



Bedienungshandbuch

für CBM 600/700



Commodore Büromaschinen GmbH
Lyoner Str. 38 · 6000 Frankfurt/Main 71 · Tel. (0611) 6638-0
Telefax 6638-159 · Telex 4185663 comod

Dieses Handbuch wurde gescannt, bearbeitet und ins PDF-Format konvertiert von

Rüdiger Schuldes

schuldes@itsm.uni-stuttgart.de

(c) 2002

B E D I E N U N G S H A N D B U C H

F Ü R C B M 6 0 0 / 7 0 0

Inhaltsverzeichnis

1	Einleitung
2	Installation
2.1	Installationsbedingungen
2.2	Inbetriebnahme
3	Ein- / Ausschalten
4	Die Hardware
4.1	Die Modelle der Serien CBM 600 / 700
4.2	Systemüberblick
4.2.1	Bausteine
4.2.2	Speicherorganisation
5	Bedienung (Tastatur)
6	Systemstart
7	Abschluß der Arbeit mit dem Rechner
8	Allgemeines über Datenträger (Disketten) ...
8.1	Behandlung der Disketten
8.2	Datensicherung
8.3	Initialisieren neuer Disketten
9	Programmentwicklung
10	Editor

Anhang

1	Editor-ESCAPE-Funktionen
2	Systemvariablen
2.1	Statusvariablen
2.1.1	ST (Statusvariable)
2.1.2	DS (Diskstatus)
2.1.3	DS\$ (Diskstatus)
2.2	Zeitvariable (TI\$)
2.3	Fehlervariablen
2.3.1	EL (Fehlerzeile)
2.3.2	ER (Fehlernummer)
3	BASIC-Sprachumfang
3.1	BASIC-Kommandos
3.2	BASIC-Anweisungen
3.3	BASIC-Funktionen
4	Schnittstellen (Pinouts, Bausteine)
5	Die Handhabung der RS 232 C Schnittstelle ..
5.0	Vorbemerkungen
5.1	Extended BASIC 4.0 und RS 232 C
5.1.1	Sekundäradresse (sa)
5.1.2	Fileparameter (fp)
5.2	Die Hardware der Schnittstelle RS 232 c .
6	Tastaturcode

1 Einleitung

Systembeschreibung

Die Modelle der Systemfamilie CBM 600/700 gehören zu den modernsten Mikrocomputern der 8-Bit-Generation. Durch die Verdoppelung des Systemtaktes wurde eine Verarbeitungsgeschwindigkeit erreicht, welche sich hinter der der gängigen 16-Bit-Prozessoren nicht zu verstecken braucht.

Durch den großen Speicher (mindestens 128 KB RAM, ausbaufähig bis 960 KB) ist die Systemfamilie CBM 600/700 in einer Größenordnung angesiedelt, welche bisher nur wesentlich teureren Großanlagen vorbehalten war.

Die Leistung eines Computers wird jedoch nicht allein durch die Größe des Speichers und der Verarbeitungsgeschwindigkeit bestimmt. Genauso entscheidend ist die Peripherie. Um fast jedes auf dem Markt erhältliche Peripheriegerät anschließen zu können, sind die Computer der Systemfamilie CBM 600/700 mit mehreren Schnittstellen ausgerüstet. Zunächst ist hier die IEEE-488 (bzw. IEC 625) Schnittstelle zu erwähnen, über welche die gesamte Commodore System-Peripherie angeschlossen wird. Sehr viele Meßgeräte, welche auf Computerunterstützung angewiesen sind, besitzen ebenfalls diese Schnittstelle.

Ferner besitzt die Systemfamilie CBM 600/700 eine RS-232 (V.24) Schnittstelle. Über diesen Anschluß wird z. B. Datenfernübertragung oder Rechnerkommunikation realisiert. Viele der auf dem Markt erhältlichen Peripheriegeräte verwenden ebenfalls diese Schnittstelle zum Datenaustausch.

Zudem existiert der User-Port; das ist eine frei programmierbare parallele Schnittstelle, welche dadurch eine große Variabilität besitzt. So kann z.B. die oft verwendete sog. 'Centronics'-Schnittstelle implementiert werden. Der User-Port ist nicht aus dem Gehäuse herausgeführt, er befindet sich direkt auf der Platine.

2 Installation

2.1 Installationsbedingungen

Die Modelle der Systemfamilie stellen hinsichtlich der Klimabedingungen keine besonderen Anforderungen. Die Rechner arbeiten in jedem Klima, in dem auch Sie es als einigermaßen erträglich empfinden.

Auch was die elektrischen Gegebenheiten betrifft, brauchen Sie sich keine Gedanken zu machen: Die Netzteile der Commodore-Produkte haben genug Reserven, um auch große Schwankungen im Stromnetz oder hohe Spannungsspitzen "glattzubügeln". Lediglich in unmittelbarer Nähe von sehr starken Elektromotoren kann es beim Einschalten derselben zu Störungen kommen.

Um eine ausreichende Stromversorgung an dem Arbeitsplatz Ihres Computers brauchen Sie sich ebenfalls keine Gedanken zu machen: Er benötigt nicht mehr Strom als zwei normale Schreibtischlampen (ca. 130 Watt).

Der Vollständigkeit halber sei noch erwähnt, daß sehr hohe Radioaktivität oder "harte" Röntgenstrahlung zu Informationsverlust führen kann.

2.2 Inbetriebnahme

Vergewissern Sie sich, daß Ihr Rechner ausgeschaltet ist, bevor Sie mit der Installation beginnen. Bei den Modellen der Systemfamilie CBM 600 beachten Sie bitte, daß der Monitor ebenfalls ausgeschaltet sein sollte. Beachten Sie auch die Hinweise zur Inbetriebnahme des Monitors, die diesem beigelegt sind!

Der Ein-/Ausschalter des Rechners befindet sich auf seiner Rückseite.

Die Inbetriebnahme der Rechner der Systemfamilie CBM 600 unterscheidet sich nur im ersten Punkt von derjenigen der Systemfamilie CBM 700:

1. CBM 600:

Verbinden Sie als erstes Ihren Rechner mit dem Monitor. Verwenden Sie hierfür ein Videokabel. An dem Rechner befindet sich für diesen Zweck eine 5-polig (bzw. 7-polige) DIN-Buchse.

CBM 700:

Verbinden Sie zuerst die Tastatur mit dem Gehäuse, indem Sie den an der Tastatur befestigten 25-poligen Stecker in die entsprechende Buchse an der Frontseite des Computergehäuses stecken. Das Commodore-Zeichen auf dem Stecker zeigt dabei nach oben.

2. Als nächstes ist die Verbindung Ihrer Peripherie mit dem Rechner herzustellen. Dazu benötigen Sie ein S-P-Kabel (System-Peripherie-Kabel). Der flachere Stecker dieses Kabels gehört in den IEEE-Stecker am Rechner. Achten Sie bitte darauf, daß die Schrift auf diesem Stecker nach oben zeigt. Der andere Stecker wird mit einem der Peripheriegeräte verbunden. Für den Anschluß der restlichen Peripheriegeräte benötigen Sie jeweils ein P-P-Kabel (Peripherie-Peripherie-Kabel). Das eine Ende dieses Kabels wird auf den Stecker eines bereits angeschlossenen Peripheriegerätes gesteckt ("Huckepack"), das andere wird mit dem neu anzuschließenden Gerät verbunden.
3. Jetzt können Sie die Stromkabel anschließen. Ihr Computer ist nun betriebsbereit.

3 Ein- / Ausschalten

Die Ein-/Ausschalter befinden sich bei den Commodore-Rechner genauso wie bei den Diskettenstationen auf der Geräterückseite, bei den Druckern meist auf der Frontseite. Um nicht bei jedem Ein- und Ausschalten alle Geräte einzeln schalten zu müssen, empfiehlt es sich, eine Steckerleiste mit eingebautem Zentralschalter zu verwenden. Solche Steckerleisten sind in jedem Elektrohandel erhältlich.

Beim Einschalten des Rechners durchläuft er eine Testroutine, durch welche er sein 'Innenleben' auf eventuelle Fehler durchsucht. Nach ca. 4 bis 6 Sekunden (je nach Speicherausbau) meldet er seine Bereitschaft durch die Ausgabe einer Kopfzeile und der Meldung 'ready.'. Ihr Rechner ist nun betriebsbereit, Sie können sofort anfangen.

Bevor Sie den Rechner ausschalten, vergewissern Sie sich, daß Sie Ihre Daten gesichert (d.h., auf Diskette abgelegt) haben, falls Sie diese Daten später weiterverwenden wollen. Das gleiche gilt natürlich auch für Programme, die Sie erstellt haben.

4 Die Hardware

4.1 Die Modelle der Serien CBM 600 / CBM 700

Beiden Serien liegt das gleiche Betriebssystem zugrunde. Der eigentliche Unterschied zwischen den Serien CBM 600 und CBM 700 besteht darin, daß bei der Serie CBM 700 der Bildschirm integriert ist, wohingegen bei der Serie CBM 600 ein externer Monitor benötigt wird.

Bei den Modellen der Serie CBM 600 ist die Tastatur in dem Hauptgehäuse integriert, bei der Serie CBM 700 ist der Bildschirm mit dem Hauptgehäuse verbunden und die Tastatur über ein Kabel an dem Gehäuse angeschlossen.

4.2 Systemüberblick

4.2.1 Bausteine

Der RAM-Speicher besteht aus den Speichertypen 4864. Dies sind dynamische Speicherbausteine, welche jeweils eine Kapazität von 64 K Bit besitzen. Jeweils 8 dieser Bausteine werden 'parallelgeschaltet', um 64 K Byte zu erhalten. 8 dieser Bausteine stellen also eine Bank (oder ein Segment) dar. Für den RAM-Bereich in dem Systemsegment (Zero-Page etc.) wird ein statischer RAM-Baustein des Typs 2016 verwendet (2 K Byte).

Das Betriebssystem und der Basic-Interpreter sind in ROM-Bausteinen des Typs 2364 A enthalten. Diese Bausteine besitzen eine Speicherkapazität von jeweils 8 K Byte.

Außer den Speicherbausteinen besitzt die Systemfamilie noch einige Prozessoren, durch welche das richtige Zusammenspiel der verschiedenen Systemkomponenten gewährleistet wird. Es sind dies im einzelnen:

- MOS 6509 (Mikroprozessor)

Dieser Mikroprozessor ist ein Produkt aus der MOS 65xx-Reihe. Er unterscheidet sich von dem sehr bekannten Mikroprozessor MOS 6502 durch die Fähigkeit, 1 Megabyte adressieren zu können. Der Befehlssatz des MOS 6509 ist mit dem des MOS 6502 identisch, bis auf die Tatsache, daß bei 2 Befehlen des MOS 6509 alle 20 Addressleitungen ausgewertet werden gegenüber nur 16 Leitungen beim MOS 6502. Der Mikroprozessor MOS 6509 wird bei der Systemfamilie CBM 600/700 mit einer Taktfrequenz von 2 MHz versorgt.

- MOS 6525 (TPI, IEEE-Bus-Controller, Keyboard-Controller)

Dieser Baustein (ein 'Tri-Port Interface') verfügt über 24 E/A-Leitungen, mit welchen der IEEE-Bus kontrolliert wird. 2 Handshake-Leitungen und 5 Unterbrechungseingänge erleichtern diese Aufgabe. Die Tastatur Ihres Rechners wird ebenfalls mit einem Baustein dieses Typs behandelt.

- MOS 6526 (CIA, User-Port-Controller)

Mit diesem Prozessor (ein 'Complex Interface Adapter') wird u.a. der User-Port gesteuert. Ferner besitzt er zwei 16-Bit-Zähler sowie eine 24-Stunden-Uhr.

- MOS 6551 (ACIA, RS-232-C Schnittstellen-Controller)

Dieser 'Asynchronous Communication Interface Adapter' übernimmt die Kontrolle über die RS-232-C Schnittstelle. Er ermöglicht Übertragungsraten von 50 bis 19200 Baud, wandelt die zu übertragende Information in das gewünschte Format (Anzahl der Bits, Parität etc.) und überwacht den ordnungsgemäßen Datentransfer.

- MOS 6581 (SID, dreistimmiger Synthesizer)

Dieses 'Sound Interface Device' bietet äußerst umfangreiche Möglichkeiten zur Erzeugung von Tönen und Musik. So kann z.B. für jeden der 3 Oszillatoren die Wellenform, die Frequenz, die Lautstärke, Anstieg- und Abfallszeit etc. angegeben werden. Zusätzlich läßt sich noch ein programmierbares Filter dazuschalten.

- MOS 6545 (CRTC, Bildschirm-Controller)

Mit diesem Chip ('Cathode Ray Tube Controller') wird die Bildschirm-Ausgabe kontrolliert. Er bildet dadurch das Bindeglied zwischen dem Speicher (Bildschirmspeicher) und dem Bildschirm unter Zuhilfenahme des Zeichengenerators.

Bei Verwendung eines Coprozessors wird noch einer der beiden Mikroprozessoren verwendet:

- Z80 (Zilog)

Mit diesem Prozessor ist es möglich, das Betriebssystem CP/M 2.2 (c) zu verwenden. Damit steht die ganze für dieses Betriebssystem vorhandene Software zur Verfügung.

- 8088 (Intel)

Dieser 16-Bit-Mikroprozessor erlaubt es, entweder CP/M-86 (c) oder MS-DOS als Betriebssystem zu verwenden.

4.2.2 Speicherorganisation

Der gesamte Speicher ist bei der Systemfamilie CBM 600/700 in sogenannte Segmente (oder 'Banks') unterteilt. Jedes dieser Segmente umfaßt einen Adreßraum von 64 K Byte. Es können maximal 16 solcher Segmente verarbeitet werden. Diese Segmente sind von 0 bis 15 durchnummeriert.

Jedes Segment hat eine feste Bedeutung, die (zum Teil) vom implementierten Speicherausbau abhängt.

Bei den Modellen CBM 610 / CBM 710 (Modelle mit jeweils 128 K Byte Speicherkapazität) gilt folgende Aufteilung:

Segment Nr. 1 enthält den BASIC-Text, d.h. die Programme, die Sie verwenden.

In Segment Nr. 2 werden die Daten abgelegt, welche durch das Programm errechnet werden.

Bei Modellen mit 256 K Speicherkapazität ist die Verwendung von Segment Nr. 1 identisch mit den 128 K Byte Versionen.

In Segment Nr. 2 werden Felder (Dimensions, Arrays) abgelegt.

In Segment Nr. 3 werden einfache Variablen (nicht-indizierte Variablen) abgelegt. Zudem ist in diesem Segment der Platz für das Disketten-Betriebssystem für eventuell integrierte Laufwerke reserviert.

Segment Nr. 4 enthält die Strings, die vom Programm erstellt werden.

Bei allen Modellen ist die Verwendung des Segments Nr. 15 identisch: Dieses ist das Systemsegment. In ihm sind der BASIC-Interpreter, der Editor, der Kernal (Betriebssystem), die E/A-Bausteine sowie die Systeminformation (Zero-Page etc.) enthalten. Ein Teil des Adressraums in der Systembank (genauer: die Adressen von \$2000 (8192) bis \$7fff (32767)) sind freigehalten; er ist für individuelle Erweiterungen vorgesehen. Zu diesem Zweck sind die Adreßleitungen aus dem Gerät herausgeführt (Cartridge-Connector). In diesem Adreßraum können je nach Bedarf ROM-Bausteine, RAM-Speicher oder beliebige E/A-Bausteine (dies alles auch gemischt) untergebracht werden.

