value can be used along with other values below. Attempts to delete the file with the Delete File system call (13H) or Delete a Directory Entry (41H) will also fail.
- 02 Hidden file. The file is excluded from normal directory searches.
- 04 System file. The file is excluded from normal directory searches.
- 08 The entry contains the volume label in the first 11 bytes. The entry contains no other usable information (except date and time of creation), and may exist only in the root directory.

10 The entry defines a sub-directory, and is excluded from normal directory searches.

20 Archive bit. The bit is set to "on" whenever the file has been written to and closed.

Note: The system files (IO.SYS and MSDOS.SYS) are marked as read-only, hidden, and system files. Files can be marked hidden when they are created. Also, the read-only, hidden, system, and archive attributes may be changed through the Change Attributes system call (Function Request 43H).

0C-15 Reserved.

16-17 Time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

Offset 17H

H	H	H	H	H	M	M	M
7				3	2		

Offset 16H

M	M	M	S	S	S	S	S
		5	4				0

where:

H is the binary number of hours (0-23)
M is the binary number of minutes (0-59)
S is the binary number of two-second increments

18-19 Date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

Offset 19H

Y	Y	Y	Y	Y	Y	Y	M
7						1	0

Offset 18 H

M	M	M	D	D	D	D	D
		5	4				0

where:

Y is 0-119 (1980-2099)
M is 1-12
D is 1-31

1A-1B Starting cluster; the cluster number of the first cluster in the file.

Note that the first cluster for data space on all disks is cluster 002.

The cluster number is stored with the least significant byte first.

NOTE

Refer to Section 3.5.1, "How to Use the File Allocation Table," for details about converting cluster numbers to logical sector numbers.

1C-1F File size in bytes. The first word of this four-byte field is the low-order part of the size.

3.5 FILE ALLOCATION TABLE (FAT)

The following information is included for system programmers who wish to write installable device drivers. This section explains how MS-DOS uses the File Allocation Table to convert the clusters of a file to logical sector numbers. The driver is then responsible for locating the logical sector on disk. Programs must use the MS-DOS file management function calls for accessing files; programs that access the FAT are not guaranteed to be upwardly-compatible with future releases of MS-DOS.

The File Allocation Table is an array of 12-bit entries (1.5 bytes) for each cluster on the disk. The first two FAT entries map a portion of the directory; these FAT entries indicate the size and format of the disk.

The second and third bytes currently always contain FFH.

The third FAT entry, which starts at byte offset 4, begins the mapping of the data area (cluster 002). Files in the data area are not always written sequentially on the disk. The data area is allocated one cluster at a time, skipping over clusters already allocated. The first free cluster found will be the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by erasing files can be allocated for new files.

Each FAT entry contains three hexadecimal characters:

000	If the cluster is unused and available.
FF7	The cluster has a bad sector in it. MS-DOS will not allocate such a cluster. CHKDSK counts the number of bad clusters for its report. These bad clusters are not part of any allocation chain.
FF8-FFF	Indicates the last cluster of a file.
XXX	Any other characters that are the cluster number of the next cluster in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.

The File Allocation Table always begins on the first section after the reserved sectors. If the FAT is larger than one sector, the sectors are continguous. Two copies of the FAT are usually written for data integrity. The FAT is read into one of the MS-DOS buffers whenever needed (open, read, write, etc.). For performance reasons, this buffer is given a high priority to keep it in memory as long as possible.

3.5.1 How To Use The File Allocation Table

Use the directory entry to find the starting cluster of the file. Next, to locate each subsequent cluster of the file:

1. Multiply the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).
2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.
3. Use a MOV instruction to move the word at the calculated FAT offset into a register.
4. If the last cluster used was an even number, keep the low-order 12 bits of the register by ANDing it with FFF; otherwise, keep the high-order 12 bits by shifting the register right 4 bits with a SHR instruction.
5. If the resultant 12 bits are FF8H-FFFH, the file contains no more clusters. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster to a logical sector number (relative sector, such as that used by Interrupts 25H and 26H and by DEBUG):

1. Subtract 2 from the cluster number.
2. Multiply the result by the number of sectors per cluster.
3. Add to this result the logical sector number of the beginning of the data area.

3.6 MS-DOS STANDARD DISK FORMATS

On an MS-DOS disk, the clusters are arranged on disk to minimize head movement for multi-sided media. All of the space on a track (or cylinder) is allocated before moving on to the next track. This is accomplished by using the sequential sectors on the lowest-numbered head, then all the sectors on the next head, and so on until all sectors on all heads of the track are used. The next sector to be used will be sector 1 on head 0 of the next track.
For disks, the following table can be used:

# Sides	Sectors/ Track	FAT size Sectors	Dir Sectors	Dir Entries	Sectors/ Cluster
1	8	1	4	64	1
2	8	1	7	112	2
1	9	2	4	64	1
2	9	2	7	112	2

Figure 4. 5-¼" Disk Format

The first byte of the FAT can sometimes be used to determine the format of the disk. The following 5-¼" formats have been defined for the IBM Personal Computer, based on values of the first byte of the FAT. The formats in Table 3.1 are considered to be the standard disk formats for MS-DOS.

Table 3.1 MS-DOS Standard Disk Formats

	5-¼	5-¼	5-¼	5-¼	8	8	8
No. sides	1	1	2	2	1	1	2
Tracks/side	40	40	40	40	77	77	77
Bytes/sector	512	512	512	512	128	128	1024
Sectors/track	8	9	8	9	26	26	8
Sectors/allocation unit	1	1	2	2	4	4	1
Reserved sectors	1	1	1	1	1	4	1
No. FATS	2	2	2	2	2	2	2
Root directory entries	64	64	112	112	68	68	192
No. sectors	320	360	640	720	2002	2002	616
Media Descriptor Byte	FE	FC	FF	FD	FE*	FD	FE*
Sectors for 1 FAT	1	2	1	2	6	6	2

* The two media descriptor bytes that are the same for 8" disks (FEH) is not a misprint. To establish whether a disk is single- or double-density, a read of a single-density address mark should be made. If an error occurs, the media is double-density.

CHAPTER 4
MS-DOS CONTROL BLOCKS AND WORK AREAS

4.1 TYPICAL MS-DOS MEMORY MAP

0000:0000	Interrupt vector table
XXXX:0000	IO.SYS - MS-DOS interface to hardware
XXXX:0000	MSDOS.SYS - MS-DOS interrupt handlers, service routines (Interrupt 21H functions) MS-DOS buffers, control areas, and installed device drivers
XXXX:0000	Resident part of COMMAND.COM - Interrupt handlers for Interrupts 22H (Terminate Address), 23H (CONTROL-C Exit Address), 24H (Fatal Error Abort Address) and code to reload the transient part
XXXX:0000	External command or utility - (.COM or .EXE file)
XXXX:0000	User stack for .COM files (256 bytes)
XXXX:0000	Transient part of COMMAND.COM - Command interpreter, internal commands, batch processor

1. Memory map addresses are in segment:offset format. For example, 0090:0000 is absolute address 0900H.
2. User memory is allocated from the lowest end of available memory that will meet the allocation request.

4.2 MS-DOS PROGRAM SEGMENT

When an external command is typed, or when you execute a program through the EXEC system call, MS-DOS determines the lowest available free memory address to use as the start of the program. This area is called the Program Segment.
The first 256 bytes of the Program Segment are set up by the EXEC system call for the program being loaded into memory. The program is then loaded following this block. An .EXE file with minalloc and maxalloc both set to zero is loaded as high as possible.
At offset 0 within the Program Segment, MS-DOS builds the Program Segment Prefix control block. The program returns form EXEC by one of four methods:

1. A long jump to offset 0 in the Program Segment Prefix
2. By issuing an INT 20H with CS:0 pointing at the PSP
3. By issuing an INT 21H with register AH = 0 with CS:0 pointing at the PSP, or 4CH and no restrictions on CS
4. By a long call to location 50H in the Program Segment Prefix with AH = 0 or Function Request 4CH

NOTE

It is the responsibility of all programs to ensure that the CS register contains the segment address of the Program Segment Prefix when terminating via any of these methods, except Function Request 4CH. For this reason, using Function Request 4CH is the preferred method.