Speicheraufteilung in Segment 15:

Adresse (hexadezimal)

\$FFFF	I	-----	I
	I	Kernal ROM	I
	I	(Betriebssystem)	I
\$E000	I	-----	I
	I	E/A-Bereich	I
	I	(s. nächste Seite)	I
\$C000	I	-----	I
	I	BASIC-ROM	I
	I		I
\$A000	I	-----	I
	I	BASIC-ROM	I
	I		I
\$8000	I	-----	I
	I	Cartridge	I
	I	(Bank 3)	I
\$6000	I	-----	I
	I	Cartridge	I
	I	(Bank 2)	I
\$4000	I	-----	I
	I	Cartridge	I
	I	(Bank 1)	I
\$2000	I	-----	I
	I	4K Disk ROM	I
\$1000	I	-----	I
	I	2K Ext. Buffer RAM	I
\$0800	I	-----	I
	I	2K RAM	I
	I	-----	I
\$0001	I	Indirect Register	I
	I	-----	I
\$0000	I	Execute Register	I
	I	-----	I

E/A-Bereich

Adresse

\$E000	I	-----I	I
	I	TPI 6525 (Keyboard)	I
\$DF00	I	-----I	I
	I	TPI 6525 (IEEE, User)	I
\$DE00	I	-----I	I
	I	ACIA 6551 (RS 232 C)	I
\$DD00	I	-----I	I
	I	CIA 6526 (User, Interr)	I
\$DC00	I	-----I	I
	I	frei (Co-Proz.)	I
\$DB00	I	-----I	I
	I	SID 6581 (Sound)	I
\$DA00	I	-----I	I
	I	frei (Disk-E/A)	I
\$D900	I	-----I	I
	I	CRTC 6545 (Bildschirm)	I
\$D800	I	-----I	I
	I		I
	I	Bildschirmspeicher	I
	I		I
\$D000	I	-----I	I
	I		I
	I	unbenutzt	I
	I		I
\$C000	I	-----I	I

5 Bedienung (Tastatur) -----

Schauen Sie sich zunächst die Tastatur einmal genau an. Sie werden feststellen, daß die Tastatur in fünf zusammenhängende Blöcke unterteilt ist:

Betrachten wir zuerst den Block links oben, welcher aus zehn Tasten besteht. Auf diesen Tasten steht 'F1', 'F2', ..., 'F10'. Dies sind die Funktionstasten. Drücken Sie eine beliebige Taste dieser Reihe. Auf dem Bildschirm erscheint eine kurze Zeichenfolge. Alle zehn Funktionstasten sind mit solchen kurzen Zeichenfolgen belegt; es handelt sich dabei um diejenigen, welche erfahrungsgemäß am häufigsten benötigt werden.

Diese Funktionstasten können Sie jederzeit mit anderen Werten belegen, insbesondere mit jenen, die Sie selbst häufig verwenden. Wie dies zu erreichen ist, ist im Anhang ('KEY'-Kommando) beschrieben.

Zu bemerken ist noch, daß Ihr Rechner nicht über zehn, sondern über zwanzig Funktionstasten verfügt. Die Tasten 'F11' bis 'F20' erreichen Sie, wenn Sie die 'SHIFT'-Taste und die entsprechende Funktionstaste drücken. Z.B. ergibt 'SHIFT' und 'F4' die Taste 'F14'.

Der nächste Block sind die vier Tasten, mit denen Sie den Cursor über den Bildschirm bewegen können. Mit 'Cursor' wird das blinkende Zeichen auf Ihrem Bildschirm bezeichnet. Er steht an der Position, auf welcher das nächste eingegebene Zeichen ausgegeben wird.

Mit diesen vier Tasten können Sie nun den Cursor frei über den Bildschirm bewegen. Diese Cursor-Steuertasten haben 'Auto repeat'-Funktion; d.h., wenn diese Tasten länger als 0.5 Sekunden gedrückt werden, wird das Zeichen (hier: das Cursor-Steuersymbol) mit einer Geschwindigkeit von ca. 12 Zeichen pro Sekunde wiederholt. Probieren Sie es einfach aus!

Der nächste Block ist derjenige rechts oben. Er besteht ebenfalls aus vier Tasten. Diese vier Tasten müssen einzeln erläutert werden:

CLR / HOME:

Betätigen Sie diese Taste. Der Cursor springt in die linke obere Ecke des Bildschirms (Home-Position). Ansonsten bleibt der Bildschirm unverändert.

Betätigen Sie gleichzeitig die 'SHIFT'-Taste, so wird der ganze Bildschirm gelöscht. 'Gleichzeitig' bedeutet in diesem Zusammenhang: zuerst die 'SHIFT'-Taste drücken und diese niederhalten, bis die Taste 'CLR / HOME' gedrückt wurde.

OFF / RVS:

Drücken Sie diese Taste. Geben Sie anschließend irgendwelche Buchstaben oder Zahlen ein. Sie stellen fest, daß diese eingegebenen Zeichen in reverser Darstellung erscheinen.

Drücken Sie nun diese Taste in Verbindung mit der 'SHIFT'-Taste. Wenn Sie jetzt wieder irgendwelche Zeichen eingeben, so erscheinen diese wieder in normaler Darstellung.

NORM / GRAPH:

Mit dieser Taste können Sie zwischen dem Grafikmodus und dem Normalmodus umschalten. Im Normalmodus können Klein- und Großbuchstaben und alle anderen Zeichen verwendet werden, im Grafikmodus werden (nur auf dem Bildschirm!) Kleinbuchstaben in Großbuchstaben verwandelt und Großbuchstaben in Grafiksymbole. Alle anderen Zeichen bleiben unverändert.

Durch Drücken dieser Taste schalten Sie in den Grafikmodus um. Die zu den Tasten gehörenden Grafikzeichen sind auf der Vorderseite der Tasten eingraviert. Wollen Sie nun diese Zeichen eingeben, so müssen Sie unterscheiden, ob es sich um eine Buchstabentaste oder eine andere Taste handelt. Die Grafikzeichen der Buchstabentaste erreichen Sie durch gleichzeitiges Drücken der 'SHIFT'-Taste und der Buchstabentaste. Liegt das Grafikzeichen auf einer der anderen Taste, so ist diese Taste gleichzeitig mit der 'CONTROL'-Taste zu drücken.

Drücken Sie die 'SHIFT'-Taste und 'NORM / GRAPH' gleichzeitig, so kehren Sie in den Normalmodus zurück.

RUN / STOP:

Durch das Drücken dieser Taste wird ein gerade laufendes Programm unterbrochen. Wird diese Taste gleichzeitig mit der 'SHIFT'-Taste gedrückt, so wird das erste Programm, daß sich auf der Diskette im Laufwerk Null der Diskettenstation mit der Nummer 8 befindet, geladen und gleich gestartet.

Als nächstes schauen Sie sich den Tastenblock rechts unten an. Dies ist der 'Tischrechner'-Block. Sie sehen auf diesen Tasten die Ziffern, die vier Zeichen für die Grundrechenarten sowie vier weitere Tasten. Die Ziffern sind wohl klar (die Tasten mit den beiden Nullen erzeugt eben zwei Nullen). Die Taste mit dem Punkt dient als Dezimalpunkt (Überall in der EDV wird, wie im Englischen Sprachraum allgemein üblich, kein Dezimalkomma, sondern stattdessen der Dezimalpunkt verwendet.)

Die Taste mit dem Fragezeichen muß am Anfang jeder Eingabe stehen, wenn Sie die Tischrechnerfunktion benutzen wollen. Das Fragezeichen steht bei CBM-Computern als Kurzzeichen für die Anweisung 'print'.

Die Taste 'CE' löscht die letzte Eingabe, wenn Sie sich vertippt haben sollten. Die letzte eingetippte Zahl wird dadurch gelöscht.

Mit der Taste 'ENTER' bewirken Sie die Berechnung des von Ihnen eingegebenen Ausdrucks (In ihrer Funktion ist sie mit der 'RETURN'-Taste identisch).

Probieren Sie es einfach aus! Z.B.:

```
?3*5  ENTER
? 12 + 15 * 2  ENTER
usw.
```

Der letzte zu behandelnde Block ist derjenige links unten, den Sie wohl am meisten benötigen. Die Bedeutung der helleren Tasten dieses Blockes ist klar: Sie erzeugen das auf Ihnen eingravierte Zeichen (evtl. in Verbindung mit der 'SHIFT'-Taste). Die anderen Tasten bedeuten im einzelnen:

ESC:

Diese Taste leitet eine Sonderfunktion ein. Dazu ist zuerst diese Taste und dann eine andere Taste zu drücken. Die Bedeutung der diversen Sonderfunktionen ist bei der Editorbeschreibung erklärt.

TAB:

Mit dieser Taste können Sie Tabulatoren setzen, löschen oder Tabulatorpositionen anspringen. Wenn Sie diese Taste betätigen, wird die nächste gesetzte Tabulatorposition angesprungen. Einen Tabulator setzen Sie, indem Sie den Cursor auf die Position in einer Zeile bringen, an der Sie den Tabulator wünschen. Drücken Sie jetzt die Tasten 'SHIFT' und 'TAB', so wird an dieser Position ein Tabulator gesetzt. Das Löschen eines Tabulators geschieht auf die gleiche Art wie das Setzen: Positionieren Sie den Cursor auf die zu löschende Tabulatorposition und drücken Sie 'SHIFT' und 'TAB'. Der dort gesetzte Tabulator wird gelöscht.

'SHIFT LOCK':

Diese Taste dient zum Feststellen der Umschalttaste ('SHIFT')

'SHIFT':

Die beiden 'SHIFT'-Tasten sind die Umschalttasten. Mit ihnen können Sie Großbuchstaben (bzw. Grafikzeichen im Grafikmodus) sowie die Zeichen erreichen, die bei den zweireihig beschrifteten Tasten in der oberen Reihe erscheinen. Drücken Sie hierzu bitte zuerst (eine der beiden) 'SHIFT'-Tasten, halten Sie diese nieder und betätigen Sie die gewünschte Taste.

'CTRL':

Diese Taste ('Control'-Taste) hat mehrere Funktionen. Ähnlich wie die 'SHIFT'-Taste können hiermit weitere Zeichen (Grafik-

Zeichen, Steuerzeichen) erreicht werden. Drücken Sie z.B. einmal die 'CTRL'-Taste, halten diese nieder und drücken dann ein 'g'!

Ferner hat diese Taste die Aufgabe, Ausgaben, die auf den Bildschirm erfolgen, zu verlangsamen. Wenn Sie z.B. ein größeres Programm listen wollen, so findet die Bildschirmausgabe so schnell statt, daß Sie das Programm nicht mehr lesen können. Durch Drücken dieser Taste wird die Ausgabe so verlangsamt, daß dies wieder möglich wird.

'C=':

Diese Taste hat z.Z. nur eine Aufgabe: die Ausgabe auf den Bildschirm anzuhalten. Die Ausgabe kann durch Drücken einer beliebigen Taste fortgesetzt werden.

'RETURN':

Mit dieser Taste übergeben Sie die Information, die Sie eingegeben haben, zur Auswertung an den Rechner. Es wird nur jeweils die Zeile zur Auswertung übergeben, in welcher sich der Cursor befindet. Beachten Sie bitte dabei, daß eine 'Zeile' aus mehreren Bildschirmzeilen bestehen kann! Der Cursor springt beim Betätigen der 'RETURN'-Taste an den Anfang der nächsten Zeile. Wird diese Taste gleichzeitig mit der 'SHIFT'-Taste gedrückt, so springt der Cursor ebenfalls an den Anfang der nächsten Zeile, eine Übergabe zur Auswertung der Zeile findet jedoch nicht statt.

'INS / DEL':

Durch Drücken dieser Taste wird das Zeichen, auf dem der Cursor steht, gelöscht ('delete') und der Rest der Zeile um jeweils eine Position nach links verschoben. Wird sie gleichzeitig mit der 'SHIFT'-Taste betätigt, so wird die Zeile ab dem Zeichen, auf dem der Cursor steht, um eine Position nach rechts verschoben ('insert').

6 Systemstart

Die einzige Aktivität, die zum Systemstart notwendig ist, beschränkt sich auf das Einschalten Ihres Rechners. Das Betriebssystem und der BASIC-Interpreter sind in Speicherbausteinen enthalten, die ihre Information beim Ausschalten des Rechners nicht verlieren. Sie können also sofort mit dem Eingeben oder dem Laden eines Programmes beginnen.

7 Abschließen der Arbeit mit dem Rechner

Das Beenden der Arbeit ist genauso einfach wie der Anfang:
Schalten Sie den Rechner aus.

Vergewissern Sie sich jedoch vorher, daß Sie Ihre Daten
oder Ihr Programm auf Disketten gesichert haben, wenn Sie diese
später noch benötigen!

8 Allgemeines über Datenträger (Disketten)

8.1 Behandlung der Disketten

Die am meisten verwendeten Datenträger sind bei Commodore-Rechnern die Disketten. Wenn Sie die folgenden Punkte beachten, werden Sie keine Probleme mit Ihren Disketten haben:

1. Wenn Sie Disketten nicht benötigen, stecken Sie sie in die zu jeder Diskette gehörende Diskettentasche. Dort sind sie am sichersten aufgehoben.
2. Bewahren Sie Ihre Disketten bei normalen Klimabedingungen auf. Große Hitze oder hohe Luftfeuchtigkeit vertragen sie genauso wenig wie Ihre Schallplatten.
3. Vermeiden Sie mechanische Beschädigungen oder Kontakt mit Flüssigkeiten. Berühren Sie die Diskette auch nie an dem etwa drei Zentimeter langen und ein Zentimeter breiten Schlitz!
4. Vermeiden Sie auch die Nähe von starken Magnetfeldern!

Und vor allem:

8.2 Datensicherung

5. Fertigen Sie Kopien von Ihren Disketten an!

Es ist schnell passiert: Eine Kaffetasse fällt um, Ihr Sohn testet mit seinem neuen Magneten, ob Disketten magnetisch sind, die Sonne kommt während der Mittagspause hinter den Wolken hervor und erwärmt gleichermaßen Ihr Herz und Ihre Disketten auf der Fensterbank, und, und, und...

Durch das 'BACKUP'-Kommando (s. Anhang, BASIC-Kommandos) ist das Anfertigen einer 1:1-Kopie Ihrer Disketten einfach und problemlos möglich. Während dieser Kopiervorgang läuft, können Sie an Ihrem Rechner weiterarbeiten, sofern Sie keinen Kontakt mit Peripheriegeräten benötigen. Das Kopieren dauert je nach Typ der Diskettenstation bis zu ca. 12 Minuten.

8.3 Initialisieren neuer Disketten

Wenn Sie neue Disketten kaufen, sind diese noch nicht formatiert, d.h., Ihre Diskettenstation kann mit ihnen noch nichts anfangen. Deshalb muß diese Diskette zuerst formatiert werden. Dies geschieht am einfachsten mit dem 'HEADER'-Kommando (s. Anhang, BASIC-Kommandos). Danach können Sie die Disketten ganz normal benutzen.

Wenn Sie alte Disketten nicht mehr benötigen (sind Sie sicher ?) dann können Sie diese behandeln wie fabrikneue Disketten, d.h., mit dem 'HEADER'-Kommando bearbeiten. Die 'alten' Daten sind danach unwiederbringlich gelöscht.

9 Programmentwicklung

Wie ist nun die Vorgehensweise bei der Programmentwicklung? Gehen wir einmal vom 'Rohzustand' Ihrer Anlage aus: Sie haben einen Rechner, eine Diskettenstation und fabrikneue Disketten. Das erste, daß Sie tun müssen, ist, die Diskette zu formatieren. Fabrikneue Disketten sind in der Regel noch nicht formatiert. Mit solchen Disketten kann die Diskettenstation noch nichts anfangen; sie muß zunächst die Spuren auf der Diskette anlegen, in welche später die Informationen eingetragen werden, sowie das Inhaltsverzeichnis (das sich ebenfalls auf der Diskette befinden muß) anlegen. Um eine Diskette zu formatieren dient das Kommando 'header'. Legen Sie nun eine fabrikneue Diskette in das rechte Laufwerk Ihrer Diskette ('drive 0') ein und schließen Sie die Laufwerksklappe. Geben Sie nun an Ihrem Rechner folgendes ein:

```
header d0,"test.1",i01
```

und drücken Sie die RETURN-Taste. Ihr Computer stellt die Frage:
are you shure?

Geben Sie jetzt 'y' (oder 'yes') und die RETURN-Taste ein, so beginnt die Diskettenstation nun mit dem Formatieren. Das Knattergeräusch am Anfang braucht Sie nicht zu irritieren, es ist normal. Bei einer anderen Eingabe als 'y' (oder 'yes') wird das Kommando nicht ausgeführt. Während die Diskettenstation nun arbeitet, lesen Sie bitte im Anhang, Kapitel 3.1.20, nach, welche Bedeutungen die einzelnen Angaben im obigen 'header'-Kommando haben.

Sobald die Formatierung abgeschlossen ist (das dauert nur wenige Minuten), können Sie mit der Eingabe eines Programmes beginnen.

Geben Sie z.B. folgendes kleine Programm ein:

```
10 print "programm 1"  
20 end
```

Geben Sie nun das Kommando 'run', so wird dieses Programm ausgeführt: der Text 'programm 1' erscheint auf dem Bildschirm.

Sichern Sie nun dieses Programm auf die Diskette mit folgendem Kommando:

```
dsave "prog.1"
```

Dieses Programm ist nun auf der Diskette (und natürlich noch in Ihrem Rechner). Sie können es nun jederzeit mit dem Kommando

```
dload "prog.1"
```

wieder in Ihren Rechner holen.