All four methods result in transferring control to the program that issued the EXEC. During this returning process, Interrupts 22H, 23H, and 24H (Terminate Address, CONTROL-C Exit Address, and Fatal Error Abort Address) addresses are restored from the values saved in the Program Segment Prefix of the terminating program. Control is then given to the terminate address. If this is a program returning to COMMAND.COM, control transfers to its resident portion. If a batch file was in process, it is continued; otherwise, COMMAND.COM performs a checksum on the transient part, reloads it if necessary, then issues the system prompt and waits for you to type the next command.
When a program receives control, the following conditions are in effect:

For all programs:

The segment address of the passed environment is contained at offset 2CH in the Program Segment Prefix.

The environment is a series of ASCII strings (totaling less than 32K) in the form:

NAME = parameter

Each string is terminated by a byte of zeros, and the set of strings is terminated by another byte of zeros. The environment built by the command processor contains at least a COMSPEC = string (the parameters on COMSPEC define the path used by MS-DOS to locate COMMAND.COM on disk). The last PATH and PROMPT commands issued will also be in the environment, along with any environment strings defined with the MS-DOS SET command.

The environment that is passed is a copy of the invoking process environment. If your application uses a "keep process" concept, you should be aware that the copy of the environment passed to you is static. That is, it will not change even if subsequent SET, PATH, or PROMPT commands are issued.

Offset 50H in the Program Segment Prefix contains code to call the MS-DOS function dispatcher. By placing the desired function request number in AH a program can issue a far call to offset 50H to invoke an MS-DOS function, rather than issuing an Interrupt 21H. Since this is a **call** and not an interrupt, MS-DOS may place any code appropriate to making a system call at this position. This makes the process of calling the system portable.

The Disk Transfer Address (DTA) is set to 80H (default DTA in the Program Segment Prefix).

File control blocks at 5CH and 6CH are formatted from the first two parameters typed when the command was entered. If either parameter contained a pathname, then the corresponding FCB contains only the valid drive number. The filename field will not be valid.

An unformatted parameter area at 81H contains all the characters typed after the command (including leading and imbedded delimiters), with the byte at 80H set to the number of characters. If the <, >, or parameters were typed on the command line, they (and the filenames associated with them) will not appear in this area; redirection of standard input and output is transparent to applications.

Offset 6 (one word) contains the number of bytes available in the segment.

Register AX indicates whether or not the drive specifiers (entered with the first two parameters) are valid, as follows:

> Al = FF if the first parameter contained an invalid drive specifier (otherwise AL = 00)
> AH = FF if the second parameter contained an invalid drive specifier (otherwise AH = 00)

Offset 2 (one word) contains the segment address of the first byte of unavailable memory. Programs must not modify addresses beyond this point unless they were obtained by allocating memory via the Allocate Memory system call (Function Request 48H).

For Executable (EXE) programs:

DS and ES registers are set to point to the Program Segment Prefix.

CS,IP,SS, and SP registers are set to the values passed by MS-LINK.

For Executable (.COM) programs:

All four segment registers contain the segment address of the initial allocation block that starts with the Program Segment Prefix control block.

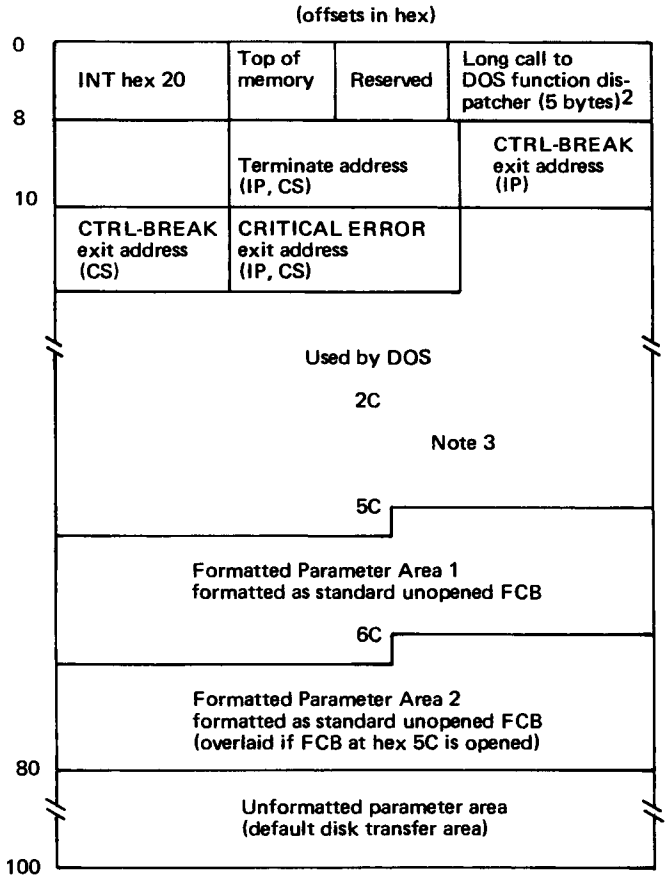
All of user memory is allocated to the program. If the program invokes another program through Function Request 4BH, it must first free some memory through the Set Block (4AH) function call, to provide space for the program being executed.

The Instruction Pointer (IP) is set to 100H.

The Stack Pointer register is set to the end of the program's segment. The segment size at offset 6 is reduced by 100H to allow for a stack of that size.

A word of zeros is placed on top of the stack. This is to allow a user program to exit to COMMAND.COM by doing a RET instruction last. This assumes, however, that the user has maintained his stack and code segments.

Figure 5. illustrates the format of the Program Segment Prefix. All offsets are in hexadecimal.



1. First segment of available memory is in segment (paragraph) form (for example, hex 1000 would represent 64K).
2. The word at offset 6 contains the number of bytes available in the segment.
3. Offset hex 2C contains the segment address of the environment.

Figure 5 Program Segment Prefix

IMPORTANT

Programs must not alter any part of the Program Segment Prefix below offset 5CH.

CHAPTER 5
EXE FILE STRUCTURE AND LOADING

NOTE

This chapter describes .EXE file structure and loading procedures for systems that use a version of MS-DOS that is lower than 2.0. For MS-DOS 2.0 and higher, use Function Request 4BH, Load and Execute a Program, to load (or load and execute) an .EXE file.

The .EXE files produced by MS-LINK consist of two parts:
Control and relocation information
The load module

The control and relocation information is at the beginning of the file in an area called the header. The load module immediately follows the header.
The header is formatted as follows. (Note that offsets are in hexadecimal.)

Offset	Contents
00-01	Must contain 4DH, 5AH.
02-03	Number of bytes contained in last page; this is useful in reading overlays.
04-05	Size of the file in 512-byte pages, including the header.
06-07	Number of relocation entries in table.

08-09 Size of the header in 16-byte paragraphs. This is used to locate the beginning of the load module in the file.

0A-0B Minimum number of 16-byte paragraphs required above the end of the loaded program.

0C-0D Maximum number of 16-byte paragraphs required above the end of the loaded program. If both minalloc and maxalloc are 0, then the program will be loaded as high as possible.

0E-0F Initial value to be loaded into stack segment before starting program execution. This must be adjusted by relocation.

10-11 Value to be loaded into the SP register before starting program execution.

12-13 Negative sum of all the words in the file.

14-15 Initial value to be loaded into the IP register before starting program execution.

16-17 Initial value to be loaded into the CS register before starting program execution. This must be adjusted by relocation.

18-19 Relative byte offset from beginning of run file to relocation table.

1A-1B The number of the overlay as generated by MS-LINK.

The relocation table follows the formatted area described above. This table consists of a variable number of relocation items. Each relocation item contains two fields: a two-byte offset value, followed by a two-byte segment value. These two fields contain the offset into the load module of a word which requires modification before the module is given control. The following steps describe this process:

1. The formatted part of the header is read into memory. Its size is 1BH.
2. A portion of memory is allocated depending on the size of the load module and the allocation numbers (0A-0B and 0C-0D). MS-DOS attempts to allocate FFFFH paragraphs. This will always fail, returning the size of the largest free block. If this block is smaller than minalloc and loadsize, then there will be no memory error. If this block is larger than maxalloc and loadsize, MS-DOS will allocate (maxalloc + loadsize). Otherwise, MS-DOS will allocate the largest free block of memory.
3. A Program Segment Prefix is built in the lowest part of the allocated memory.
4. The load module size is calculated by subtracting the header size from the file size. Offsets 04-05 and 08-09 can be used for this calculation. The actual size is downward-adjusted

based on the contents of offsets 02-03. Based on the setting of the high/low loader switch, an appropriate segment is determined at which to load the load module. This segment is called the start segment.