Ändern Sie nun die Zeile 10:

```
10 print "programm 2"
```

und sichern Sie das neue Programm mit folgendem Kommando:

```
dsave "zweites Programm"
```

Geben Sie dieses Kommando (zu Versuchszwecken) ein zweites Mal. Sie werden anschließend feststellen, daß an Ihrer Diskettenstation die Fehlerlampe leuchtet. Geben Sie jetzt die Anweisung:

```
print ds$
```

an Ihren Rechner. Durch diese Anweisung teilt Ihnen der Computer mit, was ihm nicht gefallen hat; in diesem Fall:

```
63, file exists,00,00,0
```

Das bedeutet, daß die Datei mit dem Namen "zweites Programm" zu dem Zeitpunkt, als Sie das entsprechende 'dsave'-Kommando eingaben, bereits existierte. Bei dieser zweiten Eingabe wurde das Programm nicht auf die Diskette übertragen. Auf diese Art ist gewährleistet, daß Sie nicht aus Versehen ein Programm überschreiben, das sich bereits unter dem angegebenen Namen auf der Diskette befindet.

Das bedeutet nun aber nicht, daß ein Programm für alle Zeiten auf der Diskette liegt. In der Regel ist es so, daß Sie ein bereits geschriebenes Programm, das sich auf einer Diskette befindet, leicht modifizieren und dieses erneuerte Programm unter dem gleichen Namen wieder auf der Diskette ablegen wollen. Zu diesem Zweck müssen Sie im 'dsave'-Kommando unmittelbar vor dem Namen des Programms das Zeichen '\$' (Klammeraffe, auf der Tastatur über der '2') angeben. Das obige Kommando würde dann so aussehen:

```
dsave "$zweites Programm"
```

Hiermit würde das bereits bestehende Programm überschrieben werden (ohne Fehlermeldung).

Mit diesen wenigen Kommandos sind Sie nun in der Lage, eigene BASIC-Programme zu schreiben und diese auf Disketten abzulegen, um sie irgendwann später wieder ausführen oder verändern zu können.

10 Der Editor

Zunächst einmal: was ist der Editor?

Ein Editor ist ein Programm, mit dem Sie Programme oder sonstige Anweisungen in Ihren Computer eingeben können. Er bereitet Ihre Eingaben so auf, wie Sie Ihr Rechner am besten verstehen kann.

Wie bringen Sie nun den Editor zum Laufen?

Überhaupt nicht. Er läuft von selbst. D.h., immer dann, wenn kein Programm abläuft, ist der Editor aktiv.

Wie arbeitet der Editor?

Der Editor ist bildschirmorientiert. Das bedeutet, Sie können Ihren Cursor auf eine beliebige Zeile des Bildschirms positionieren, in dieser Zeile etwas eingeben (oder verändern) und diese Information an den Rechner übergeben. Diese Übergabe an den Rechner geschieht mit der 'RETURN'-Taste oder mit der 'ENTER'-Taste. In der Funktion sind diese beiden Tasten identisch.

An dieser Stelle sollte nun der Begriff der 'Zeile' genauer erklärt werden. Es gibt zunächst einmal die 'Bildschirmzeile'. Sie besteht aus genau 80 Zeichen, beginnt in der ersten Spalte des Bildschirms und endet in der achtzigsten Spalte. (Intern werden die Spalten von Null beginnend durchnummeriert; die letzte Spalte hat also die Nummer 79, nicht 80!). Eine allgemeine Zeile kann sich jedoch über mehrere Bildschirmzeilen erstrecken. So kann z.B. eine Programmzeile bis zu 160 Zeichen lang sein. Theoretisch kann eine Zeile den ganzen Bildschirm, also alle 2000 Zeichen, belegen. Der Editor wertet nun immer die ganze Zeile, in welcher der Cursor steht, aus. Ihr Rechner hat sich gemerkt, in welcher Bildschirmzeile die gerade zu verarbeitende Zeile beginnt und in welcher sie endet.

Durch Drücken der 'RETURN'-Taste wird nun die Zeile, in welcher sich der Cursor befindet, zur Auswertung übergeben. Woran erkennt der Editor nun, ob eine Anweisung an den Rechner gegeben wird oder ob eine Programmzeile eingegeben werden soll? Ganz einfach daran, ob die Zeile mit einer Zahl beginnt oder nicht. Programmzeilen beginnen IMMER mit einer Zeilennummer, eine Anweisung an den Rechner beginnt NIE mit einer Zahl.

Was passiert nun, wenn Sie zwar Änderungen in einer Zeile vornehmen, diese aber nicht mit 'RETURN' zur Auswertung übergeben, sondern die Zeile anderweitig verlassen (z.B. Cursor-Steuertasten)? Auf dem Bildschirm wird die Änderung zwar dargestellt, im Rechner selbst wurde jedoch nichts geändert.

Der Editor selbst bietet sehr umfangreiche Möglichkeiten. So

können Sie z. B. die Bildschirmgröße definieren, sich e i b g
P o r m z i e a z i e l s e n, die Ausgabe verlangsamen, unter-
brechen oder abbrechen, Zeilen(bereiche) löschen und vieles mehr.
Die meisten dieser Funktionen sind über die 'ESC'ape-Taste erreichbar.
Die meisten dieser Funktionen sind im Anhang 1 aufgeführt,
der andere Teil wurde bereits bei der Tastaturerklärung erläutert.

Hier noch einmal die wesentlichen Funktionen für den Editor:

CTRL-Taste: Ausgabe verlangsamen
C= -Taste: Ausgabe anhalten
INS/DEL : Zeichen einfügen / löschen
CLR/HOME : Bildschirm löschen / Cursor in linke obere Ecke
OFF/RVS : Normaldarstellung / Reversdarstellung
NORM/GRAPH: Normaler Zeichensatz / Grafik-Zeichensatz
TAB : Tabulator setzen / löschen / anspringen
Cursorsteuertasten: Cursorbewegungen
Funktionstasten: individuell verwendbar

ESC-Taste : Einleiten Sonderfunktion

ESC A: Automatisches Einfügen
ESC B: Bildschirmfenster definieren (rechte untere Ecke)
ESC C: Automatisches Einfügen löschen
ESC D: Zeile löschen
ESC E: Cursor nicht-blinkend setzen
ESC F: Cursor blinkend setzen
ESC G: Ausgabe des Klingelzeichens erlauben
ESC H: Ausgabe des Klingelzeichens unterbinden
ESC I: Zeile einfügen
ESC J: An den Zeilenanfang springen
ESC K: An das Zeilenende springen
ESC L: Bildschirmrollen erlauben
ESC M: Bildschirmrollen unterbinden
ESC N: Normale Bildschirmdarstellung
ESC O: Einfüge- und Reversmodus aufheben
ESC P: Zeilenanfang löschen
ESC Q: Zeilenerde löschen
ESC R: Reverse Bildschirmdarstellung
ESC S: Cursor ist invertiertes Leerzeichen
ESC T: Bildschirmfenster definieren (linke obere Ecke)
ESC U: Cursor ist Unterstreichungsstrich
ESC V: Bildschirm nach oben rollen
ESC W: Bildschirm nach unten rollen
ESC X: Escape-Modus ohne Wirkung abbrechen
ESC Y: Normalen Zeichensatz einstellen
ESC Z: Alternativen Zeichensatz einstellen

1 Editor ESC-Funktionen

Die ESC (Escape)-Funktionen erreichen Sie, indem Sie nacheinander die ESC-Taste und dann den entsprechenden Buchstaben drücken. Die Buchstaben wurden, soweit es möglich war, so gewählt, daß ein Zusammenhang mit der Funktion hergestellt werden kann. Dies konnte jedoch leider nicht generell durchgeführt werden, da alle 26 Buchstaben des (englischen) Alphabets für die 26 ESC-Funktionen benötigt werden.

Noch etwas ist zu bemerken: Befinden Sie sich im sogenannten 'Quote-Mode', dann hebt die ESC-Taste (ohne zusätzlichen Buchstaben) diesen Modus wieder auf. Dieser Modus wird jedesmal dann eingestellt, wenn in einer Zeile eine ungerade Anzahl von Gänsefüßchen oben (") eingegeben wurde. Er ist dadurch gekennzeichnet, daß (wie im Insert-Mode) die Cursor-Steuerzeichen nicht ausgeführt werden, sondern ihr Codewert ausgegeben wird. Durch das Drücken der ESC-Taste wird diese Anzahl auf Null zurückgesetzt.

ESC A ("A"utomatic Insert)

Automatisches Einfügen. Wird der Cursor auf dem Bildschirm in eine Position innerhalb irgendeines Textes gebracht und dann neuer Text eingegeben, so wird der alte Text nicht, wie im Normalfall, überschrieben, sondern der neue Text wird vor dem Zeichen, auf dem der Cursor steht, eingefügt. Der Rest des bisherigen Textes rückt dabei bei jedem neuen Zeichen um eine Position nach rechts. Durch 'ESC C' wird dieser Zustand wieder aufgehoben.

ESC B (Set "B"ottom)

Die rechte untere Ecke eines Bildschirmfensters wird gesetzt. Das bedeutet, daß alle Ausgaben auf den Bildschirm sich auf einen Bereich beschränken, der durch die linke obere Ecke und der hiermit erklärten rechten unteren Ecke definiert ist. Der Rest des Bildschirms bleibt unverändert. Durch zweimaliges Drücken der 'HOME'-Taste wird wieder der ganze vorhandene Bildschirm als 'Fenster' eingestellt (Normalzustand). S. auch 'ESC T'.

ESC C ("C"ancel Automatic Insert)

Hiermit wird der durch 'ESC A' erreichte Zustand aufgehoben.

ESC D ("D"elete Line)

Die Zeile, in welcher der Cursor gerade steht, wird (auf dem Bildschirm) gelöscht. Umfaßt die Zeile mehr als eine Bildschirmzeile, so verschwindet die ganze Zeile. Der Rest bis zum Bildschirmende wird nach oben nachgerückt. Der Cursor steht am Anfang der Zeile, welche nun die Position der "alten" Zeile einnimmt.

ESC E ("E"verlasting Cursor)

Der Cursor blinkt nicht, er ist ständig sichtbar. Siehe auch die nächste ESC-Funtion.

ESC F ("F"lashing Cursor)

Der Cursor erscheint blinkend (Normalfall). Siehe auch vorherige ESC-Funktion.

ESC G (Enable Bell; Glocke ist 'Control G')

Im Bedarfsfall ertönt die Glocke (z.B. Zeile länger als 69 Zeichen) (Normalfall). Siehe auch nächste ESC-Funktion.

ESC H (Disable Bell)

Die Glocke wird stummgeschaltet. Siehe auch vorherige ESC-Funktion.

ESC I ("I"nsert Line)

Auf dem Bildschirm wird in der Zeile, in welcher sich der Cursor befindet, eine Leerzeile eingefügt. Der Rest bis zum Bildschirmende wird dabei um jeweils eine Zeile nach unten verschoben (beginnend in der Zeile, in der der Cursor sich gerade befindet). Die letzte Bildschirmzeile geht dabei verloren. Der Cursor steht am Anfang der neu eingefügten Zeile.

ESC J (Move to Start of Line)

Der Cursor springt an den Anfang der Zeile, in der er sich gerade befindet.

ESC K (Move to End of Line)

Der Cursor springt an das Ende der Zeile, in der er sich gerade befindet. Genauer: Er springt auf das letzte Zeichen der Zeile.

ESC L (Enable Scrolling)

Befindet sich der Cursor auf der letzten Bildschirmzeile und es erscheinen Ausgaben (vom Programm oder Betriebssystem), so wird bei jeder Ausgabe der Bildschirm um eine Zeile nach oben verschoben. Die oberste Zeile verschwindet vom Bildschirm. Siehe auch nächste ESC-Funktion.

ESC M (Disable Scrolling)

Ist der Bildschirm bis zur letzten Zeile beschrieben und stehen noch weitere Ausgaben an, so wird die Ausgabe am Anfang des Bildschirms fortgeführt. Der bisherige Inhalt wird dabei überschrieben. Siehe auch vorherige ESC-Funktion.

ESC N ("N"ormal Screen)

Der Bildschirm erscheint wieder 'normal', d.h., die durch 'ESC R' erreichte Invertierung des Bildschirms wird rückgängig gemacht.

ESC O (Cancel Insert and Reverse Mode)

Diese Funktion hebt die Reversdarstellung der einzutippenden Zeichen auf. Befindet sich der Cursor auf einer Position, welche durch die 'INS'-Taste freigemacht wurde, so werden Cursor-Steuerzeichen nicht ausgeführt, sondern der Code des entsprechenden Zeichens ausgegeben. 'ESC O' hebt diesen Zustand auf.

ESC P (Erase to Start of Line)

Der Anfang der Zeile, in welcher der Cursor gerade steht, wird bis einschließlich dem Zeichen, auf dem der Cursor steht, durch Leerzeichen überschrieben (die ganze Zeile ist hier gemeint, was bei langen Zeilen nicht mit der Bildschirmzeile identisch sein muß!).

ESC Q (Erase to End of Line)

Der Rest der Zeile, in welcher der Cursor gerade steht, wird ab einschließlich dem Zeichen, auf dem der Cursor steht, gelöscht. Auch hier ist die ganze Zeile (nicht unbedingt die Bildschirmzeile) gemeint.

ESC R ("R"everse Screen)

Der Darstellung aller Zeichen auf dem Bildschirm wird invertiert (dunkle Schrift auf hellem Grund). Siehe auch 'ESC N'.

ESC S ("S"olid Cursor)

Der Cursor wird durch ein invertiertes Leerzeichen dargestellt (Normalfall). Siehe auch 'ESC U'.

ESC T (Set "T"op of Screen)

Analog zu 'ESC B' wird mit dieser ESC-Funktion die linke obere Ecke des Bildschirmfensters definiert. Eingaben und Ausgaben beschränken sich auf den durch die linke obere und die rechte untere Ecke definierten Bildschirmausschnitt. Der Rest des Bildschirms bleibt unverändert. Durch zweimaliges Drücken der 'HOME'-Taste (hintereinander!) wird der Normalzustand wieder hergestellt. Von einem Programm aus kann dies durch zweimaliges Ausgeben von 'chr\$(19)' erreicht werden.

ESC U ("U"nderline Cursor)

Der Cursor wird durch einen Unterstreichungsstrich dargestellt. Wegen der unterschiedlichen Art der Zeichendarstellung bewirkt diese Eingaben bei den Rechnern der Serie CBM 600, daß der Cursor verschwindet. Der Bereich, in dem der Cursor in diesem Falle erscheinen würde, ist auf einem der europäischen Fernsehnorm entsprechenden Monitor bei 25 Bildschirmzeilen nicht unterzubringen.

ESC V (Scroll up)

Der ganze Bildschirm wird um eine Bildschirmzeile nach oben verschoben. Die oberste Bildschirmzeile geht dabei verloren, unten wird eine Leerzeile erzeugt.

ESC W (Scroll down)

Der ganze Bildschirm wird um eine Bildschirmzeile nach unten verschoben. Die unterste Bildschirmzeile geht dabei verloren, oben wird eine Leerzeile erzeugt.

ESC X (Cancel Escape Sequence)

Hebt die Funktion einer versehentlich eingegebenen ESC-Taste auf.

ESC Y (Normal Character Set)

Schaltet den Normalen Zeichensatz ein. Siehe auch 'ESC Z'.

ESC Z (Alternate Character Set)

Schaltet den alternativen Zeichensatz ein. Siehe auch 'ESC Y'.

2 Systemvariablen

2.0 Vorbemerkungen

Die Systemvariablen kann man in 3 Gruppen unterteilen:

1. Statusvariablen
2. Zeitvariable
3. Fehlervariablen

Die Statusvariablen geben Auskunft darüber, wie der Verkehr mit Ein- und Ausgabegeräten abläuft.

Die Zeitvariable gibt die Zeit an, die seit dem Einschalten des Rechners verstrichen ist.

Fehlervariablen geben Auskunft über aufgetretene Fehler, um deren Ursache leichter erkennen und beheben zu können.

2.1 Statusvariablen

Es gibt drei Variablen, die über den Status des Verkehrs mit E/A-Geräten Auskunft geben: ST, DS und DS\$

2.1.1 ST

Diese Variable wird als die eigentliche Statusvariable bezeichnet. Nach jeder E/A-Operation mit einem Peripheriegerät wird ihr Wert neu festgelegt. Grundsätzlich ist ihr Verhalten von der verwendeten Schnittstelle bestimmt. Im Falle der RS 232 C-Schnittstelle ist es im Anhang 5 (RS 232 C) beschrieben. Die hier geliefert Beschreibung bezieht sich auf den Datenverkehr über die IEC-Schnittstelle.