5. The load module is read into memory beginning with the start segment.
6. The relocation table items are read into a work area.
7. Each relocation table item segment value is added to the start segment value. This calculated segment, plus the relocation item offset value, points to a word in the load module to which is added the start segment value. The result is placed back into the word in the load module.
8. Once all relocation items have been processed, the SS and SP registers are set from the values in the header. Then, the start segment value is added to SS. The ES and DS registers are set to the segment address of the Program Segment Prefix. The start segment value is added to the header CS register value. The result, along with the header IP value, is the initial CS:IP to transfer to before starting execution of the program.

INDEX

NCR

MS-LIB Library Manager

MS-LIB
CONTENTS

INTRODUCTION

Features and Benefits

MS-LIB creates and modifies library files that are used with Microsoft's MS-LINK Linker Utility. MS-LIB can add object files to a library, delete modules from a library, or extract modules from a library and place the extracted modules into separate object files.

MS-LIB provides a means of creating either general or special libraries for a variety of programs or for specific programs only. With MS-LIB you can create a library for a language compiler, or you can create a library for one program only, which would permit very fast linking and possibly more efficient execution.

You can modify individual modules within a library by extracting the modules, making changes, then adding the modules to the library again. You can also replace an existing module with a different module or with a new version of an existing module.

The command scanner in MS-LIB is the same as the one used in Microsoft's MS-LINK, MS-Pascal, MS-FORTRAN, and other 16-bit Microsoft products. If you have used any of these products, using MS-LIB is familiar to you. Command syntax is straightforward, and MS-LIB prompts you for any of the commands it needs that you have not supplied. There are no surprises in the user interface.

Overview of MS-LIB Operation

MS-LIB performs two basic actions: it deletes modules from a library file, and it changes object files into modules and appends them to a library file. These two actions underlie five library manager functions:

delete a module
extract a module and place it in a separate object file
append an object file as a module of a library
replace a module in the library file with a new module
create a library file

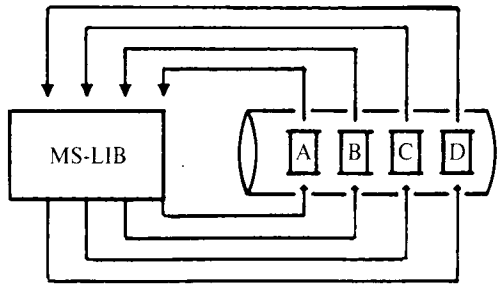
During each library session, MS-LIB first deletes or extracts modules, then appends new ones. In a single operation, MS-LIB reads each module into memory, checks it for consistency, and writes it back to the file. If you delete a module, MS-LIB reads in that module but does not write it back to the file. When MS-LIB writes back the next module to be retained, it places the module at the end of the last module written. This procedure effectively "closes up" the disk space to keep the library file from growing larger than necessary. When MS-LIB has read through the whole library file, it appends any new modules to the end of the file. Finally, MS-LIB creates the index, which MS-LINK uses to find modules and symbols in the library file, and outputs a cross reference listing of the PUBLIC symbols in the library, if you request such a listing. (Building the library index may take some extra time, up to 20 seconds in some cases.)

For example:

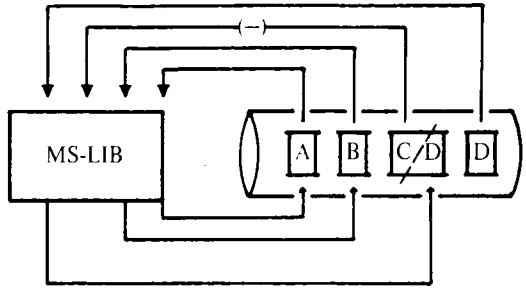
```
LIB PASCAL+HEAP-HEAP;
```

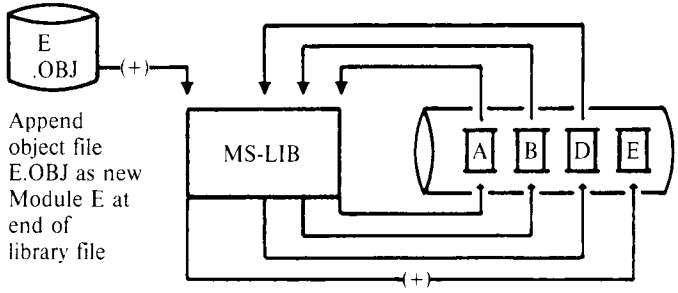
first deletes the library module HEAP from the library file, then adds the file HEAP.OBJ as the last module in the library. This order of execution prevents confusion in MS-LIB when a new version of a module replaces a version in the library file. Note that the replace function is simply the delete-append functions in succession. Also note that you can specify delete, append, or extract functions in any order; the order is insignificant to the MS-LIB command scanner.

Consistency Check only

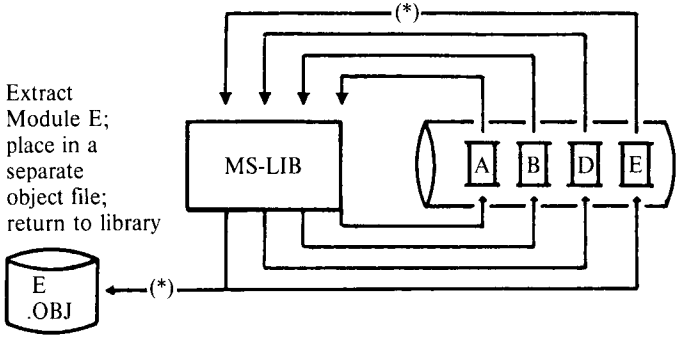


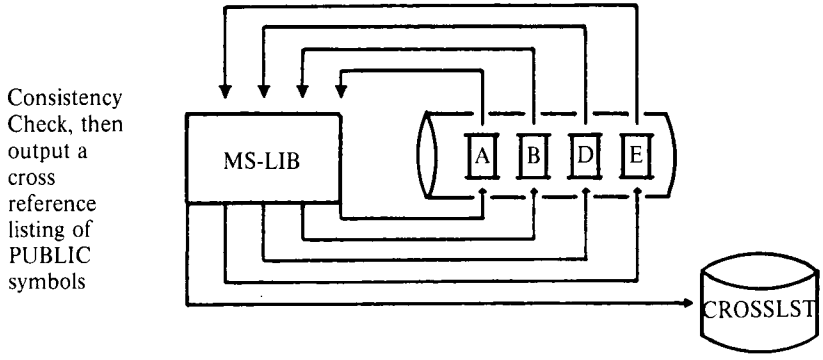
Delete Module C; Module D written to space of Module C





Append object file E.OBJ as new Module E at end of library file


Extract
Module E;
place in a
separate
object file;
return to library
(*)
MS-LIB
A
B
D
E
E
.OBJ
(*)


Consistency
Check, then
output a
cross
reference
listing of
PUBLIC
symbols
MS-LIB
A
B
D
E
CROSSLST

CHAPTER 1
RUNNING MS-LIB

Running MS-LIB requires two types of commands: a command to invoke MS-LIB and answers to command prompts. Usually you will enter all the commands to MS-LIB on the terminal keyboard. As an option, answers to the command prompts may be contained in a Response File. Some special command characters exist. Some are used as a required part of MS-LIB commands. Others assist you while entering MS-LIB commands.

1.1 INVOKING MS-LIB

MS-LIB may be invoked three ways. By the first method, you enter the commands as answers to individual prompts. By the second method, you enter all commands on the line used to invoke MS-LIB. By the third method, you create a Response File that contains all the necessary commands.

Summary of Methods to invoke MS-LIB

Method 1	LIB
Method 2	LIB <library> <operations>,<listing>
Method 3	LIB @ <filespec>

1.1.1 Method 1: LIB

Enter:

LIB

MS-LIB will be loaded into memory. Then, MS-LIB returns a series of three text prompts that appear one at a time. You answer the prompts as commands to MS-LIB to perform specific tasks.
The Command Prompts and Command Characters are summarized here. The Command Prompts and Command Characters are described fully in Sections 1.2 and 1.3.

Summary of Command Prompts

PROMPT	RESPONSES
Library file:	List filename of library to be manipulated (default: filename extension .LIB)
Operation:	List command character(s) followed by module name(s) or object filename(s) (default action: no changes. default object filename extension: .OBJ)
List file:	List filename for a cross reference listing file (default: NUL; no file)

Summary of Command Characters

Character	Action
+	Append an object file as the last module
-	Delete a module from the library
*	Extract a module and place in an object file
;	Use default responses to remaining prompts
&	Extend current physical line; repeat command prompt
Control-C	Abort library session.