Die Statusvariable ST kann vier verschiedene Werte annehmen. Sie bedeuten im einzelnen:

ST= 1: Zeitüberschreitung beim Lesen. Dieser Zustand wird erreicht, wenn von einem Peripheriegerät Werte gelesen werden sollen und das Peripheriegerät nicht innerhalb von ca. 64 Millisekunden antwortet.

ST= 2: Wie ST=1, jedoch beim Schreiben.

timeout:

Manche Geräte für den IEC-Bus, insbesondere Meßgeräte, sind zu 'langsam' für diese Zeitspanne. Durch den Befehl 'poke 862,128'

kann dieses 'timeout'-Verhalten abgeschaltet werden. Antwortet ein danach angesprochenes Gerät jedoch nicht, so kann der Rechner nur durch Drücken des 'RESET'-Knopfes auf der Gehäuserückseite wieder rückgesetzt werden. Entsprechend kann durch 'poke 862,0'

das normale 'timeout'-Verhalten wiederhergestellt werden.

ST= 64: Dateiende wurde erreicht. Weiterlesen ergibt ST=2, als Zeichen wird jeweils chr\$(13) (Carriage Return) zurückgeliefert.

ST=-128: Das angesprochene Gerät existiert nicht.

2.1.2 DS (Disk-Status)

Diese Variable gibt genauere Informationen zum Status des Verkehrs mit Diskettenstationen. Sie wird bei jedem Datenverkehr mit einer Diskettenstation aktualisiert. Eine genaue Beschreibung der Bedeutung der Werte, die diese Variable annehmen kann, ist in Ihrem Disketten-Handbuch genau beschrieben. Schauen Sie bitte dort nach.

2.1.3 DS\$ (Disk-Status)

Diese Variable ist in Ihrer Funktion mit der Variablen DS identisch. Sie liefert jedoch zusätzlich zu der Fehlernummer 'DS' noch den Text, der zu dieser Fehlernummer gehört. Ferner werden noch Spur-, Sektornummer und evtl. Diskettenseite angegeben, wo der Fehler auftrat. Diese Variable ist ebenfalls genau in Ihrem Disketten-Handbuch beschrieben.

2.2 Zeitvariable TI\$

Diese Variable gibt die Zeit an, die seit dem Einschalten des Rechners vergangen ist. Sie kann auch durch eine explizite Wertzuweisung verändert werden. Diese Zuweisung muß aus einem String bestehen, der aus genau sieben Ziffern auf e a t 's'. Die 'Z'i' wird dann mit dem hiermit angegebenen Wert weitergezählt. Die sieben Ziffern haben folgende Bedeutung:

Die beiden ersten Ziffern geben die Stunden an, die beiden nächsten die Minuten und wiederum die nächsten die Sekunden. Die letzte Ziffer repräsentiert die Zehntel-Sekunden.

Bei einer Abfrage dieser Variablen werden auch diese sieben Ziffern (als String) zurückgeliefert.

Anmerkung: Die bei älteren Commodore-Rechnern vorhandene Zeitvariable 'TI' existiert nicht mehr.

2.3 Fehlervariablen

Es gibt zwei Fehlervariablen, die Ihnen das Abhandeln von Fehlern erleichtern: 'EL' und 'ER'

2.3.1 EL (Fehlerzeile)

Tritt in einem Programm ein Fehler auf, so wird die Zeilennummer, in der der Fehler auftrat, in dieser Variablen abgelegt. Ihre Fehlerbehandlungsprozedur kann dies entsprechend auswerten.

2.3.2 ER (Fehlernummer)

In dieser Variablen wird die Nummer des aufgetretenen Fehlers abgelegt. Auch sie kann in einer Fehlerbehandlungsprozedur ausgewertet werden. Sie kann folgende Werte annehmen:

Nummer	Bedeutung
0	Stop-Taste wurde gedrückt.
1	Das Inhaltsverzeichnis der angesprochenen Diskette ist bereits voll.
2	Die angegebene logische Gerätenummer wird bereits verwendet.
3	Die angegebene logische Gerätenummer wurde nicht eröffnet.
4	Die angegebene Datei wurde nicht gefunden.
5	Das angesprochene Gerät existiert nicht.
6	Das angesprochene Gerät kann nur beschrieben werden.
7	Das angesprochene Gerät kann nur gelesen werden.
8	Die Angabe des Dateinamens fehlt.
9	Unzulässige Gerätenummer
11	Diskette defekt (zerstört)
20	Eine 'NEXT'-Anweisung wurde durchlaufen, zu der keine 'FOR'-Anweisung gehört.
21	Syntaxfehler: Im Programmtext steht etwas unverständliches.
22	Eine 'RETURN'-Anweisung wurde erreicht, ohne daß eine 'GOSUB'-Anweisung erreicht wurde.
23	Es standen für eine 'READ'-Anweisung keine Daten mehr zur Verfügung.
24	Ein Wert ist außerhalb des zulässigen Wertebereichs.
25	Eine Zahl überschreitet den zulässigen Zahlenbereich (overflow)
26	Kein Speicherplatz mehr frei
27	Angegebene Zeilennummer existiert nicht.
28	Fehlerhafter Index (zu groß, negativ)
29	Feld bereits definiert
30	Division durch Null
31	Anweisung nicht im Direktmodus ausführbar
32	keine Typenkompatibilität (z.B. Strings und Zahlen)
33	String zu groß
34	Fehlerhafte Daten beim Einlesen von Datei
35	Ausdruck zu komplex
37	Funktion nicht definiert
38	Fehler bei 'LOAD'
39	Fehler bei 'VERIFY'
40	Stackbereich voll
41	'RESUME' kann nicht ausgeführt werden.
42	'DISPOSE' kann nicht ausgeführt werden.

Die Funktion 'ERR\$' liefert die Fehlermeldung im Klartext, die zu dem Fehler mit der als Parameter angegebenen Nummer gehört.

'print err\$(41)'

bewirkt z.B. die Ausgabe der Meldung 'unable to resume'.

3 Extended Basic 4.0

3.0 Vorbemerkungen

In diesem Kapitel werden sämtliche Kommandos, Statements und Funktionen, die Ihnen bei der Systemfamilie CBM 600/700 zur Verfügung stehen, erklärt. Innerhalb der einzelnen Gruppen (Kommandos, Statements, Funktionen) werden die zugehörigen Elemente in alphabetischer Reihenfolge erklärt.

Es ist nicht immer sehr einfach, zu bestimmen, zu welcher der Gruppen eine bestimmte Anweisung gehört, insbesondere bei den Gruppen Kommandos bzw. Statements. Sollte daher in einer Gruppe eine bestimmte Anweisung vermißt werden, so sollte sie in einer der anderen Gruppen gesucht werden.

Format der Erklärungen:

Format: Zeigt und erklärt die genaue, syntaktisch richtige(n) Form(en), in welcher die Anweisung anzugeben ist. Zur ausführlichen Erklärung der dabei verwendeten Beschreibungsmethode siehe unten.

Modus: Manche Anweisungen können nur im Programm ordnungsgemäß funktionieren (Programm-Modus), andere hingegen nur im Grundzustand des Rechners (Direkt-Modus); manche können aber auch in beiden Modi laufen. Im letzteren Fall ist wiederum zu unterscheiden, ob die Anweisung im Programm-Modus die gleiche Wirkung hat wie im Direkt-Modus oder nicht. Diese eventuell vorhandenen Unterschiede werden in der Beschreibung erläutert.

Zweck: Hierin wird die Aufgabe der Anweisung erklärt.

Voraussetzung: Es werden die Voraussetzungen, um das richtige Funktionieren der beschriebenen Funktion zu gewährleisten, aufgeführt.

Beschreibung: In diesem Abschnitt erfolgt eine genaue Beschreibung der Anweisung (Handhabung, Wirkung usw.).

Defaultwerte: Es werden die Werte, die das System für nicht angegebene Parameter einsetzt, aufgeführt.

Wertebereich: Die möglichen Werte der Parameter werden hier erklärt.

Abkürzung: Zu (fast) jedem Befehl existiert eine Kurzschreibweise, welche hier aufgeführt ist.

Beispiel: Zeigt ein kurzes Programm(stück), welches die Anwendung und Wirkungsweise verdeutlicht.

Format-Beschreibung

Es gelten folgende Regeln:

1. Zeichen, welche in Großbuchstaben geschrieben sind, sind so wie angegeben zu verwenden. (Im Basic-Text sind jedoch Kleinbuchstaben einzusetzen!) Eine Sonderstellung nehmen jedoch die Abkürzungen dabei ein: Sie werden so wie angegeben geschrieben, z.B. 'bA' (kleines 'b', großes 'A') für 'BACKUP'.
2. Zeichen, die in Kleinbuchstaben geschrieben sind, müssen beim Aufruf durch aktuelle Werte ersetzt werden.
3. Angaben, welche in eckigen Klammern stehen, sind optional; d.h., wenn sie fehlen, wird ein Standardwert für Sie eingesetzt.
4. Angaben, die in runden Klammern stehen, können mehrmals hintereinander angegeben werden, solange die Eingabezeile maximal 160 Zeichen lang ist.
5. Gibt es für einen Parameter mehrere alternativen Schreibweisen, so sind sie in mehreren Zeilen aufgeführt.
6. Alles, was hinter einem Schrägstrich erscheint, ist als Kommentar zu werten, der sich nur auf diese Zeile bezieht.
7. Alle anderen Zeichen sind so zu übernehmen, wie sie angegeben sind.

Generell ist zu sagen: Die einzelnen Parameter eines Kommandos bzw. eines Befehls sind durch Kommata zu trennen. Wenn diese entfallen können, ist dies in der Formatbeschreibung ausdrücklich vermerkt. Direkt hinter einem Kommando darf nie ein Komma stehen !

Der Leser, dem das alles etwas unheimlich erscheint, möge in diesem Kapitel wahllos einige Beschreibungen aufschlagen, sich diese genau ansehen und mit den jeweiligen Beispielen vergleichen. Er wird in kurzer Zeit verstehen, was mit dem oben gesagten gemeint ist.

Inhaltsübersicht

(F=Funktion, S=Statement, K=Kommando)

abs	(F)	133
Addition	(F)	118
and	(F)	134
append	(K)	8
asc	(F)	135
atn	(F)	136
backup	(K)	10
bank	(K)	12
bload	(K)	13
bsave	(K)	15
catalog	(K)	17
chr\$	(F)	137
close	(K)	19
clr	(S)	66
cmd	(K)	20
collect	(K)	22
concat	(K)	24
cont	(K)	26
copy	(K)	27
cos	(F)	138
data	(S)	68
dclear	(K)	29
dclose	(K)	31
def ...	(S)	70
delete	(K)	33
dim	(S)	72
directory	(K)	35
dispose	(S)	74
Division	(F)	121
dload	(K)	37
dopen	(K)	39
dsave	(K)	41
else	(S. if ...)	(84)
end	(S)	76
err\$	(F)	139
exp	(F)	140
fn	(S. def ...)	(70)
for ...	(S)	77
fre	(F)	141
get	(S)	79
get#	(S)	81
Gleich	(F)	125
gosub	(S, s.a. on ...)	82
goto	(S, s.a. on ...)	83
Größer	(F)	126
Größer gl.	(F)	131

header	(K)	43
if ...	(S)	84
input	(S)	86
input#	(S)	88
instr	(F)	142
int	(F)	143
key	(K)	45
Kleiner	(F)	123
Kleiner gl.	(F)	128
left\$	(F)	144
len	(F)	145
let	(S)	90
list	(K)	46
load	(K)	48
log	(F)	146
mid\$	(F)	147
Multiplik.	(F)	120
new	(K)	50
next	(s. for ...)	92
not	(F)	149
on ...	(S)	94
open	(K)	51
or	(F)	150
peek	(F)	151
poke	(S)	96
pos	(F)	152
Potenzier.	(F)	122
print (...)	(S)	97
print#(...)	(S)	101
pundef	(S)	102
read	(S)	104
record	(K)	55
rem	(S)	106
rename	(K)	57
restore	(S)	107
resume	(S)	108
return	(S)	110
right\$	(F)	153
rnd	(F)	154
run	(K)	59
save	(K)	60
scratch	(K)	62
sgn	(F)	156
sin	(F)	157
spc((F)	158
sqr	(F)	159
step	(s. for ...)	(77)
stop	(S)	111
str\$	(F)	160
Subtraktion	(F)	119
sys	(S)	112
tab((F)	161
tan	(F)	162

then	(s. if ...)(84)
to	(s. for ...)(77)
trap	(S) 113
Ungleich	(F) 130
using	(s. print ..., print# ...)(97,101)
usr	(F) 163
val	(F) 164
verify	(K) 64
wait	(S) 115

3.1 Kommandos

Inhalt

3.1.1	append	8
3.1.2	backup	10
3.1.3	bank	12
3.1.4	bload	13
3.1.5	bsave	15
3.1.6	catalog	17
3.1.7	close	19
3.1.8	cmd	20
3.1.9	collect	22
3.1.10	concat	24
3.1.11	cont	26
3.1.12	copy	27
3.1.13	dclear	29
3.1.14	dclose	31
3.1.15	delete	33
3.1.16	directory	35
3.1.17	dload	37
3.1.18	dopen	39
3.1.19	dsave	41
3.1.20	header	43
3.1.21	key	45
3.1.22	list	46
3.1.23	load	48
3.1.24	new	50
3.1.25	open	51
3.1.26	record	55
3.1.27	rename	57
3.1.28	run	59
3.1.29	save	60
3.1.30	scratch	62
3.1.31	verify	64

3.1.1 APPEND

Format: APPEND p1 p2 [p3] [p4]
 p1: # dateinummer
 p2: dateiname
 p3: D drive
 p4: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Erweitern einer bestehenden sequentiellen Datei.

Voraussetzung: Die angegebene Datei existiert. Sie muß vom Typ 'seq', 'prg' oder 'usr' sein.

Beschreibung: Die Datei 'dateiname' auf dem Laufwerk 'drive' des Gerätes mit der Adresse 'unit' wird zum Schreiben eröffnet. Die auf diese Datei auszugebende Information wird dabei hinter die bereits existierenden Daten angefügt.

Die logische Gerätenummer, unter der fortan die Datei angesprochen wird, ist in p1 zu deklarieren. Der Name der Datei wird im Parameter p2 angegeben, das Laufwerk im Parameter p3. Die Gerätenummer (Primäradresse) ist in p4 anzugeben. Werden bei den Parametern p2, p3 oder p4 keine Konstanten, sondern Variablen oder Ausdrücke angegeben, so sind sie in runde Klammern einzuschließen.

Die Reihenfolge der Parameter ist beliebig.

Defaultwerte: drive: 0
 unit: 8

Wertebereich: dateinummer: 1 bis 255 (ganzzahlig)
 dateiname: String der Länge 1 bis 16
 drive: 0 oder 1
 unit: 8 bis 15 (ganzzahlig)

Abkürzung: aP

Beispiel:

```
10 open 1,8,2,"0:test,s,w"
20 print#1,"abc"
30 close 1
100 f$="test"
110 lw=0
120 append#1,(f$),d(lw)
130 print#1,"xyz"
140 close 1
150 rem *** Die Datei "test" enthält jetzt
160 rem *** die beiden Zeichenketten
170 rem *** "abc" und "xyz"
```

3.1.2 BACKUP

Format: BACKUP p1 [p2]
 p1: D qdrive TO zdrive
 p2: U unit / Komma kann entfallen
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Duplizieren einer Diskette (1:1 Kopie)

Voraussetzung:-

Beschreibung: Durch dieses Kommando wird eine Diskette auf eine andere Diskette innerhalb einer Diskettenstation dupliziert. Der alte Inhalt der Zieldiskette (falls vorhanden) wird dabei überschrieben und ist damit physikalisch gelöscht. 'qdrive' spezifiziert das Laufwerk, in dem sich die Quelldiskette befindet, 'zdrive' das der Zieldiskette. 'unit' gibt die Gerätenummer der Diskettenstation an, auf welchem dupliziert werden soll.

Werden für 'qdrive', 'zdrive' oder 'unit' keine Konstanten, sondern Variablen oder arithmetische Ausdrücke angegeben, so sind diese in runde Klammern einzuschliessen.

Die Reihenfolge von p1 und p2 ist frei wählbar.