1.1.2 Method 2: LIB <library> <operations>,<listing>

Enter:

LIB <library> <operations>,<listing>

The entries following LIB are responses to the command prompts. The **library** and **operations** fields and all operations entries must be separated by one of the command characters plus, minus, and asterisk (+, -, *). If a cross reference listing is wanted, the name of the file must be separated from the last operations entry by a comma.

where: **library** is the name of a library file. MS-LIB assumes that the filename extension is .OBJ, which you may override by specifying a different extension. If the filename given for the **library** fields does not exist, MS-LIB will prompt you:

Library file does not exist. Create?

Enter Yes (or any response beginning with Y) to create a new library file. Enter No (or any other response not beginning with Y) to abort the library session.

operations is deleting a module, appending an object file as a module, or extracting a module as an object file from the library file. Use the three command characters plus (+), minus (-), and asterisk (*) to direct MS-LIB what to do with each module or object file.

listing is the name of the file you want to receive the cross reference listing of PUBLIC symbols in the modules in the library. The list is compiled after all module manipulation has taken place.

To select the default for remaining field(s), you may enter the semicolon command character.

If you enter a Library filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you enter a Library filename followed immediately by a comma and a List filename, MS-LIB will perform its consistency check of the library file, then produce the cross reference listing file.

Example
LIB PASCAL-HEAP+HEAP;

This example causes MS-LIB to delete the module HEAP from the library file PASCAL.LIB, then append the object file HEAP.OBJ as the last module of PASCAL.LIB (the module will be named HEAP).
If you have many operations to perform during a library session, use the ampersand (&) command character to extend the line so that you can enter additional object filenames and module names. Be sure to always include one of the command characters for operations (+, -, *) before the name of each module or object filename.

Example

LIB PASCAL<CR>

causes MS-LIB to perform a consistency check of the library file PASCAL.LIB. No other action is performed.

Example

LIB PASCAL,PASCROSS.PUB

causes MS-LIB to perform a consitency check of the library file PASCAL.LIB, then output a cross reference listing file named PASCROSS.PUB.

1.1.3 Method 3: LIB @ <filespec>

Enter:

LIB @ <filespec>

where: **filespec** is the name of a Response File. A Response File contains answers to the MS-LIB prompts (summarized under method 1 for invoking and described fully in Section 1.2). Method 3 permits you to conduct the MS-LIB session without interactive (direct) user responses to the MS-LIB prompts.

IMPORTANT

Before using method 3 to invoke MS-LIB, you must first create the Response File.

A Response File has text lines, one for each prompt. Responses must appear in the same order as the command prompts appear.

Use Command Characters in the Response File the same way as they are used for responses entered on the terminal keyboard.

When the library session begins, each prompt will be displayed in turn with the responses from the response file. If the response file does not contain answers for all the prompts, MS-LIB will use the default responses (no changes to the modules currently in the library file for Operation, and no cross reference listing file created).

If you enter a Library filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you enter a Library filename then only a carriage return of Operations then a comma and a List filename, MS-LIB will perform its consistency check of the library file, then produce the cross reference listing file.

Example:

PASCAL<CR>
+CURSOR+HEAP-HEAP*FOIBLES<CR>
CROSSLST<CR>

This Response File will cause MS-LIB to delete the module HEAP from the PASCAL.LIB library file, extract the module FOIBLES and place in an object file named FOIBLES.OBJ, then append the object files CURSOR.OBJ and HEAP.OBJ as the last two modules in the library. Then, MS-LIB will create a cross reference file named CROSSLST.

1.2 COMMAND PROMPTS

MS-LIB is commanded by entering responses to three text prompts. When you have entered your response to the current prompt, the next appears. When the last prompt has been answered, MS-LIB performs its library management functions without further command. When the library session is finished, MS-LIB exits to the operating system. When the operating system prompt is displayed, MS-LIB has finished the library session successfully. If the library session is unsuccessful, MS-LIB returns the appropriate error message.
MS-LIB prompts you for the name of the library file, the operation(s) you want to perform, and the name you want to give to a cross reference listing file, if any.

Library file:

Enter the name of the library file that you want to manipulate. MS-LIB assumes that the filename extension is .LIB. You can override this assumption by giving a filename extension when you enter the library filename. Because MS-LIB can manage only one library file at a time, only one filename is allowed in response to this prompt. Additional responses, except the semicolon command character, are ignored.
If you enter a library filename and follow it immediately with a semicolon command character, MS-LIB will perform a consistency check only, then return to the operating system. Any errors in the file will be reported.
If the filename you enter does not exist, MS-LIB returns the prompt:

Library file does not exist. Create?

You must enter either Yes or No, in either upper or lower (or mixed) case. Actually, MS-LIB checks the response of the letter Y as the first character. If any other character is entered first, MS-LIB terminates and returns to the operating system.

Operation:

Enter one of the three command characters for manipulating modules (+, -, *), followed immediately (no space) by the module name or the object filename. Plus sign appends an object file as the last module in the library file (see further discussion under the description of plus sign below). Minus sign deletes a module from the library file. Asterisk extracts a module from the library and places it in a separate object file with the filename taken from the module name and a filename extension .OBJ.

When you have a large number of modules to manipulate (more than can be typed on one line), enter an ampersand (&) as the last character on the line. MS-LIB will repeat the Operation prompt, which permits you to enter additional module names and object filenames.

MS-LIB allows you to enter operations on modules and object files in any order you want.

More information about order of execution and what MS-LIB does with each module is given in the descriptions of each Command Character.

List file:

If you want a cross reference list of the PUBLIC symbols in the modules in the library file after your manipulations, enter a filename in which you want MS-LIB to place the cross reference listing. If you do not enter a filename, no cross reference listing is generated (a NUL file).

The response to the List file prompt is a file specification. Therefore, you can specify, along with the filename, a drive (or device) designation and a filename extension. The List file is not given a default filename extension. If you want the file to have a filename extension, you must specify it when entering the filename.

The cross reference listing file contains two lists. The first list is an alphabetical listing of all PUBLIC symbols. Each symbol name is followed by the name of its module. The second list is an alphabetical list of the modules in the library. Under each module name is an alphabetical listing of the PUBLIC symbols in that module.

1.3 COMMAND CHARACTERS

MS-LIB provides six command characters: three of the command characters are required in responses to the Operation prompt; the other three command characters provide you additional helpful commands to MS-LIB.

+ The plus sign followed by an object filename appends the object file as the last module in the library named in response to the Library file prompt. When MS-LIB sees the plus sign, it assumes that the filename extension is .OBJ. You may override this assumption by specifying a different filename extension.
MS-LIB strips the drive designation and the extension from the object file specification, leaving only the filename. For example, if the object file to be appended as a module to a library is:

B:CURSOR.OBJ

a response to the Operation prompt of:

+B:CURSOR.OBJ

causes MS-LIB to strip off the B: and the .OBJ, leaving only CURSOR, which becomes a module named CURSOR in the library.

NOTE

The distinction between an object file and a module (or object module) is that the file possesses a drive designation (even if it is default drive) and a filename extension. Object modules possess neither of these.

- The minus sign followed by a module name deletes that module from the library file. MS-LIB then "closes up" the file space left empty by the deletion. This cleanup action keeps the library file from growing larger than necessary with empty space. Remember that new modules, even replacement modules are added to the end of the file, not stuffed into space vacated by deleting modules.

* The asterisk followed by a module name extracts that module from the library file and places it into a separate object file. The module will still exist in the library (extract means, essentially, copy the module to a separate object file). The module name is used as the filename. MS-LIB adds the default drive designation and the filename extension .OBJ. For example, if the module to be extracted is:

CURSOR

and the current default disk drive is A:, a reponse to the Operation prompt of:

*CURSOR

causes MS-LIB to extract the module named CURSOR from the library file and to set it up as an object file with the file specification of:

default drive:CURSOR.OBJ

(The drive designation and filename extension cannot be overridden. You can, however, rename the file, giving a new filename extension, and/or copy the file to a new disk drive, giving a new filename and/or filename extension.)

; Use a single semicolon (;) followed immediately by a carriage return at any time after responding to the first prompt (from Library file on) to select default responses to the remaining prompts. This feature saves time and overrides the need to answer additional prompts.

NOTE

Once the semicolon has been entered, you can no longer respond to any of the prompts for that library session. Therefore, do not use the semicolon to skip over some prompts. For this, use carriage return.

Example:

Library file: FUN <CR>
Operation: +CURSOR;<CR>

The remaining prompt will not appear, and MS-LIB will use the default value (no cross reference file).