Defaultwerte: unit: 8

Wertebereich: qdrive: 0 oder 1
 zdrive 0 oder 1
 unit: 8 bis 15, ganzzahlig

Abkürzung: bA

Beispiel: 10 geraet=9
 20 backup d0 to d1 u(geraet)
 30 rem *** Die Diskette in Laufwerk 0 der
 40 rem *** Diskettenstation mit der Geräte-
 50 rem *** nummer 9 wird auf die Diskette im
 60 rem *** Laufwerk 1 kopiert.
 70 end

3.1.3 BANK

Format: BANK p1
 p1: segment

Modus: Direkt / Programm

Zweck: Auswahl eines der 16 Speichersegmente für die Befehle PEEK, POKE, BLOAD, BSAVE.

Voraussetzung:-

Beschreibung: Durch dieses Kommando besteht die Möglichkeit, das Speichersegment auszuwählen, auf das sich die nachfolgenden Befehle PEEK, POKE, BLOAD und BSAVE beziehen. Diese Auswahl ist bis zum nächsten BANK-Befehl gültig. Wird bei dem Aufruf der Befehle BLOAD bzw. BSAVE ein anderes Segment angegeben, so hat dieses während der Ausführung dieser Befehle Priorität. Anschließend ist das beim letzten BANK-Befehl gewählte Speichersegment wieder gültig.

Defaultwerte: -

Wertebereich: segment: 0 bis 15, ganzzahlig

Abkürzung: baN

Beispiel: 10 seg=3
 20 for i=14*4096 to 16*4096-1
 30 bank 15
 40 a=peek(i)
 50 bank seg
 60 poke i,a
 70 next i
 80 rem *** Die Information, die in dem Segment 15
 90 rem *** von 57344 bis 65535 steht (der sog.
 100 rem *** Kernal), wird an die gleiche Stelle im
 110 rem *** Segment 3 kopiert.
 120 end

3.1.4 BLOAD

Format: BLOAD p1 [p2] [p3] [p4] [p5]
 p1: dateiname
 p2: D drive
 p3: U unit
 ON U unit / Komma kann entfallen
 p4: B segment
 ON B segment / Komma kann entfallen
 p5: P startadr

Modus: Direkt / Programm

Zweck: Kopieren von Informationen auf einer Diskette in
 den Speicher des Rechners (1:1Kopie)

Voraussetzung:-

Beschreibung: Dieses Kommando dient dazu, Information, die auf einer Diskette steht, direkt in den Speicher des Rechners zu laden. Dabei kann das Diskettengerät, das Laufwerk, das Segment, in welches geladen werden soll, sowie die Anfangsadresse gewählt werden.

Der Vorteil dieses Kommandos gegenüber DLOAD bzw. LOAD liegt darin, daß die Information auf der Diskette 1:1 in den Speicher kopiert wird.

Eine Interpretation von Pointern, wie sie z.B. in BASIC-Quelltexten benutzt werden, findet nicht statt. Sinnvolle Anwendungen dieses Kommandos sind z.B. das Laden von Assembler-Unterprogrammen oder das Laden direkt in den Bildschirm.

Das Laufwerk bzw. die Gerätenummer ist im Parameter p2 resp. p3 anzugeben, der Dateiname in p1, das Segment, in das geladen wird, in p4 und die Adresse, ab der geladen wird, in p5.

Die Reihenfolge der einzelnen Parameter ist frei wählbar. Es ist jedoch zu beachten, daß direkt hinter dem Kommando kein Komma stehen darf. Werden Variablen oder arithmetische Ausdrücke anstatt Konstanten verwendet, so sind diese in runde Klammern einzuschließen.

Defaultwerte: drive: 0
unit: 8
segment: 15 bzw. das letzte durch das Kommando
BANK eingestellte Segment
startadr: 2

Wertebereich: dateiname: String der Länge 1 bis 16
drive: 0 oder 1
unit: 8 bis 15, ganzzahlig
segment: 0 bis 15, ganzzahlig
startadr: 0 bis 65535, ganzzahlig

Abkürzung: bL

Beispiel: 10 bload "int351",b15,p1024
20 screen=13*4096
30 bload d1 on u9,b15, p(screen), "bild.01"
40 rem *** Das Ass.-Programm "int351" wird von
50 rem *** der Diskettenstation 8, Laufwerk 0
60 rem *** in das Segment 15 ab Adresse 1024
70 rem *** geladen. Die Information "bild.01"
80 rem *** wird direkt in den Bildschirmspeicher
90 rem *** des Rechners vom Gerät Nr. 9, Lauf-
100 rem *** werk 1, kopiert.

3.1.5 BSAVE

Format: BSAVE p1 [p2] [p3] [p4] [p5]
 p1: dateiname
 p2: D drive
 p3: U unit
 ON U unit / Komma kann entfallen
 p4: B segment
 ON B segment / Komma kann entfallen
 p5: P startadr TO P endadr

Modus: Direkt / Programm

Zweck: Kopieren von Informationen aus dem Speicher des Rechners auf eine Diskette (1:1Kopie)

Voraussetzung:-

Beschreibung: Dieses Kommando dient dazu, Information, die im Speicher des Rechners steht, direkt auf eine Diskette zu übertragen. Dabei kann das Diskettengerät, das Laufwerk, das Segment, aus welchem gesichert werden soll, sowie Anfangs- und Endadresse gewählt werden.

Der Vorteil dieses Kommandos gegenüber DSAVE bzw. SAVE liegt darin, daß die Information des Speichers 1:1 auf die Diskette kopiert wird.

Eine Interpretation von Pointern, wie sie z.B. in BASIC-Quelltexten benutzt werden, findet nicht statt. Sinnvolle Anwendungen dieses Kommandos sind z.B. das Sichern von Assembler-Unterprogrammen oder von Bildschirminhalten.

Das Laufwerk bzw. die Gerätenummer ist im Parameter p2 resp. p3 anzugeben, der Dateiname in p1, das Segment, in das geladen wird, in p4 und der Adressbereich in p5.

Die Reihenfolge der einzelnen Parameter ist frei wählbar. Es ist jedoch zu beachten, daß direkt hinter dem Kommando kein Komma stehen darf. Werden Variablen oder arithmetische Ausdrücke anstatt Konstanten verwendet, so sind diese in runde Klammern einzuschließen.

Defaultwerte: drive: 0
unit: 8
segment: 15 bzw. das letzte durch das Kommando
BANK eingestellte Segment
startadr: 2
endadr: 65535

Wertebereich: dateiname: String der Länge 1 bis 16
drive: 0 oder 1
unit: 8 bis 15, ganzzahlig
segment: 0 bis 15, ganzzahlig
startadr: 0 bis 65535, ganzzahlig
endadr: 0 bis 65535, ganzzahlig

Abkürzung: bS

Beispiel: 10 bsave "int351",b15,p1024 to p1100
20 sstart=13*4096
30 send=sstart+1999
40 bsave d1 on u9,b15, p(sstart) to p(send), "bild.01"
50 rem *** Das Ass.-Programm, das im Segment 15 den
60 rem *** Adreßbereich von 1024 bis 1100 belegt,
70 rem *** wird unter dem Namen "int351" auf die
80 rem *** Diskette in Laufwerk 0 des Gerätes Nr. 8
90 rem *** gesichert. Der momentane Bildschirminhalt
100 rem *** wird unter dem Namen "bild.01" wird auf
110 rem *** die Diskette in Laufwerk 1 der Disketten-
120 rem *** station mit der Geräteadresse 9 kopiert.

3.1.6 CATALOG

Format: CATALOG [p1] [p2] [p3]
p1: dateiname
p2: D drive
p3: U unit
ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Ausgabe des Disketten-Inhaltsverzeichnisses

Voraussetzung:-

Beschreibung: Durch dieses Kommando wird das Directory von Disketten ausgegeben. Dabei ist es möglich, die entsprechende Diskettenstation ('unit') sowie das Laufwerk ('drive') auszuwählen. Ferner kann bei Bedarf ein Muster ('dateiname') (siehe dort) angegeben werden, welches die auszugebende Liste auf die Dateien reduziert, welche dem Muster entsprechen.

Die Reihenfolge der Parameter ist beliebig.

Defaultwerte: dateiname: *
drive: 0 und 1, beide Laufwerke werden bearbeitet
unit: 8

Wertebereich: dateiname: String der Länge 1 bis 16
drive: 0 oder 1
unit: 8 bis 15, ganzzahlig

Abkürzung: cA

Beispiel:

```
10 catalog
20 rem *** Das Directory von beiden Laufwerken
30 rem *** (falls Doppellaufwerk) der Disketten-
40 rem *** station mit der Gerätenummer 8 wird
50 rem *** aufgelistet
60 catalogd1,"a*",u9
70 rem *** Alle Dateien auf dem Laufwerk 1 der
80 rem *** Diskettenstation Nr. 9, deren Name
90 rem *** mit dem Buchstaben 'a' beginnt, werden
100 rem *** auf dem Drucker ausgegeben.
```


3.1.7 CLOSE

Format: CLOSE p1
 p1: ziel

Modus: Direkt / Programm

Zweck: Schließen einer Datei bzw. eines Gerätes.

Voraussetzung:-

Beschreibung: Jede Datei und jedes Gerät, welche durch 'OPEN', 'DOPEN' oder 'APPEND' eröffnet wurden, sollten am Ende der Bearbeitung wieder geschlossen werden, um sie in einen ordnungsgemässen Zustand zu versetzen. 'ziel' ist dabei die logische Geräte-Nummer, unter der das abzumeldende Medium bisher angesprochen wurde.

Defaultwerte: -

Wertebereich: ziel: 1 bis 255, ganzzahlig

Abkürzung: c10

Beispiel: open 4,4
 print#4,"abc"
 close 4
 10 rem *** 'abc' wurde auf dem Drucker ausgegeben
 20 open 1,8,2,"0:testfile,s,w"
 30 print#1,"abc"
 40 close 1
 30 rem *** Es wird eine Datei mit dem Namen 'testfile'
 40 rem *** auf der Diskette im Laufwerk 0 der
 50 rem *** Diskettenstation 8 zum Schreiben erstellt
 60 rem *** und 'abc' eingetragen. Würde Zeile 40
 70 rem *** fehlen und die Datei auch später nicht
 80 rem *** geschlossen werden, so würde die Datei
 90 rem *** im Directory als falsch geschlossen
 100 rem *** aufgeführt.
 110 end

3.1.8 CMD

Format: CMD p1
 p1: ziel

Modus: Direkt / Programm

Zweck: Verändern des Standard-Ausgabegerätes.

Voraussetzung: 'ziel' wurde vorher (zum Schreiben) eröffnet.

Beschreibung: Alle Ausgaben (außer Fehlermeldungen), die normalerweise auf dem Bildschirm ausgegeben werden, werden auf das durch 'ziel' definierte Gerät (z.B. Drucker) umgeleitet. 'ziel' ist dabei die logische Gerätenummer, unter der das Ausgabegerät angesprochen wird (s. 'OPEN').

Soll diese Umstellung erneut geändert (oder zurückgesetzt) werden, so ist zuerst die Ausgabe (durch 'CMD') erneut umzuleiten, ein 'PRINT#'-Befehl auf das bisherige Ausgabegerät abzusetzen und letzteres dann schließen ('CLOSE'). Wird dies nicht beachtet, kann die Steuerung des Cursors eine falsche Information erhalten.

Ein weiterer Unterschied besteht bei diesem Kommando zwischen Direkt- und Programmmodus: Im Direktmodus werden nur die Ausgaben, die durch ein Kommando erzeugt werden (z.B. 'LIST', 'DIRECTORY'), umgeleitet. Wird anschliessend ein Programm gestartet, so wird die Ausgabe des Programms (durch 'PRINT'-Befehl) auf den Bildschirm. Wird 'CMD' im Programm durchlaufen, so werden auch die durch 'PRINT' erzeugten Ausgaben umgeleitet.

Defaultwerte: -

Wertebereich: ziel: 1 bis 255, ganzzahlig

Abkürzung: cM

Beispiel:

```
open 10,4
cmd 10
directory d0
dload "test"
list
10 print "abc"
20 rem *** Das Directory wird auf dem Drucker,
30 rem *** "abc" auf dem Bildschirm ausgegeben.
40 end
print#10
close 10
dload "test2"
list
10 open 10,4
20 cmd 10
30 directory d0
40 print "abc"
50 open 3,3
60 print#10
70 close 10
80 rem *** Das Directory und "abc" wird auf
90 rem *** dem Drucker ausgegeben.
100 end
```

3.1.9 COLLECT

Format: COLLECT [p1] [p2]
 p1: D drive
 p2: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Logisches Bereinigen einer Diskette

Voraussetzung:-

Beschreibung: Die Diskette, die sich in dem Laufwerk 'drive' der Diskettenstation 'unit' befindet, wird überprüft. Dabei werden Blöcke auf der Diskette, die zwar als belegt gekennzeichnet, jedoch von keiner existierenden Datei auf der Diskette benutzt werden, freigegeben. Außerdem werden Dateien, die beim Schreiben nicht ordnungsgemäß geschlossen wurden (und daher im Directory mit einem Stern (*) gekennzeichnet sind), gelöscht.

Wurde die Diskette vorher mit Direktzugriffen beschrieben, so kann diese Information beim anschließenden Schreiben auf diese Diskette zerstört werden.

Werden für 'drive' bzw. 'unit' keine Konstanten, sondern Variable angegeben, so sind sie in Klammern einzuschließen.

Die Reihenfolge von p1 und p2 ist frei wählbar.

Defaultwerte: drive: das zuletzt angesprochene Laufwerk;
 unit: beim Einschalten Laufwerk 0
 8

Wertebereich: drive: 0 oder 1
 unit: 8 bis 15, ganzzahlig

Abkürzung: col

Beispiel:

```
10 unit=9
20 catalog d1, u(unit)
30 collect u(unit)
40 rem *** Die Diskette im Laufwerk 1 der
50 rem *** Diskettenstation mit der Geräte-
60 rem *** nummer 9 wird bereinigt.
70 end
```

3.1.10 CONCAT

Format: CONCAT [p1] p2 TO [p3] p4 [p5]
 p1: D quellldr
 p2: qdatei
 p3: D zieldr
 p4: zdatei
 p5: U unit / Komma kann entfallen
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Anhängen einer Datei hinter die Daten einer
 anderen Datei

Voraussetzung: Beide Dateien müssen vom Typ 'seq' oder 'usr' sein.

Beschreibung: Mit diesem Kommando wird die Information, die in der Datei 'qdatei' auf dem Laufwerk 'quellldr' enthalten ist, hinter die Daten, welche sich in der Datei 'zdatei' auf dem Laufwerk 'zieldr' befinden, angefügt. 'unit' bezeichnet dabei die Diskettenstation, auf der sich die Quell- und die Zieldiskette befinden.

p1 kann mit p2 vertauscht werden, ebenso p3 mit p4. p5 kann irgendwo dazwischen stehen. Direkt nach 'TO' darf kein Komma stehen. In p1 ist der Name der Quelldatei anzugeben, in p2 das dazugehörige Laufwerk; in p3 bzw. p4 die entsprechenden Angaben für die Zieldiskette. p5 bezeichnet die Geräteadresse der Diskettenstation. Werden als Dateinamen bzw. als Laufwerk keine Konstanten, sondern Variablen oder Ausdrücke benutzt, so sind diese in Klammern einzuschließen.

Defaultwert: quellldr: 0
 zieldr: 0
 unit: 8

Wertebereich: qdatei: String der Länge 1 bis 16
 quelldr: 0 oder 1
 zdatei: String der Länge 1 bis 16
 zielldr: 0 oder 1
 unit: 8 bis 15, ganzzahlig

Abkürzung: conC

Beispiel: 10 concat "special.data" to "global.data"
 20 rem *** Die Daten der Datei 'special.data'
 30 rem *** werden hinter die bereits vorhandenen
 40 rem *** Daten in der Datei 'global.data' einge-
 50 rem *** fügt. Die Datei 'special.data' steht
 60 rem *** anschließend unverändert zur Ver-
 70 rem *** fügung.
 80 end

3.1.11 CONT

Format: CLOSE

Modus: Direkt

Zweck: Fortsetzen eines unterbrochenen Programmes.

Voraussetzung: Es wurden seit der Unterbrechung keine Programm-
änderungen vorgenommen; hinter der Unterbrechungs-
stelle existieren Anweisungen.

Beschreibung: Ein Programm, das eine 'STOP'-Anweisung oder eine
'END'-Anweisung durchlief, kann durch dieses
Kommando fortgesetzt werden. Genauso kann ein
Programm, das durch die 'STOP'-Taste unterbrochen
wurde, fortgesetzt werden. Der Wert von Variablen
wird hierdurch nicht verändert.