&

Use the ampersand to extend the current physical line. This command character will only be needed for the Operation prompt. MS-LIB can perform many functions during a single library session. The number of modules you can append is limited only by disk space. The number of modules you can replace or extract is also limited only by disk space. The number of modules you can delete is limited only by the number of modules in the library file. However, the line length for a response to any prompt is limited to the line length of your system. For a large number of responses to the Operation prompt, place an ampersand at the end of a line. MS-LIB will display the Operation prompt again, then enter more responses. You may use the ampersand character as many times as you need. For example:

```
Library file: FUN<CR>
Operation: +CURSOR-HEAP+HEAP*FOIBLES&
Operation: *INIT+ASSUME+RIDE;<CR>
```

MS-LIB will delete the module HEAP, extract the modules FOIBLES and INIT (creating two files, FOIBLES.OBJ and INIT.OBJ), then append the object files CURSOR, HEAP, ASSUME, and RIDE. Note, however, that MS-LIB allows you to enter your Operation responses in any order.

Control-C

Use Control-C at any time to abort the library session. If you enter an erroneous response, such as the wrong filename or module name, or an incorrectly spelled filename or module name, you must press CTRL-C to exit MS-LIB then reinvoke MS-LIB and start over. If the error has been typed but not entered, you may delete the erroneous characters, but for that line only.

CHAPTER 2
ERROR MESSAGES

<symbol> is a multiply defined PUBLIC. Proceed?
Cause: two modules define the same public symbol. The user is asked to confirm the removal of the definition of the old symbol. A No response leaves the library in an undetermined state.
Cure: Remove the PUBLIC declaration from one of the object modules and recompile or reassemble.

Allocate error on VM.TMP
Cause: out of space

Cannot create extract file
Cause: no room in directory for extract file

Cannot create list file
Cause: No room in directory for library file

Cannot nest response file
Cause: "@filespec" in response (or indirect) file

Cannot open VM.TMP
Cause: no room for VM.TMP in disk directory

Cannot write library file
Cause: Out of space

Close error on extract file
Cause: out of space

Error: An internal error has occurred.
Contact Microsoft, Inc.

Fatal Error: Cannot open input file
Cause: Mistyped object file name

Fatal Error: Module is not in the library
Cause: trying to delete a module that is not in the library

Input file read error
Cause: bad object module or faulty disk

Invalid object module/library
Cause: bad object and/or library

Library Disk is full
Cause: no more room on diskette

Listing file write error
Cause: out of space

No library file specified
    Cause: no response to Library File prompt
Read error on VM.TMP
    Cause: disk not ready for read
Symbol table capacity exceeded
    Cause: too many public symbols (about 30K chars in symbols)
Too many object modules
    Cause: more than 500 object modules
Too many public symbols
    Cause: 1024 public symbols maximum
Write error on library/extract file
    Cause: Out of space
Write error on VM.TMP
    Cause: out of space

NCR

DEBUG Utility

DEBUG UTILITY
CONTENTS

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW OF DEBUG

The Microsoft DEBUG Utility (DEBUG) is a debugging program that provides a controlled testing environment for binary and executable object files. Note that EDLIN is used to alter source files; DEBUG is EDLIN's counterpart for binary files. DEBUG eliminates the need to reassemble a program to see if a problem has been fixed by a minor change. It allows you to alter the contents of a file or the contents of a CPU register, and then to immediately reexecute a program to check on the validity of the changes.

All DEBUG commands may be aborted at any time by pressing <CONTROL-C>. <CONTROL-S> suspends the display, so that you can read it before the output scrolls away. Entering any key other than <CONTROL-C> or <CONTROL-S> restarts the display. All of these commands are consistent with the control character functions available at the MS-DOS command level.

1.2 HOW TO START DEBUG

DEBUG may be started two ways. By the first method, you type all commands in response to the DEBUG prompt (a hyphen). By the second method, you type all commands on the line used to start DEBUG.

Summary of Methods to Start DEBUG

Method 1	DEBUG
Method 2	DEBUG [<filespec> [<arglist>]]

1.2.1 Method 1: DEBUG

To start DEBUG using method 1, type:

```
DEBUG
```

DEBUG responds with the hyphen (-) prompt, signaling that it is ready to accept your commands. Since no filename has been specified, current memory, disk sectors, or disk files can be worked on by using other commands.

Warnings

1. When DEBUG (Version 2.0) is started, it sets up a program header at offset 0 in the program work area. On previous versions of DEBUG, you could overwrite this header. You can still overwrite the default header if no <filespec> is given to DEBUG. If you are debugging a .COM or .EXE file, however, do not tamper with the program header below address 5CH, or DEBUG will terminate.
2. Do not restart a program after the "Program terminated normally" message is displayed. You must reload the program with the N and L commands for it to run properly.

1.2.2 Method 2: Command Line

To start DEBUG using a command line, type:

```
DEBUG [<filespec> [<arglist>]]
```

For example, if a <filespec> is specified, then the following is a typical command to start DEBUG:

```
DEBUG FILE.EXE
```

DEBUG then loads FILE.EXE into memory starting at 100 hexadecimal in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory.
An <arglist> may be specified if <filespec> is present. The <arglist> is a list of filename parameters and switches that are to be passed to the program <filespec> . Thus, when <filespec> is loaded into memory, it is loaded as if it had been started with the command:

<filespec> <arglist>

Here, <filespec> is the file to be debugged, and the <arglist> is the rest of the command line that is used when <filespec> is invoked and loaded into memory.

CHAPTER 2
COMMANDS

2.1 COMMAND INFORMATION

Each DEBUG command consists of a single letter followed by one or more parameters. Additionally, the control characters and the special editing functions described in the **MS-DOS User's Guide,** apply inside DEBUG.

If a syntax error occurs in a DEBUG command, DEBUG reprints the command line and indicates the error with an up-arrow (^) and the word "error."

For example:

```
dcs:100 cs:110
     ^ error
```

Any combination of uppercase and lowercase letters may be used in commands and parameters.

The DEBUG commands are summarized in Table 2.1 and are described in detail, with examples, following the description of command parameters.

Table 2.1 DEBUG COMMANDS

DEBUG Command	Function
A[<address>]	Assemble
C<range> <address>	Compare
D[<range>]	Dump
E<address> [<list>]	Enter
F<range> <list>	Fill
G[=<address> [<address>. . .]]	Go
H<value> <value>	Hex
I<value>	Input
L[<address> [<drive> <record> <record>]]	Load
M<range> <address>	Move
N<filename> [<filename>]	Name
O<value> <byte>	Output
Q	Quit
R[<register-name>]	Register
S<range> <list>	Search
T[=<address>] [<value>]	Trace
U[<range>]	Unassemble
W[<address> [<drive> <record> <record>]]	Write

2.2 PARAMETERS

All DEBUG commands accept parameters, except the Quit command. Parameters may be separated by delimiters (spaces or commas), but a delimiter is required only between two consecutive hexadecimal values. Thus, the following commands are equivalent:

```
dcs:100 110
d cs:100 110
d,cs:100,110
```

PARAMETER DEFINITION

<drive> A one-digit hexadecimal value to indicate which drive a file will be loaded from or written to. The valid values are 0-3. These values designate the drives as follows: 0=A:, 1=B:, 2=C:, 3=D:.

<byte> A two-digit hexadecimal value to be placed in or read from an address or register.

<record> A 1- to 3-digit hexadecimal value used to indicate the logical record number on the disk and the number of disk sectors to be written or loaded. Logical records correspond to sectors. However, their numbering differs since they represent the entire disk space.

<value> A hexadecimal value up to four digits used to specify a port number or the number of times a command should repeat its functions.

<address> A two-part designation consisting of either an alphabetic segment register designation or a four-digit segment address plus an offset value. The segment designation or segment address may be omitted, in which case the default segment is used. DS is the default segment for all commands except G, L, T, U, and W, for which the default segment is CS. All numeric values are hexadecimal.

For example:

```
CS:0100
04BA:0100
```

The colon is required between a segment designation (whether numeric or alphabetic) and an offset.

<range< Two <address>es: e.g., <address> <address>; or one <address>, an L, and a <value>: e.g., <adress> L <value> where <value> is the number of lines the command should operate on, and LB0 is assumed. The last form cannot be used if another hex value follows the <range>, since the hex value would be interpreted as the second <address> of the <range>.
Examples:

```
CS:100 110
CS:100 L 10
CS:100
```

The following is illegal:

```
CS:100 CS:110
       ^
       error
```

The limit for <range> is 10 000 hex. To specify a <value> of 10 000 hex within four digits, type 0000 (or 0).