Defaultwerte: -

Wertebereich: -

Abkürzung: c0

Beispiel:

```
10 i=10
20 print i
30 end
40 i=i+1
50 print i
60 end
run
10 (Ausgabe)
ready. (Ausgabe)
cont (Ausgabe)
11 (Ausgabe)
ready. (Ausgabe)
```


3.1.12 COPY

Format: COPY [p1] [p2] TO [p3] p4 [p5]

p1: D quelldr

p2: qdatei

p3: D zildr

p4: zdatei

p5: U unit / Komma kann entfallen

ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Kopieren von Dateien innerhalb einer Diskettenstation

Voraussetzung:-

Beschreibung: Das COPY-Kommando erlaubt es auf vielfältige Art, eine oder mehrere Dateien in kurzer Zeit innerhalb einer Diskettenstation zu kopieren. 'unit' gibt dabei die Gerätenummer der Diskettenstation an. Wenn der Name der Quelldatei ('qdatei') von dem der Zieldatei ('zdatei') verschieden ist, so kann das Laufwerk, auf dem sich die Quelldatei befindet ('quelldr'), mit dem Laufwerk der Zieldatei ('zildr') identisch sein.

Als (Quell-)Dateiname kann auch ein Musterstring (s. dort) angegeben werden. Als Name der Zieldatei ist dabei "*" anzugeben. Es werden alle Dateien, die dem Musterstring entsprechen, kopiert. Da hierbei die Namen von Quell- und Zieldateien identisch sind, müssen die Quell- und Ziellaufwerke verschieden sein.

Auf der Zieldiskette darf keine Datei existieren, die den gleichen Namen wie eine zu kopierende Quelldatei hat.

p1 kann mit p2 vertauscht werden, ebenso p3 mit p4. p5 kann irgendwo dazwischen stehen. Direkt nach 'TO' darf kein Komma stehen. In p1 ist der Name (oder Musterstring) anzugeben, in p2 das dazugehörige Laufwerk; in p3 bzw. p4 die entsprechenden Angaben für die Zieldiskette. p5 bezeichnet die Geräteadresse der Diskettenstation. Werden als Dateinamen bzw. als

Laufwerk keine Konstanten, sondern Variablen oder Ausdrücke benutzt, so sind diese in Klammern einzuschließen.

p1 und p2 dürfen nicht gleichzeitig fehlen. Wird p1 nicht angegeben, so muß auch p3 fehlen.

Defaultwerte: quellldr: 0

Wertebereich: qdatei: String der Länge 1 bis 16
 quellldr: 0 oder 1
 zdatei: String der Länge 1 bis 16
 zielldr: 0 oder 1
 unit: 8 bis 15, ganzzahlig

Abkürzung: coP

Beispiel:

```

10 copy d0 to d1
20 rem *** Alle Dateien auf der Diskette im
30 rem *** Laufwerk 0 der Diskettenstation
40 rem *** mit der Gerätenummer 8 werden kopiert.
50 copy "prog.1" to "prog.1.save" on u9
60 rem *** Von der Datei 'prog.1' in Laufwerk 1
70 rem *** der Diskettenstation Nr. 9 wird eine
80 rem *** Sicherheitskopie auf der gleichen
90 rem *** Diskette angelegt.
100 copy "*",d0 to "*",d1
110 rem *** wie Zeile 10
120 unit=10
130 copy "graf*",d0 to "*",d1 u(unit)
140 rem *** Alle Dateien, deren Namen mit der
150 rem *** Zeichenfolge 'graf' beginnen, werden
160 rem *** von Laufwerk 0 nach Laufwerk 1 in
170 rem *** Diskettenstation Nr. 10 kopiert.
```

3.1.13 DCLEAR

Format: DCLEAR [p1] [p2] [p3]
 p1: # dateinummer
 p2: D drive
 p3: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Abschließen von nicht ordnungsgemäß geschlossenen Dateien

Voraussetzung:-

Beschreibung: Dieses Kommando versucht, Diskettendateien, welche nicht ordnungsgemäß abgeschlossen wurden, korrekt zu beenden. Ein erfolgreiches Ausführen ist jedoch nicht immer gewährleistet, da dabei viele Parameter zusammenspielen. Außerdem können nicht gewollte Nebenwirkungen auftreten. Dieses Kommando sollte daher nur als Notlösung angewandt werden. Mit 'dateinummer' kann eine einzelne Datei bestimmt werden, welche bearbeitet werden soll; dabei wird in p2 das Laufwerk ('drive'), in p3 die Diskettenstation ('unit') angegeben, auf welcher sich die nicht korrekt beendeten Dateien befinden.

Werden für 'drive' oder 'unit' keine Konstanten, sondern Variable oder Ausdrücke verwendet, so sind sie in runde Klammern einzuschließen. Die Reihenfolge der Parameter ist beliebig.

Defaultwerte: drive: 0
 unit: 8

Wertebereich: drive: 0 oder 1
 "n"t 8 1 1 , a z a l g

b ü z n : c E

Beispiel:

```
10 open 1,8,2,"0:schrott,s,w"
20 print#1,"xyz"
30 x=1/0
40 rem *** Das Programm ist hier (fehlerhaft)
50 rem *** beendet, die Datei "schrott" jedoch
60 rem *** noch nicht beendet.
dclear
70 rem *** 'DCLEAR' behebt nun diesen ungewollten
80 rem *** Zustand.
```

3.1.14 DCLOSE

Format: DCLOSE# [p1] [p2]
 p1: dateinummer
 p2: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Schließen einer Datei

Voraussetzung:-

Beschreibung: Jede Datei, welche durch 'OPEN', 'DOPEN' oder 'APPEND' eröffnet wurde, sollte am Ende der Bearbeitung wieder geschlossen werden, um sie in einen ordnungsgemässen Zustand zu versetzen. 'dateinummer' ist dabei die logische Geräte- nummer, unter der die abzumeldende Datei bisher angesprochen wurde. Fehlt diese Angabe, so werden alle Dateien auf der entsprechenden Diskettensta- tion abgemeldet.

Die Reihenfolge der Parameter ist beliebig.

Werden statt Konstanten Variablen verwendet, so sind sie in runde Klammern einzuschließen.

Defaultwerte: unit: 8

Wertebereich: dateinummer: 1 bis 255, ganzzahlig
 unit: 8 bis 15, ganzzahlig

Abkürzung: dC#

Beispiel:

```
10 open 1,9,2,"0:testfile,s,r"
20 open 2,9,3,"1:testfile,s,w"
30 for i=1 to 20
40 input#1,a$
50 print#2,a$
60 next i
70 dclose u9
80 rem *** Es wird eine Datei mit dem Namen 'testfile'
90 rem *** auf der Diskette im Laufwerk 0 der
100 rem *** Diskettenstation 9 zum Lesen angemeldet
110 rem *** und eine gleichnamige Datei auf dem
120 rem *** Laufwerk 1 kreiert. Nachdem 20 Werte
130 rem *** von einer Datei zur anderen übertragen
140 rem *** wurden, werden beide Dateien geschlossen.
150 end
```

3.1.15 DELETE

Format: DELETE [p1]
 p1: anfangszeile
 anfangszeile -
 anfangszeile - endzeile
 - endzeile

Modus: Direkt / Programm

Zweck: Loeschen von Programmzeilen

Voraussetzung:-

Beschreibung: Durch dieses Kommando werden die angegebenen Programmzeilen gelöscht. Wird nur die Anfangszeile angegeben, so wird nur diese gelöscht. Wird die Möglichkeit 'anfangszeile -' gewählt, so wird das Programm ab der Anfangszeile bis zum Ende gelöscht. Werden die Anfangs- und die Endzeile angegeben, so werden alle Zeilen, deren Zeilennummern in diesem Bereich liegen (einschließlich Anfangs- und Endzeile, falls vorhanden) gelöscht. Wird nur die Endzeile ('- endzeile') angegeben, so wird der Bereich von der ersten gültigen Zeile bis zur Endzeile (einschließlich) gelöscht. In folgenden Fällen wird das ganze Programm gelöscht:

Anfangszeile	Endzeile
nicht angegeben	nicht angegeben
0	nicht angegeben
nicht angegeben	0
0	0

Dieses Kommando kann zwar im Programm aufgerufen werden; dies ist jedoch nicht sinnvoll, da das Programm anschließend nicht fortgesetzt werden kann.

Defaultwerte: anfangszeile: 0
 endzeile: 63999

Wertebereich: anfangszeile: 0 bis 63999, ganzzahlig
endzeile: 0 bis 63999, ganzzahlig

Abkürzung: del

Beispiel: 10 print 10
20 print 20
30 print 30
delete 20
40 rem *** Es sind nur noch die Zeilen 10 und 30
50 rem *** vorhanden.
delete -0
60 rem *** Es ist nichts mehr vorhanden.

3.1.16 DIRECTORY

Format: DIRECTORY [p1] [p2] [p3]
 p1: dateiname
 p2: D drive
 p3: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Ausgabe des Disketten-Inhaltsverzeichnisses

Voraussetzung:-

Beschreibung: Durch dieses Kommando wird das Directory von Disketten ausgegeben. Dabei ist es möglich, die entsprechende Diskettenstation ('unit') sowie das Laufwerk ('drive') auszuwählen. Ferner kann bei Bedarf ein Muster ('dateiname') (siehe dort) angegeben werden, welches die auszugebende Liste auf die Dateien reduziert, welche dem Muster entsprechen.

Die Reihenfolge der Parameter ist beliebig.

Defaultwerte: dateiname: *
 drive: 0 und 1, beide Laufwerke werden bearbeitet
 unit: 8

Wertebereich: dateiname: String der Länge 1 bis 16
 drive: 0 oder 1
 unit: 8 bis 15, ganzzahlig

Abkürzung: diR

Beispiel:

```
10 directory
20 rem *** Das Directory von beiden Laufwerken
30 rem *** (falls Doppellaufwerk) der Disketten-
40 rem *** station mit der Gerätenummer 8 wird
50 rem *** aufgelistet
60 directory d1,"a*",u9
70 rem *** Alle Dateien auf dem Laufwerk 1 der
80 rem *** Diskettenstation Nr. 9, deren Name
90 rem *** mit dem Buchstaben 'a' beginnt, werden
100 rem *** auf dem Drucker ausgegeben.
```

3.1.17 DLOAD

Format: DLOAD p1 [p2] [p3]
 p1: dateiname
 p2: D drive
 p3: U unit
 ON U unit

Modus: Direkt / Programm

Zweck: Laden eines Programmes von Diskette

Voraussetzung:-

Beschreibung: Dieses Kommando bewirkt das Laden eines Programmes, das sich auf einer Diskette befindet. Das Programm wird dabei automatisch ab den Anfangsadresse des Basic-Text-Bereiches abgelegt, sodaß auch Basic-Programme von älteren Commodore-Rechnern problemlos eingelesen werden können. (Soll die zu ladende Information an einer anderen Stelle abgelegt werden, so ist das Kommando 'BLOAD' zu verwenden). Der Name der Datei ist im Parameter 'dateiname' anzugeben. Das Laufwerk, in welchem sich die Diskette mit dem zu ladenden Programm befindet, wird in 'drive' spezifiziert, die Geräteadresse in 'unit'.

Wird dieses Kommando in einem Programm gegeben, so wird als nächster Befehl der erste Befehl des nachgeladenen Programms ausgeführt. Alle Variablen, die im vorherigen Programm deklariert wurden, stehen weiter zur Verfügung.

Werden als Parameter keine Konstanten, sondern Variable oder Ausdrücke angegeben, so sind diese in runde Klammern einzuschließen. Die Reihenfolge der Parameter ist beliebig.

Defaultwerte: drive: 0
 unit: 8

Wertebereich: dateiname: String der Länge 1 bis 16
 drive: 0 oder 1
 unit: 8 bis 15, ganzzahlig

Abkürzung: dL

Beispiel: 10 input "Modulname";na\$
 20 dload (na\$),d1,u8
 30 rem *** Das Programm auf der Diskette in
 40 rem *** Laufwerk 0 der Diskettenstation 8
 50 rem *** mit dem eingelesenen Namen wird
 60 rem *** geladen und an seiner ersten An-
 70 rem *** weisung begonnen.

3.1.18 DOPEN

Format: DOPEN p1 p2 [p3] [p4] [p5] [p6]
p1: # dateinummer
p2: dateiname
p3: L satzlänge
p4: D drive
p5: U unit
ON U unit / Komma kann entfallen
p6: modus

Modus: Direkt / Programm

Zweck: Eröffnen von SEQ-/ PRG-/ USR- oder REL-Dateien
zum Lesen oder von SEQ- oder REL-Dateien zum
Schreiben

Voraussetzung:-

Beschreibung: Durch dieses Kommando wird eine Datei zur Bearbeitung eröffnet. Die einzelnen Parameter haben dabei folgende Bedeutung:

'dateinummer': Dies ist die Nummer, mit der die Datei im weiteren Verlauf angesprochen wird.

'dateiname': ist, wie der Name schon verdeutlicht, der Name der Datei, welche bearbeitet werden soll. Handelt es sich hierbei um eine bereits bestehende sequentielle Datei, welche neu beschrieben werden soll, so ist dem Namen das Zeichen '\$' (Klammeraffe) voranzustellen.

'satzlänge': wird nur bei REL-Dateien akzeptiert. Hiermit wird die Länge der Sätze der Datei festgelegt.

'drive': spezifiziert das Laufwerk, auf dem sich die zu bearbeitende Datei befindet bzw. auf dem die neue Datei zu kreieren ist.

'unit': bezeichnet die Gerätenummer der zugehörigen Diskettenstation.

'modus': definiert die Zugriffsart; 'W' bedeutet Beschreiben einer sequentiellen Datei. Fehlt dieser Parameter, so wird lesender Zugriff angenommen,

sofern es sich nicht um eine REL-Datei handelt, bei welcher immer lesender und schreibender Zugriff erlaubt ist (es sei denn, der Schreibschutz ist auf der Diskette aufgeklebt).

Sollen PRG- oder USR-Dateien geschrieben werden, so ist das Kommando 'OPEN' (Kap. 3.1.25) zu verwenden.

Werden statt Konstanten Variable oder Ausdrücke eingesetzt, so sind diese in Klammern einzuschließen. Die Reihenfolge der Parameter ist beliebig.

Defaultwerte: drive: 0
unit: 8

Wertebereich: dateinummer: 1 bis 255, ganzzahlig
dateiname: string der Länge 1 bis 16
satzlänge: 2 bis 254, ganzzahlig
drive: 0 oder 1
unit: 8 bis 15, ganzzahlig
modus: W

Abkürzung: d0#

Beispiel:

```

10 input "Dateiname";dn$
20 input "Laufwerk ";lw
30 input "Geraet ";ge
40 dopen w, d(lw) on u(ge), (dn$), # 10
50 get a$: if a$="" then a$=chr$(0)
60 if asc(a$)=27 then dclose #10
70 print a$;
80 print #10,a$;: go to 50
90 rem *** Es wird eine Datei kreiert, welche
100 rem *** den eingelesenen Namen erhält und
110 rem *** auf der Diskette im Laufwerk 'lw'
120 rem *** der Diskettenstation 'ge' angelegt
130 rem *** wird. Dann werden alle von der Tastatur
140 rem *** eingegebenen Zeichen in ihr abgelegt,
150 rem *** bis die 'ESC'-Taste gedrückt wird.
```

3.1.19 DSAVE

Format: DSAVE p1 [p2] [p3]
p1: dateiname
p2: D drive
p3: U unit
ON U unit

Modus: Direkt / Programm

Zweck: Sichern eines Programmes auf Diskette

Voraussetzung:-

Beschreibung: Dieses Kommando bewirkt das Sichern eines Programmes auf eine Diskette. Das Programm wird dabei ab der Anfangsadresse des Basic-Text-Bereiches erwartet. Soll Information von einer anderen Stelle aus abgelegt werden, so ist das Kommando 'BSAVE' zu verwenden. Der Name, den das Programm auf der Diskette erhalten soll, ist im Parameter 'dateiname' anzugeben. Das Laufwerk, in welchem sich die Diskette für das zu sichernde Programm befindet, wird in 'drive' spezifiziert, die die Geräteadresse in 'unit'. Wenn sich auf der so angesprochenen Diskette bereits eine Datei mit dem angegebenen Namen befindet, so wird dieses Kommando nicht ausgeführt; die Fehlerlampe an der Diskettenstation leuchtet auf. Soll eine bestehende Datei überschrieben werden, so ist deshalb dem Dateinamen das Zeichen '\$' (Klammeraffe) voranzustellen.

Werden als Parameter keine Konstanten, sondern Variable oder Ausdrücke angegeben, so sind diese in runde Klammern einzuschließen. Die Reihenfolge der Parameter ist beliebig.