<list> A series of <byte> values or of <string>s. <list> must be the last parameter on the command line.

Example:

```
fcs:100 42 45 52 54 41
```

<string> Any number of characters enclosed in quote marks. Quote marks may be either single (') or double ("). If the delimiter quote marks must appear within a <string>, the quote marks must be doubled. For example, the following strings are legal:

```
'This is a "string" is okay.'
'This is a "string" is okay.'
```

However, this string is illegal:

```
'This is a 'string' is not.'
```

Similarly, these strings are legal:

```
"This is a 'string' is okay."
"This is a ""string"" is okay."
```

However, this string is illegal:

"This is a "string" is not."

Note that the double quote marks are not necessary in the following strings:

"This is a "string" is not necessary."
'This is a ""string"" is not necessary.'

The ASCII values of the characters in the string are used as a <list> of byte values.

NAME Assemble

PURPOSE Assembles 8086/8087/8088 mnemonics directly into memory.

SYNTAX A[<address>]

COMMENTS If a syntax error is found, DEBUG responds with

^Error

and redisplays the current assembly address.
All numeric values are hexadecimal and must be entered as 1-4 characters. Prefix mnemonics must be specified in front of the opcode to which they refer. They may also be entered on a separate line.
The segment override mnemonics are CS:, DS:, ES:, and SS:. The mnemonic for the far return is RETF.
String manipulation mnemonics must explicitly state the string size. For example, use MOVSW to move word strings and MOVSB to move byte strings.
The assembler will automatically assemble short, near or far jumps and calls, depending on byte displacement to the destination address. These may be overridden with the NEAR or FAR prefix. For example:

```
0100:0500 JMP  502        ; a 2-byte short jump
0100:0502 JMP  NEAR 505   ; a 3-byte near jump
0100:505  JMP  FAR 50A    ; a 5-byte far jump
```

The NEAR prefix may be abbreviated to NE, but the FAR prefix cannot be abbreviated.
DEBUG cannot tell whether some operands refer to a word memory location or to a byte memory location. In this case, the data type must be explicitly stated with the prefix "WORD PTR" or "BYTE PTR". Acceptable abbreviations are "WO" and "BY". For example:

```
NEG   BYTE PTR [128]
DEC   WO [SI]
```

DEBUG also cannot tell whether an operand refers to a memory location or to an immediate operand. DEBUG uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
MOV     AX,21    ; Load AX with 21H
MOV     AX,[21]  ; Load AX with the
                 ; contents
                 ; of memory location 21H
```

Two popular pseudo-instructions are available with Assemble. The DB opcode will assemble byte values directly into memory. The DW opcode will assemble word values directly into memory. For example:

```
DB      1,2,3,4,"THIS IS AN EXAMPLE"
DB      'THIS IS A QUOTE: " '
DB      "THIS IS A QUOTE: ' "

DW      1000,2000,3000,"BACH"
```

Assemble supports all forms of register indirect commands. For example:

```
ADD     BX,34[BP+2].[SI-1]
POP     [BP+DI]
PUSH    [SI]
```

All opcode synonyms are also supported. For example:

```
LOOPZ   100
LOOPE   100

JA      200
JNBE    200
```

For 8087 opcodes, the WAIT or FWAIT must be explicitly specified. For example:

```
FWAIT FADD ST,ST(3) ; This line will assemble
                    ; an FWAIT prefix
LD TBYTE PTR [BX]   ; This line will not
```

NAME Compare

PURPOSE Compares the portion of memory specified by <range> to a portion of the same size beginning at <address>.

SYNTAX C<range> <address>

COMMENTS If the two areas of memory are identical, there is no display and DEBUG returns with the MS-DOS prompt. If there **are** differences, they are displayed in this format:

<address1> <byte1> <byte2> <address2>

EXAMPLE The following commands have the same effect:

```
C100,1FF  300
    or
C100L100  300
```

Each command compares the block of memory from 100 to 1FFH with the block of memory from 300 to 3FFH.

NAME Dump

PURPOSE Displays the contents of the specified region of memory.

SYNTAX D[<range>]

COMMENTS If a range of addresses is specified, the contents of the range are displayed. If the D command is typed without parameters, 128 bytes are displayed at the first address (DS:100) after the address displayed by the previous Dump command.
The dump is displayed in two portions: a hexadecimal dump (each byte is shown in hexadecimal value) and an ASCII dump (the bytes are shown in ASCII characters). Nonprinting characters are denoted by a period (.) in the ASCII portion of the display. Each display line shows 16 bytes with a hyphen between the eighth and ninth bytes. At times, displays are split in this manual to fit them on the page. Each displayed line begins on a 16-byte boundary.

If you type the command:

```
dcs:100 110
```

DEBUG displays the dump in the following format:

```
04BA:0100 42 45 52 54 41 . . . 4E 44 TOM SAWYER
```

If you type the following command:

```
D
```

the display is formatted as described above. Each line of the display begins with an address, incremented by 16 from the address on the previous line. Each subsequent D (typed without parameters) displays the bytes immediately following those last displayed.

If you type the command:

```
DCS:100 L 20
```

the display is formatted as described above, but 20H bytes are displayed.
If then you type the command:

```
DCS:100 115
```

the display is formatted as described above, but all the bytes in the range of lines from 100H to 115H in the CS segment are displayed.

NAME Enter

PURPOSE Enters byte values into memory at the specified <address>.

SYNTAX E<address> [<list>]

COMMENTS If the optional <list> of values is typed, the replacement of byte values occurs automatically. (If an error occurs, no byte values are changed.)
If the <address> is typed without the optional <list>, DEBUG displays the address and its contents, then repeats the address on the next line and wait for your input. At this point, the Enter command waits for you to perform one of the following actions:

1. Replace a byte value with a value you type. Simply type the value after the current value. If the value typed in is not a legal hexadecimal value or if more than two digits are typed, the illegal or extra character is not echoed.
2. Press the <SPACE> bar to advance to the next byte. To change the value, simply type the new value as described in (1.) above. If you space beyond an 8-byte boundary, DEBUG starts a new display line with the address displayed at the beginning.
3. Type a hyphen (-) to return to the preceding byte. If you decide to change a byte behind the current position, typing the hyphen returns the current position to the previous byte. When the hyphen is typed, a new line is started with the address and its byte value displayed.
4. Press the <RETURN> key to terminate the Enter command. The <RETURN> key may be pressed at any byte position.

EXAMPLE Assume that the following command is typed:

```
ECS:100
```

DEBUG displays:

```
04BA:0100  EB.-
```

To change this value to 41, type 41 as shown:

```
04BA:0100  EB.41-
```

To step through the subsequent bytes, press the <SPACE> bar to see:

```
04BA:0100  EB.41  10.  00.  BC.-
```

To change BC to 42:

```
04BA:0100  EB.41  10.  00.  BC.42-
```

Now, realizing that 10 should be 6F, type the hyphen as many times as needed to return to byte 0101 (value 10), then replace 10 with 6F:

```
04BA:0100  EB.41  10.  00.  BC.42-
04BA:0102  00.--
04BA:0101  10.6F-
```

Pressing the <RETURN> key ends the Enter command and returns to the DEBUG command level.

NAME Fill

PURPOSE Fills the addresses in the <range> with the values in the <list>.

SYNTAX F<range> <list>

COMMENTS If the <range> contains more bytes than the number of values in the <list>, the <list> will be used repeatedly until all bytes in the <range> are filled. If the <list> contains more values than the number of bytes in the <range>, the extra values in the <list> will be ignored. If any of the memory in the <range> is not valid (bad or nonexistent), the error will occur in all succeeding locations.

EXAMPLE Assume that the following command is typed:

```
F04BA:100 L 100 42 45 52 54 41
```

DEBUG fills memory locations 04BA:100 through 04BA:1FF with the bytes specified. The five values are repeated until all 100H bytes are filled.

NAME Go

PURPOSE Executes the program currently in memory.

SYNTAX G [=<address> [<address>. . .]]

COMMENTS If only the Go command is typed, the program executes as if the program had run outside DEBUG.

If = <address> is set, execution begins at the address specified. The equal sign (=) is required, so that DEBUG can distinguish the start = <address> from the breakpoint <address>es.

With the other optional addresses set, execution stops at the first <address> encountered, regardless of that address' position in the list of addresses to halt execution or program branching. When program execution reaches a breakpoint, the registers, flags, and decoded instruction are displayed for the last instruction executed. (The result is the same as if you had typed the Register command for the breakpoint address.)

Up to ten breakpoints may be set. Breakpoints may be set only at addresses containing the first byte of an 8086 opcode. If more than ten breakpoints are set, DEBUG returns the BP Error message.

The user stack pointer must be valid and have 6 bytes available for this command. The G command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set, and the user flags, Code Segment register, and Instruction Pointer are pushed on the user stack. (Thus, if the user stack is not valid or is too small, the operating system may crash.) An interrupt code (0CCH) is placed at the specified breakpoint address(es).

When an instruction with the breakpoint code is encountered, all breakpoint addresses are restored to their original instructions. If execution is not halted at one of the breakpoints, the interrupt codes are not replaced with the original instructions.

EXAMPLE Assume that the following command is typed:

```
GCS:7550
```

The program currently in memory executes up to the address 7550 in the CS segment. DEBUG then displays registers and flags, after which the Go command is terminated.

After a breakpoint has been encountered, if you type the Go command again, then the program executes just as if you had typed the filename at the MS-DOS command level. The only difference is that program execution begins at the instruction after the breakpoint rather than at the usual start address.

NAME Hex

PURPOSE Performs hexadecimal arithmetic on the two parameters specified.

SYNTAX H<value> <value>

COMMENTS First, DEBUG adds the two parameters, then subtracts the second parameter from the first. The results of the arithmetic are displayed on one line; first the sum, then the difference.

EXAMPLE Assume that the following command is typed:

```
H19F 10A
```

DEBUG performs the calculations and then displays the result:

```
02A9  0095
```

NAME Input

PURPOSE Inputs and displays one byte from the port specified by <value>.

SYNTAX I<value>

COMMENTS A 16-bit port address is allowed.

EXAMPLE Assume that you type the following command:

```
I2F8
```

Assume also that the byte at the port is 42H. DEBUG inputs the byte and displays the value:

```
42
```

NAME Load

PURPOSE Loads a file into memory.

SYNTAX L[<address> [<drive> <record> <record>]]

COMMENTS Set BX:CX to the number of bytes read. The file must have been named either when DEBUG was started or with the N command. Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C.

If the L command is typed without any parameters, DEBUG loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded. If the L command is typed with an address parameter, loading begins at the memory <address> specified. If L is typed with all parameters, absolute disk sectors are loaded, not a file. The <record>s are taken from the <drive> specified (the drive designation is numeric here-0=A:, 1=8:, 2=C:, etc.); DEBUG begins loading with the first <record> specified, and continues until the number of sectors specified in the second <record> have been loaded.

EXAMPLE Assume that the following commands are typed:

```
A>DEBUG
-NFILE.COM
```

Now, to load FILE.COM, type:

```
L
```

DEBUG loads the file and then displays the DEBUG prompt. Assume that you want to load only portions of a file or certain records from a disk. To do this, type:

```
L04BA:100 2 0F 6D
```

DEBUG then loads 109 (6D hex) records beginning with logical record number 15 into memory beginning at address 04BA:0100. When the records have been loaded, DEBUG simply returns the - prompt.

If the file has a .EXE extension, it is relocated to the load address specified in the header of the .EXE file: the <address> parameter is always ignored for .EXE files. The header itself is stripped off the .EXE file before it is loaded into memory. Thus the size of an .EXE file on disk will differ from its size in memory.
If the file named by the Name command or specified when DEBUG is started is a .HEX file, then typing the L command with no parameters causes DEBUG to load the file beginning at the address specified in the .HEX file. If the L command includes the option <address>, DEBUG adds the <address> specified in the L command to the address found in the .HEX file to determine the start address for loading the file.

NAME Move

PURPOSE Moves the block of memory specified by <range> to the location beginning at the <address> specified.

SYNTAX M<range> <address>

COMMENTS Overlapping moves (i.e., moves where part of the block overlaps some of the current addresses) are always performed without loss of data. Addresses that could be overwritten are moved first. The sequence for moves from higher addresses to lower addresses is to move the data beginning at the block's lowest address and then to work towards the highest. The sequence for moves from lower addresses to higher addresses is to move the data beginning at the block's highest address and to work towards the lowest.

Note that if the addresses in the block being moved will not have new data written to them, the data there before the move will remain. The M command copies the data from one area into another, in the sequence described, and writes over the new addresses. This is why the sequence of the move is important.

EXAMPLE Assume that you type:

```
MCS:100 110 CS:500
```

DEBUG first moves address CS:110 to address CS:510, then CS:10F to CS:50F, and so on until CS:100 is moved to CS:500. You should type the D command, using the <address> typed for the M command, to review the results of the move.

NAME Name

PURPOSE Sets filenames.

SYNTAX N<filename> [<filename> . . .]

COMMENTS The Name command performs two functions. First, Name is used to assign a filename for a later Load or Write command. Thus, if you start DEBUG without naming any file to be debugged, then the N<filename> command must be typed before a file can be loaded. Second, Name is used to assign filename parameters to the file being debugged. In this case, Name accepts a list of parameters that are used by the file being debugged.
These two functions overlap. Consider the following set of DEBUG commands:

```
-NFILE1.EXE
-L
-G
```

Because of the effects of the Name command, Name will perform the following steps:

1. (N)ame assigns the filename FILE1.EXE to the filename to be used in any later Load or Write commands.
2. (N)ame also assigns the filename FILE1.EXE to the first filename parameter used by any program that is later debugged.
3. (L)oad loads FILE1.EXE into memory.
4. (G)o causes FILE1.EXE to be executed with FILE1.EXE as the single filename parameter (that is, FILE1.EXE is executed as if FILE1.EXE had been typed at the command level).

A more useful chain of commands might look like this:

```
-NFILE1.EXE
-L
-NFILE2.DAT FILE3.DAT
-G
```

Here, Name sets FILE1.EXE as the filename for the subsequent Load command. The Load command loads FILE1.EXE into memory, and then the Name command is used again, this time to specify the parameters to be used by FILE1.EXE. Finally, when the Go command is executed, FILE1.EXE is executed as if FILE1 FILE2.DAT FILE3.DAT had been typed at the MS-DOS command level. Note that if a Write command were executed at this point, then FILE1.EXE - the file being debugged - would be saved with the name FILE2.DAT! To avoid such undesired results, you should always execute a Name command before either a Load or a Write.

There are four regions of memory that can be affected by the Name command:

CS:5C	FCB for file 1
CS:6C	FCB for file 2
CS:80	Count of characters
CS:81	All characters typed

A File Control Block (FCB) for the first filename parameter given to the Name command is set up at CS:5C. If a second filename parameter is typed, then an FCB is set up for it beginning at CS:6C. The number of characters typed in the Name command exclusive of the first character, "N") is given at location CS:80. The actual stream of characters given by the Name command (again, exclusive of the letter "N") begins at CS:81. Note that this stream of characters may contain switches and delimiters that would be legal in any command typed at the MS-DOS command level.

EXAMPLE

A typical use of the Name command is:

```
DEBUG PROG.COM
-NPARAM1 PARAM2/C
-G
-
```

In this case, the Go command executes the file in memory as if the following command line had been typed:

```
PROG PARAM1 PARAM2/C
```

Testing and debugging therefore reflect a normal runtime environment for PROG.COM.

NAME Output

PURPOSE Sends the <byte> specified to the output port specified by <value>.

SYNTAX 0<value> <byte>

COMMENTS A 16-bit port address is allowed.

EXAMPLE Type:

```
02F8 4F
```

DEBUG outputs the byte value 4F to output port 2F8.

NAME Quit

PURPOSE Terminates the DEBUG utility.

SYNTAX Q

COMMENTS The Q command takes no parameters and exits DEBUG without saving the file currently being operated on. You are returned to the MS-DOS command level.

EXAMPLE To end the debugging session, type:

Q<RETURN>

DEBUG has been terminated, and control returns to the MS-DOS command level.

NAME Register

PURPOSE Displays the contents of one or more CPU registers.

SYNTAX R[<register-name>]

COMMENTS If no <register-name> is typed, the R command dumps the register save area and displays the contents of all registers and flags.
If a register name is typed, the 16-byte value of that register is displayed in hexadecimal, and then a colon appears as a prompt. You then either type a <value> to change the register, or simply press the <RETURN> key if no change is wanted.
The only valid <register-name>s are:

AX	BP	SS	
BX	SI	CS	
CX	DI	IP	(IP and PC both refer to
DX	DS	PC	the Instruction Pointer.)
SP	ES	F	

Any other entry for <register-name> results in a BR Error message.
If F is entered as the <register-name>, DEBUG displays each flag with a two-character alphabetic code. To alter any flag, type the opposite two-letter code. The flags are either set or cleared.

The flags are listed below with their codes for SET and CLEAR:

FLAG NAME	SET	CLEAR
Overflow	OV	NV
Direction	DN Decrement	UP Increment
Interrupt	EI Enabled	DI Disabled
Sign	NG Negative	PL Plus
Zero	ZR	NZ
Auxiliary Carry	AC	NA
Parity	PE Even	PO Odd
Carry	CY	NC

Whenever you type the command RF, the flags are displayed in the order shown above in a row at the beginning of a line. At the end of the list of flags, DEBUG displays a hyphen (-). You may enter new flag values as alphabetic pairs. The new flag values can be entered in any order. You do not have to leave spaces between the flag entries. To exit the R command, press the <RETURN> key. Flags for which new values were not entered remain unchanged.

If more than one value is entered for a flag, DEBUG returns a DF Error message. If you enter a flag code other than those shown above, DEBUG returns a BF Error message. In both cases, the flags up to the error in the list are changed; flags at and after the error are not.

At startup, the segment registers are set to the bottom of free memory, the Instruction Pointer is set to 0100H, all flags are cleared, and the remaining registers are set to zero.

EXAMPLE Type:

R

DEBUG displays all registers, flags, and the decoded instruction for the current location. If the location is CS:11A, then the display will look similar to this:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D
BP=0000 SI=005C DI=0000 DS=04BA ES=04BA
SS=04BA CS=04BA IP=011A
NV UP DI NG NZ AC PE NC
04BA:011A   CD21        INT     21
```

If you type:

RF

DEBUG will display the flags:

NV UP DI NG NZ AC PE NC - -

Now, type any valid flag designation, in any order, with or without spaces.

For example:

NV UP DI NG NZ AC PE NC - PLEICY<RETURN>

DEBUG responds only with the DEBUG prompt. To see the changes, type either the R or RF command:

```
RF
NV UP EI PL NZ AC PE CY - -
```

Press <RETURN> to leave the flags this way, or to specify different flag values.

NAME Search

PURPOSE Searches the <range< specified for the <list> of bytes specified.

SYNTAX S<range> <list>

COMMENTS The <list> may contain one or more bytes, each separated by a space or comma. If the <list> contains more than one byte, only the first address of the byte string is returned. If the <list> contains only one byte, all addresses of the byte in the <range> are displayed.

EXAMPLE If you type:

```
SCS:100 110 41
```

DEBUG will display a response similar to this:

```
04BA:0104
04BA:010D
-type:
```

NAME Trace

PURPOSE Executes one instruction and displays the contents of all registers and flags, and the decoded instruction.

SYNTAX T[=<address>] [<value>]

COMMENTS If the optional =<address> is typed, tracing occurs at the =<address> specified. The optional <value> causes DEBUG to execute and trace the number of steps specified by <value>.
The T command uses the hardware trace mode of the 8086 or 8088 microprocessor. Consequently, you may also trace instructions stored in ROM (Read Only Memory).

EXAMPLE TYPE:

```
T
```

DEBUG returns a display of the registers, flags, and decoded instruction for that one instruction. Assume that the current position is 04BA:011A; DEBUG might return the display:

```
AX=0E00 BX=00FF CS=0007 DX=01FF SP=039D
BP=0000 SI=005C DI=0000 DS=04BA ES=04BA
SS=04BA CS=04BA IP=011A
NV UP DI NG NZ AC PE NC
04BA:011A   CD21        INT     21
```

If you type

```
T=011A 10
```

DEBUG executes sixteen (10 hex) instructions beginning at 011A in the current segment, and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed. Then the display stops, and you can see the register and flag values for the last few instructions performed. Remember that <CONTROL-S> suspends the display at any point, so that you can study the registers and flags for any instruction.

NAME Unassemble

PURPOSE Disassembles bytes and displays the source statements that correspond to them, with addresses and byte values.

SYNTAX U[<range>]

COMMENTS The display of disassembled code looks like a listing for an assembled file. If you type the U command without parameters, 20 hexadecimal bytes are disassembled at the first address after that displayed by the previous Unassemble command. If you type the U command with the <range> parameter, then DEBUG disassembles all bytes in the range. If the <range> is given as an <address> only, then 20H bytes are disassembled instead of 80H.

EXAMPLE Type:

```
U04BA:100 L10
```

DEBUG disassembles 16 bytes beginning at address 04BA:0100:

```
04BA:0100 206472    AND   [SI+72],AH
04BA:0103 69        DB    69
04BA:0104 7665      JBE   016B
04BA:0106 207370    AND   [BP+DI+70],DH
04BA:0109 65        DB    65
04BA:010A 63        DB    63
04BA:010B 69        DB    69
04BA:010C 66        DB    66
04BA:010D 69        DB    69
04BA:010E 63        DB    63
04BA:010F 61        DB    61
```

If you type

```
004ba:0100 0108
```

The display will show:

```
04BA:0100 206472  AND  [SI+72],AH
04BA:0103 69      DB   69
04BA:0104 7665    JBE  016B
04BA:0106 207370  AND  [BP+DI+70],DH
```

If the bytes in some addresses are altered, the disassembler alters the instruction statements. The U command can be typed for the changed locations, the new instructions viewed, and the disassembled code used to edit the source file.

NAME Write

PURPOSE Wirtes the file being debugged to a disk file.

SYNTAX W[<address> [<drive> <record> <records>]]

COMMENTS If you type W with no parameters, BX:CX must already be set to the number of bytes to be written; the file is written beginning from CS:100. If the W command is typed with just an address, then the file is written beginning at that address. If a G or T command has been used, BX:CX must be reset before using the Write command without parameters. Note that if a file is loaded and modified, the name, length, and starting address are all set correctly to save the modified file (as long as the length has not changed).
The file must have been named either with the DEBUG invocation command or with the N command (refer to the Name command earlier in this manual). Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C.
If the W command is typed with parameters, the write begins from the memory address specified; the file is written to the <drive> specified (the drive designation is numeric here-0=A:, 1=B:, 2=C:, etc.); DEBUG writes the file beginning at the logical record number specified by the first <record>; DEBUG continues to write the file until the number of sectors specified in the second <record> have been written.

WARNING

Writing to absolute sectors is **EXTREMELY** dangerous because the process bypasses the file handler.

EXAMPLE Type:

```
W
```

DEBUG will write the file to disk and then display the DEBUG prompt. Two examples are shown below.

```
W
--
```

```
WCS:100 1 37 2B
```

DEBUG writes out the contents of memory, beginning with the address CS:100 to the disk in drive B:. The data written out starts in disk logical record number 37H and consists of 2BH records. When the write is complete, DEBUG displays the prompt:

```
WCS:100 1 37 2B
--
```

2.3 ERROR MESSAGES

During the DEBUG session, you may receive any of the following error messages. Each error terminates the DEBUG command under which it occurred, but does not terminate DEBUG itself.

ERROR CODE DEFINITION

BF Bad flag

You attempted to alter a flag, but the characters typed were not one of the acceptable pairs of flag values. See the Register command for the list of acceptable flag entries.

BP Too many breakpoints

You specified more than ten breakpoints as parameters to the G command. Retype the Go command with ten or fewer breakpoints.

BR Bad register

You typed the R command with an invalid register name. See the Register command for the list of valid register names.

DF Double flag

You typed two values for one flag. You may specify a flag value only once per RF command.

Lizenzprogramme für den Personalcomputer NCR - DM V

Bitte füllen Sie diese Karte aus und senden Sie an umstehende Adresse.

Ich bestätige hiermit, das (die) nachfolgende(n) aufgezählte(n) Programm(e) erhalten zu haben:

```
MS-PRODUCT       : MS (TM)-DOS
MS VERSION NO.   : 2.0
SERIAL NO.       : 16545
```

Für die Benutzung der vorstehenden Programme gelten die Allgemeinen Lizenzbedingungen der NCR GmbH, die ich erhalten habe und mir bekannt sind.

Name: _____

Vorname: _____

Anschrift: _____

Datum: _____ _____

Unterschrift

Absender

_____

_____

_____

_____ _____

(Postleitzahl) (Ort)

Gebühr
bezahlt
Empfänger

Antwort-
Postkarte

An die
NCR GmbH
Ulmer Straße 160 b

8900 Augsburg