Defaultwerte: drive: 0
unit: 8

Wertebereich: dateiname: String der Länge 1 bis 16 (17 beim Überschreiben)
drive: 0 oder 1
unit: 8 bis 15, ganzzahlig

Abkürzung: dS

Beispiel:

```
10 print "Hipp hipp hurra! "  
20 dsave d1, "$prg.hipp"  
30 rem *** Dieses Programm wird auf die Diskette  
40 rem *** in Laufwerk 0 der Diskettenstation 8  
50 rem *** unter dem Namen "prg.hipp"  
60 rem *** gesichert. Es existierte darauf  
70 rem *** bereits eine Datei mit dem gleichen  
80 rem *** Namen.
```


3.1.20 HEADER

Format: HEADER p1,[p2],p3 [p4]
 p1: diskettenname
 p2: I id
 p3: D drive
 p4: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Formatieren bzw. Umbenennen einer Diskette

Voraussetzung: Beim Umbenennen: Diskette bereits formatiert

Beschreibung: Dieses Kommando ermöglicht es, neue Disketten zu formatieren, alte Disketten neu zu formatieren bzw. alte Disketten umzubenennen. Der Name, den die Diskette erhalten soll, ist in p1 anzugeben. Bearbeitet wird die Diskette im Laufwerk 'drive' der Diskettenstation mit der Gerätenummer 'unit'.

Bei dem Parameter p2 ist folgendes zu beachten: Wird dieser Parameter angegeben, so wird die ganze Diskette neu formatiert. Fehlt dieser Parameter, so wird das Inhaltsverzeichnis der Diskette neu angelegt und der Diskette ein neuer Name zugeordnet (Umbenennen). Im letzteren Fall sind somit die bisherigen Dateien logisch gelöscht, wohingegen sie im ersten Fall physikalisch gelöscht werden.

Der Parameter p3 bezeichnet dabei das Identifikations-Kennzeichen ('id') der Diskette. Dieses kann nur verändert werden, indem die ganze Diskette neu formatiert wird, da es in jedem Kopf der Informationsblöcke für Kontrollzwecke vermerkt wird. Dieses 'id' besteht aus genau zwei Zeichen. Diese sind ohne Stringbegrenzer ('Gänsefüßchen oben') anzugeben. Es werden die nächsten beiden Zeichen hinter dem Buchstaben 'I' des Parameters p3 ausgewertet. Daher kann dieser Parameter in einem Basic-Programm nicht durch eine Variable ersetzt werden. Zu bemerken ist fernerhin, daß dies der einzige Parameter in einem Basic-Kommando ist, bei dem Leerzeichen oder Kommata ausgewertet werden, ohne daß sie durch Stringbegrenzer eingeschlossen zu werden.

Wird das Kommando im Direkt-Modus gegeben, so erscheint vor der Ausführung des Kommandos die Frage: 'are you shure ?'. Wird diese Frage mit 'y' oder 'yes' beantwortet, so wird das Kommando ausgeführt. In allen anderen Fällen wird das Kommando nicht ausgeführt. Wird dieses Kommando durch ein Programm aufgerufen, so unterbleibt diese Kontrollfrage, es wird direkt ausgeführt.

Die Reihenfolge der Parameter ist beliebig. Werden bei den Parametern p1,p2 und p3 keine Konstanten, sondern Variable oder Ausdrücke angegeben, so sind diese in runde Klammern einzuschließen.

Defaultwerte: unit: 8

Wertebereich: diskettenname: String der Länge 1 bis 16
 drive: 0 oder 1
 id: genau zwei ASCII-Zeichen
 unit: 8 bis 15, ganzzahlig

Abkürzung: hE

Beispiel:

```

10 input "Diskettenname"; na$
20 input "Laufwerk "; lw
30 input "Gerät "; ge
40 header (na$),d(lw)onu(ge),ich
50 rem *** Die Diskette im Laufwerk 'lw' der
60 rem *** Diskettenstation 'ge' wird neu
70 rem *** formatiert. Sie erhält den Namen,
80 rem *** der in die Variable na$ eingelesen wurde
90 rem *** und das Identifikationskennzeichen 'ch'.
100 rem *** Die Frage 'are you shure ?' unterbleibt.
```

3.1.21 KEY

Format: KEY [p1 p2]
 p1: fnummer
 p2: string

Modus: Direkt / Programm

Zweck: Belegen oder Auflisten der Funktionstasten

Voraussetzung:-

Beschreibung: Werden keine Parameter angegeben, so wird die derzeitige Belegung der 20 Funktionstasten aufgelistet. Werden die beiden Parameter angegeben, so wird der Funktionstaste, welche durch 'fnummer' angesprochen wird, der String 'string' zugewiesen. Beachten Sie dabei, daß die Gesamtlänge der Strings, die unter den Funktionstasten abgelegt werden, maximal 511 sein darf!

Defaultwerte: -

Wertebereich: fnummer: 1 bis 20, ganzzahlig
 string: String

Abkürzung: kE

Beispiel: key 12,"list"+chr\$(13)+"run"+chr\$(13)
 key (bewirkt Auflisten aller belegten Funktionstasten)
 shift F2 (durch Drücken der Tasten 'SHIFT' und 'F2' (= 'F12') wird das im Speicher sich befindende Programm aufgelistet und danach ausgeführt)

3.1.22 LIST

Format: LIST [p1]
 p1: anfangszeile
 anfangszeile -
 anfangszeile - endzeile
 - endzeile

Modus: Direkt / Programm

Zweck: Listen von Programm(bereichen)

Voraussetzung:-

Beschreibung: Durch dieses Kommando werden die angegebenen Programmzeilen aufgelistet. Wird nur die Anfangszeile angegeben, so wird nur diese aufgelistet. Wird die Möglichkeit 'anfangszeile -' gewählt, so wird das Programm ab der Anfangszeile bis zum Ende aufgelistet. Werden die Anfangs- und die Endzeile angegeben, so werden alle Zeilen, deren Zeilennummern in diesem Bereich liegen (einschließlich Anfangs- und Endzeile, falls vorhanden) aufgelistet. Wird nur die Endzeile ('- endzeile') angegeben, so wird der Bereich von der ersten gültigen Zeile bis zur Endzeile (einschließlich) aufgelistet. In folgenden Fällen wird das ganze Programm aufgelistet:

Anfangszeile	Endzeile
nicht angegeben	nicht angegeben
0	nicht angegeben
nicht angegeben	0
0	0

Dieses Kommando kann zwar im Programm aufgerufen werden; dies ist jedoch nicht sinnvoll, da sich das Programm anschließend abbricht und durch 'CONT' nicht mehr fortgesetzt werden kann (genauer: es wird an der Stelle fortgesetzt, an dem das 'LIST'-Kommando steht).

Defaultwerte: anfangszeile: 0
 endzeile: 63999

Wertebereich: anfangszeile: 0 bis 63999, ganzzahlig
endzeile: 0 bis 63999, ganzzahlig

Abkürzung: 1I

Beispiel: 10 ? 10 (Eingabe)
20 ? 20
30 ? 30
list 20

20 print 20 (Ausgabe)

ready.
list (Eingabe)

10 print 10 (Ausgabe)
20 print 20
30 print 30

ready.

3.1.23 LOAD

Format: LOAD p1 [p2]
 p1: drive: name
 p2: unit

Modus: Direkt / Programm

Zweck: Laden eines Programmes von Diskette

Voraussetzung:-

Beschreibung: Dieses Kommando bewirkt das Laden eines Programmes. Der Name des Programmes, das geladen werden soll, ist im Parameter 'name' anzugeben. Der ganze Parameter 'p1' ist ein String, der sich aus dem Laufwerk (bei Diskettenstationen) und dem Namen zusammensetzt. Beide Angaben sind durch einen Doppelpunkt voneinander zu trennen. Die Geräteadresse der Station, von welcher das Programm geladen wird, ist in 'unit' anzugeben.

Wird dieses Kommando in einem Programm gegeben, so wird als nächster Befehl der erste Befehl des nachgeladenen Programms ausgeführt. Alle Variablen, die im vorherigen Programm deklariert wurden, stehen weiter zur Verfügung.

Defaultwerte: drive: 0
 unit: 1

Wertebereich: drive: 0 oder 1
 name: (Teil-)String der Länge 1 bis 16
 unit: 1 bis 15, ganzzahlig

Abkürzung: 10

Beispiel: load "1:prog.01",11 (Das Programm 'prog.01'
wird vom Laufwerk 1
der (Disketten)-station
Nr. 11 geladen)

```
10 lw=1
20 un=11
30 dn$="prog.01"
40 load str$(lw)+":"+dn$,un
50 rem *** Gleiche Wirkung wie oben, hier wird
60 rem *** jedoch das nachgeladene Programm
70 rem *** mit seiner ersten Anweisung begonnen
```

3.1.24 NEW

Format: NEW

Modus: Direkt / Programm

Zweck: Programm im Speicher logisch löschen

Voraussetzung:-

Beschreibung: Durch dieses Kommando wird das Programm, das sich derzeit im Speicher befindet, (logisch) gelöscht. Es kann danach weder gelistet noch ausgeführt werden. Die Konsequenz, die der Aufruf dieses Kommandos in einem Programm zur Folge hat, ist klar: das Programm ist gelöscht, es werden keine Anweisungen dieses Programmes mehr ausgeführt.

Defaultwerte: -

Wertebereich: -

Abkürzung: N

```
Beispiel:      10 print "Das Programm ist weg! "
                20 new
                run                                (Eingabe)
                Das Programm ist weg!              (Ausgabe)
                                                        (Ausgabe)
                ready.                             (Ausgabe)
                list                                (Eingabe)
                                                        (Ausgabe)
                ready.                             (Ausgabe)
```


3.1.25 OPEN

Format: OPEN p1 p2 [p3] [p4]
 p1: mediennummer
 p2: primadresse
 p3: sekadresse
 p4: fileparam

Modus: Direkt / Programm

Zweck: Eröffnen einer Datei oder eines Gerätes zum
 Lesen oder Schreiben

Voraussetzung:

Beschreibung: Vorbemerkung: In dieser Beschreibung ist an den
 Stellen, wo von einem 'Medium' die Rede ist,
 eine Datei oder ein Gerät gemeint.

Jedes Medium, auf das (lesend oder schreibend)
zugegriffen werden soll, muß vor dem ersten
Zugriff eröffnet werden. Die einzigen Ausnahmen
von dieser Regel sind die Tastatur und der Bild-
schirm. Durch das 'OPEN'-Kommando wird dem Rechner
die Zuordnung von einem Medium zu einer Nummer,
unter welcher das Medium fortan angesprochen wird,
bekanntgegeben. Diese soeben erwähnte Nummer ist
die 'dateinnummer' (oder die 'logical unit'). Über
sie wird das entsprechende Medium angesprochen.
(s. 'GET#', 'INPUT#', 'PRINT#'; 'CLOSE', 'DCLOSE#').
Die Gesamtzahl offener Medien darf 10 nicht über-
schreiten.

Die Primäradresse ('primadresse') spezifiziert das
Gerät, welches angesprochen wird bzw. auf dem sich
die Datei befindet, welche angesprochen wird.

Die Systemfamilien CBM 600/700 sowie die COMMODORE-Peripherie haben standardmäßig folgende Primäradressen:

0	Tastatur	(nicht veränderbar)
1	Kassette	(nicht veränderbar)
2	RS 232 C	(nicht veränderbar)
3	Bildschirm	(nicht veränderbar)
4	Drucker	
5	Akustic-Koppler oder Plotter	
6	frei	
7	frei	
8	Diskettenstation oder Harddisk	
9 bis 15	frei für Diskettenstationen oder Harddisks	

Die Bedeutung der Sekundäradresse ('sekadresse') hängt von dem angesprochenen Gerät ab. So kann z. B. bei einem Drucker hierüber zwischen Klein- und Großschreibung umgeschaltet werden (drucker-abhängig), oder bei einer Diskettenstation bzw. Harddisk der Datenkanal ausgewählt werden (wichtig bei Direktzugriffen). Die Bedeutung der Sekundäradressen ist jeweils in den zu den Peripheriegeräten gehörenden Handbüchern beschrieben. Zur RS 232 C-Schnittstelle siehe auch Kap. 5.1.1

Auch der Fileparameter ('fileparam') ist geräte-abhängig. Da er jedoch überwiegend bei Diskettenstationen bzw. Harddisks eingesetzt wird, kann hier auf eine Beschreibung des Fileparameters nicht verzichtet werden. Er ist ein String, der aus bis zu vier Teilangaben besteht: Laufwerk, Dateiname, Dateityp, Zugriffsart oder Satzlänge. Diese Angaben sind in dieser Reihenfolge vorzunehmen. Das Laufwerk ist von Dateiname durch Doppelpunkt zu trennen, die anderen Angaben sind durch Kommata voneinander zu trennen.

Als Laufwerk ist '0' (Null) oder '1' (Eins) anzugeben. ('1' nur sinnvoll bei Doppellaufwerken). Soll eine bestehende Datei überschrieben werden, so ist unmittelbar vor der Laufwerksnummer ein '\$' (Klammeraffe) anzugeben. Wird kein Laufwerk angegeben, so wird im Falle des Lesens oder des Überschreibens einer existierenden Datei die Datei auf dem zuletzt angesprochenen Laufwerk gesucht. Wird sie dort nicht gefunden, wird das andere Laufwerk durchsucht. (Bei Diskettenstationen mit nur einem Laufwerk entfällt dies natürlich).

Der Dateiname ist der Name der Datei, welche zu bearbeiten ist. Er ist unbedingt anzugeben.

Als Dateityp sind folgende Angaben möglich: 'S' (sequentiell), 'P' (Programm), 'U' (User) sowie 'L' (relativ). Bei den Typen S,P und U ist außerdem die Zugriffsart zu spezifizieren; fehlt diese, so wird lesender Zugriff angenommen. Ein Sonderfall tritt bei beiden Sekundäradressen Null und Eins auf: Hierbei wird als Dateityp 'P' angenommen; Sekundäradresse Null bedeutet lesen, Sekundäradresse Eins hingegen schreiben. Beim Dateityp 'R' ist anstatt der Zugriffsart die Satzlänge anzugeben. Relativdateien werden immer zum Lesen und Schreiben eröffnet.

Zugriffsart: Mögliche Angaben sind 'W' (Schreiben) und 'R' (Lesen) Für fehlende Angaben siehe oben.

Satzlänge: Sie muß als ein Byte angegeben werden. Am einfachsten verwendet man hierzu die 'CHR\$'-Funktion. Alle Sätze einer Relativdatei haben die gleiche Länge.

Defaultwerte: sa: 0 / falls 'fileparam' nicht
/ angegeben
fileparam: ""

Wertebereich: mediennummer: 1 bis 255, ganzzahlig
primadresse: 0 bis 15, ganzzahlig
sekadresse: abhängig vom angesprochenen Gerät
fileparam: String, entsprechend den Vorschriften
des angesprochenen Gerätes

Abkürzung: oP

Beispiel:

```
10 open 4,4
20 cmd 4
30 rem *** Alle 'print'-Anweisungen im Programm
40 rem *** werden auf einem Drucker ausgegeben.
50 open 1,8,0,"0:prg.01"
60 rem *** Die (Programm-) Datei 'prg.01' wird
70 rem *** (im Laufwerk 0 der Station Nr. 8)
80 rem *** zum Lesen angemeldet (Sek.-adr. = 0!)
90 open 2,9,2,"$1:adat1,s,w"
100 rem *** Die bereits bestehende sequentielle
110 rem *** Datei 'adat1' im Laufwerk 1 der Station
120 rem *** Nr.9 wird zum Überschreiben angemeldet.
130 open 3,8,2,"1:xindexfile,l,"+chr$(66)
140 open 4,8,3,"1:yindexfile,l,b"
150 rem *** Die beiden relativen Dateien 'xindexfile'
160 rem *** und 'yindexfile' (beide im Laufwerk 1
170 rem *** der Diskettenstation Nr. 8) werden zum
180 rem *** Lesen und Schreiben angemeldet. Die
190 rem *** Sätze sind bei beiden Dateien jeweils
200 rem *** 66 Byte lang (chr$(66) = ASCII-Zeichen 'b')
```

3.1.26 RECORD -----

Format: RECORD# p1 p2 [p3]
 p1: dateinummer
 p2: satznummer
 p3: byteposition

Modus: Direkt / Programm

Zweck: Positionieren auf einen bestimmten Satz (und
 Byteposition) bei einer Relativdatei

Voraussetzung: Die angesprochene Datei ist eine eröffnete
 Relativdatei.

Beschreibung: Durch dieses Kommando ist es möglich, innerhalb
 einer relativen Datei auf einen beliebigen Satz
 zu positionieren. Ferner kann noch auf ein
 bestimmtes Byte innerhalb des Satzes positioniert
 werden. 'dateinummer' bezeichnet hierbei die
 Datei, in welcher positioniert werden soll;
 'satznummer' bzw. 'byteposition' spezifizieren
 die Position in der Datei.

Defaultwerte: byteposition: 1

Wertebereich: dateinummer: 1 bis 255, ganzzahlig
 satznummer: 0 bis 65535, ganzzahlig
 byteposition: 1 bis 254, ganzzahlig

Abkürzung: reC#

Beispiel:

```
10 open2,8,2,"0:reldat,1,"+chr$(120)
20 rem *** Eröffnen einer Relativdatei
30 rem *** mit Sätzen der Länge 120
40 record#2,25
50 input#2,sa:input#2,by
60 rem *** In dieser Datei wird auf den Anfang
70 rem *** des Satzes Nr. 25 positioniert und
80 rem *** zwei Werte (sa,by) eingelesen.
90 record#2,sa,by
100 rem *** Es wird auf den Satz und die Byte-
110 rem *** position, welche zuvor eingelesen
120 rem *** wurden, positioniert.
```

3.1.27 RENAME

Format: RENAME [p1] p2 [p3]
 p1: D drive
 p2: namealt T0 nameneu
 p3: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Umbenennen von Dateien

Voraussetzung: Die umzubenennende Datei ist nicht mehr geöffnet

Beschreibung: Die Datei mit dem Namen 'namealt', die sich auf der Station 'unit' im Laufwerk 'drive' befindet, erhält den Namen 'nameneu'. An der Datei selbst wird keine Änderung vorgenommen.

Die Stellung des Parameters p3 ist beliebig. Werden keine Konstanten, sondern Variable oder Ausdrücke verwendet, so sind diese in runde Klammern einzuschließen.

Defaultwerte: drive: 0
 unit: 8

Wertebereich: namealt: String der Länge 1 bis 16
 nameneu: String der Länge 1 bis 16
 drive: 0 oder 1, ganzzahlig
 unit: 8 bis 15, ganzzahlig

Abkürzung: reN

Beispiel:

```
10 rename u8, d0, "master" to "backup"
20 rem *** Die Datei 'master' im Laufwerk 0 der
30 rem *** Station Nr. 8 wird in 'backup' umbenannt.
40 input "Alter Name";na$
50 input "Neuer Name";nn$
60 input "Laufwerk ";dr
70 input "Station ";un
80 rename d(dr) on u(un), (na$) to (nn$)
90 rem *** Die durch die eingelesenen Parameter
100 rem *** 'namealt', 'drive' und 'unit' bestimmte
110 rem *** Datei wird umbenannt.
```


3.1.28 RUN

Format: RUN [p1]
 p1: zeilennummer

Modus: Direkt / Programm

Zweck: Starten eines Programmes

Voraussetzung: Falls p1 angegeben: Zeile existiert

Beschreibung: Mit diesem Kommando wird ein sich im Speicher befindendes Programm gestartet. Wird eine Zeilennummer angegeben, so wird das Programm in dieser Zeile begonnen. Wird ein Programm mit diesem Kommando gestartet, so werden alle bisher vorhandenen Variablen gelöscht (Kaltstart; s. auch Kap 3.2.xx, 'GOTO'). Für 'zeilennummer' kann keine Variable verwendet werden!

Defaultwerte: zeilennummer: Erste vorhandene Zeile im Programm

Wertebereich: zeilennummer: 0 bis 63999, ganzzahlig

Abkürzung: rU

Beispiel: .
 .
 1000 if fre(3) 1000 then run 250
 1010 rem *** Sind im Segment Nr. 3 jetzt weniger
 1020 rem *** als 1000 Byte frei, dann wird das
 1030 rem *** Programm neu gestartet (ab Zeile 250)
 .
 .

3.1.29 SAVE

Format: SAVE p1 [p2]
 p1: \$ drive: name
 p2: unit

Modus: Direkt / Programm

Zweck: Sichern eines Programmes

Voraussetzung: Beim Sichern auf bereits existierende Datei: Diese
Datei muß vom Typ 'PRG' sein.

Beschreibung: Dieses Kommando bewirkt das Sichern eines Programmes.
Der Name des Programmes, das gesichert werden soll,
ist im Parameter 'name' anzugeben. Der ganze Para-
meter 'p1' ist ein String, der sich aus dem Laufwerk
(bei Diskettenstationen) und dem Namen zusammensetzt.
Beide Angaben sind durch einen Doppelpunkt von einan-
der zu trennen. Die Geräteadresse der Station, von
welcher das Programm geladen wird, ist in 'unit'
anzugeben.

Soll das Programm auf eine bereits bestehende
Datei gesichert werden, so ist unmittelbar vor dem
Laufwerk das Zeichen '\$' (Klammeraffe) anzugeben.
Wird der Klammeraffe angegeben, obwohl noch keine
Datei dieses Namens existierte, so wird er ignoriert.

Defaultwerte: drive: Das zuletzt angesprochene Laufwerk;
 beim Einschalten: 0
 unit: 1

Wertebereich: drive: 0 oder 1
 name: (Teil-)String der Länge 1 bis 16
 unit: 1 bis 15, ganzzahlig

Abkürzung: sA

Beispiel:

```
10 kl$="§"
20 lw=1
30 un=11
40 dn$="prog.01"
50 save str$(lw)+":"+dn$,un
60 if ds=63 then save kl$+str$(lw)+":"+dn$,un
70 rem *** Dieses Programm wird unter dem Namen
80 rem *** 'prog.01' auf die Diskette im Lauf-
70 rem *** werk 1 der Station Nr. 11 gesichert.
80 rem *** Existierte die Datei bereits (ds=63),
90 rem *** so wird sie überschrieben.

save "§1:prog.01",11      (wie oben)
```

3.1.30 SCRATCH

Format: SCRATCH p1 [p2] [p3]
 p1: dateiname
 p2: D drive
 p3: U unit
 ON U unit / Komma kann entfallen

Modus: Direkt / Programm

Zweck: Löschen von Dateien

Voraussetzung: Datei(en) nicht löschgeschützt

Beschreibung: Die Datei 'dateiname' auf dem Laufwerk 'drive' der Station 'unit' wird (logisch) gelöscht. Die von ihr belegten Blöcke werden freigeben und diese Datei im Directory der Diskette als gelöscht vermerkt. Wird dieses Kommando im Direktmodus gegeben, so erscheint die Anfrage 'are you shure?'. Wird sie mit 'y' oder 'yes' beantwortet, so wird das Kommando ausgeführt; jede andere Eingabe bricht das Kommando vor der Ausführung ab. Diese Anfrage unterbleibt, wenn das Kommando von einem Programm aufgerufen wird.

Die Reihenfolge der Parameter ist beliebig. Werden keine Konstanten, sondern Variable oder Ausdrücke verwendet, so sind diese in runde Klammern einzuschließen.

Defaultwerte: drive: 0
 unit: 8

Wertebereich: dateiname: String der Länge 1 bis 16
 drive: 0 oder 1
 unit: 8 bis 15, ganzzahlig

Abkürzung: sC

Beispiel: 10 input "Name ";na\$
 20 input "Drive ";dr
 30 input "Unit ";un
 40 scratch (na\$),d(dr)onu(un)
 50 rem *** Die durch na\$, dr und un bestimmte
 60 rem *** Datei wird gelöscht

3.1.31 VERIFY

```
Format:      VERIFY p1 [p2]
             p1:  drive:  name
             p2:  unit
```

Modus: Direkt / Programm

Zweck: Überprüfen eines gesicherten Programmes

Voraussetzung:-

Beschreibung: Dieses Kommando bewirkt das Prüfen eines Programmes. Dabei wird festgestellt, ob das angegebene Programm mit dem, welches sich im Speicher des Rechners befindet, übereinstimmt. Es können somit eventuell aufgetretenen Übertragungsfehler erkannt werden. Der Name des Programmes, das überprüft werden soll, ist im Parameter 'name' anzugeben. Der ganze Parameter 'p1' ist ein String, der sich aus dem Laufwerk (bei Diskettenstationen) und dem Namen zusammensetzt. Beide Angaben sind durch einen Doppelpunkt voneinander zu trennen. Die Geräteadresse der Station, in welcher das zu prüfende Programm geladen wird, ist in 'unit' anzugeben.

```
Defaultwerte: drive:      0
               unit:      1
```

```
Wertebereich: drive:      0 oder 1
               name:      (Teil-)String der Länge 1 bis 16
               unit:       1 bis 15, ganzzahlig
```

Abkürzung: vE

[illegible]

3.2 Befehle (Statements)

Inhalt

3.2.1	clr	66
3.2.2	data	68
3.2.3	def	70
3.2.4	dim	72
3.2.5	dispose	74
3.2.6	end	76
3.2.7	for	77
3.2.8	get	79
3.2.9	get#	81
3.2.10	gosub	82
3.2.11	goto	83
3.2.12	if	84
3.2.13	input	86
3.2.14	input#	88
3.2.15	let	90
3.2.16	next	92
3.2.17	on	94
3.2.18	poke	96
3.2.19	print (...)	97
3.2.20	print#(...)	101
3.2.21	pundef	102
3.2.22	read	104
3.2.23	rem	106
3.2.24	restore	107
3.2.25	resume	108
3.2.26	return	110
3.2.27	stop	111
3.2.28	sys	112
3.2.29	trap	113
3.2.30	wait	115

3.2.1 CLR

Format: CLR

Modus: Direkt / Programm

Zweck: Löschen von Variablen

Voraussetzung:-

Beschreibung: Dieser Befehl bewirkt das Löschen von Feldern, einfachen Variablen, Stringvariablen sowie von definierten Funktionen. Dabei werden die Zeiger, die auf Anfang bzw. Ende von Speicherbereichen verweisen, mit ihren Anfangswerten belegt. (Ausgenommen BASIC-TEXT-Zeiger).

Außerdem wird der Stack bereinigt, d.h., die Einträge, die beim Erreichen einer 'FOR'-Anweisung dort abgelegt werden, werden gelöscht. Das gleiche gilt für Unterprogrammsprünge ('GOSUB'). Ferner wird der Zeiger, welcher auf die derzeitige 'DATA'-Zeile verweist, auf die erste 'DATA'-Zeile im Programm gesetzt.

Defaultwerte: -

Wertebereich: -

Abkürzung: cL

Beispiel:

```
10 data 1,2
20 data 3,4,5
30 read a,b,c
40 clr
50 read d,e
60 print a;b;c;d;e
70 for i=0 to 10
80 clr
90 next
run
  0  0  0  1  2

?next without for in 90
ready.
```

3.2.2 DATA

Format: DATA (p)
p: wert

Modus: Programm

Zweck: Bereitstellen von Variablenwerten

Voraussetzung:-

Beschreibung: Mit dieser Anweisung werden Werte für Variablen bereitgestellt, die während des Programmablaufs mit der 'READ'-Anweisung eingelesen werden. Als Parameter (d.h., als Werte) sind alle die Werte zugelassen, die für die zugehörige Variable erlaubt ist.

Die Zuordnung von Variablen zu den Werten der 'DATA'-Zeile(n) geschieht folgendermaßen:

Der ersten Variablen, die in der ersten durchlaufenen(!) 'READ'-Anweisung steht, wird der erste Wert der ersten gültigen (!) 'DATA'-Zeilen zugewiesen, der zweiten Variablen der zweite Wert usw. Als erste gültige 'DATA'-Zeile ist die erste vorhandene 'DATA'-Zeile bzw. bei Verwendung der 'RESTORE'-Anweisung die dort angegebene 'DATA'-Zeile zu verstehen.

Es ist darauf zu achten, daß Werte in den 'DATA'-Anweisungen mit den zugehörigen Variablen verträglich sind, daß also z.B. nicht einer REAL-Variablen ein String zugeordnet wird.

Defaultwerte: -

Wertebereich: Je nach zugehörigem Variablentyp alle zugelassenen Werte

Abkürzung: daT

Beispiel:

```
10 data a,b,"c,d",1,100
20 read x$,y$,z$,v1,v2
30 print x$
40 print y$
50 print z$
60 print v1
70 print v2
run
a
b
c,d
1
100

ready.
```

3.2.3 DEF

Format: DEF FN p1 p2 p3
 p1: name
 p2: (argument)
 p3: = formel

Modus: Programm

Zweck: Definieren von eigenen Funktionen

Voraussetzung:-

Beschreibung: Durch diese Anweisung wird eine vom Benutzer selbst definierte Funktion vereinbart. Sie kann fortan im Programm wie eine der Standardfunktionen (wie z.B. $\sin(x)$) verwendet werden. Es können nur reelwertige Funktionen definiert werden, also keine Integer- oder Stringfunktionen.

'name' gibt dabei den Namen an, unter dem die Funktion angesprochen wird. Beim Aufruf müssen diesem Namen die Zeichen 'FN' vorausgehen. Der Name muß ein zulässiger BASIC-Name sein.

'argument' dient als Platzhalter in dieser Vereinbarung. Die dort verwendete Variable ist keine "richtige" Variable, d.h., sie belegt keinen Platz im Rechner. Eine im Programm eventuell vorkommende Variable dieses Namens wird nicht benutzt.

'formel' gibt die Rechenvorschrift an, welche das gewünschte Ergebnis liefert. Beim Aufruf der Funktion wird überall dort, wo das 'argument' in der 'formel' erscheint, der Wert eingesetzt, der beim Aufruf an der Position des 'argument's steht.

Defaultwerte: -

Wertebereich: name: gültiger BASIC-Name
 argument: gültiger BASIC-Name
 formel: zulässige Formel

Abkürzung: dE

Beispiel:

```
10 def fn sh(x)=(exp(x)-exp(-x))/2
20 rem *** sinus hyperbolicus
30 def fn f1(x)=log((x+1)/(x-1))/2
40 rem *** arcus cotangens hyperbolicus
50 for i=11 to 20
60 print i, fn sh(i/10), fn f1(i/10)
70 next
80 rem *** Stellt eine Tabelle der obigen
90 rem *** Funktionen auf
```

3.2.4 DIM

Format: DIM (p)
 p: name(dimensionen)

Modus: Direkt / Programm

Zweck: Definieren von Feldern und Matrizen

Voraussetzung:-

Beschreibung: Durch diese Anweisung werden Felder oder Matrizen vereinbart. Die Angaben in den Klammern bezeichnen dabei die Größe des Feldes bzw. der Matrix. Jede Dimension beginnt mit dem Index 0, d.h. die Anweisung 'dim a(10)' definiert ein (eindimensionales) Feld mit 11 Elementen, nämlich a(0) bis a(10).

Die Anzahl der Dimensionen wird begrenzt durch die Größe des verfügbaren Speichers bzw. durch den freien Platz, der für die Verwaltung zur Verfügung gestellt wird. Die theoretische Obergrenze liegt bei 93 Dimensionen. Zur Berechnung des benötigten Speichers s. Kapitel Datenmanipulationen.

Als Indizes sind nur ganze, nicht negative Zahlen zulässig. Reelle Indexangaben werden zur nächsten ganzen Zahl abgerundet. Eindimensionale Felder mit maximal 11 Elementen brauchen nicht mit einer 'DIM'-Anweisung vereinbart werden.

Defaultwerte: -

Wertebereich: name: gültiger BASIC-Name
 dimensionen: 0 bis 32767, ganzzahlig

Abkürzung: dI

Beispiel:

```
10 dim a(12,10)
20 rem *** Eine Matrix mit 143 real-Elementen
30 rem *** (143=(12+1)*(10+1)) wird deklariert.
40 dim i%(100)
50 rem *** Ein (eindimensionales) Feld mit 101
60 rem *** ganzzahligen Elementen wird definiert.
70 dim s$(2,15)
80 rem *** Ein zweidimensionales Feld wird defi-
90 rem *** niert. Die Elemente sind Strings.
```

3.2.5 DISPOSE

Format: DISPOSE p
p: FOR
GOSUB

Modus: Programm

Zweck: Bereinigen des Stack bei Fehlerbehandlung

Voraussetzung:-

Beschreibung: Zur Erläuterung wird die Wirkung dieses Befehls im Zusammenhang mit einer 'FOR'-Schleife gezeigt. Bei Unterprogrammen ('GOSUB') gilt sinngemäß dasselbe.

Die Anweisung dieses Befehls bewirkt, daß der letzte Eintrag zu einer 'FOR'-Schleife auf dem Stack gelöscht wird. Dieser Befehl ist dann anzuwenden, wenn aus einer 'FOR'-Schleife herausgesprungen wird, d.h., wenn sie vor dem Erreichen des Endekriteriums der Schleife verlassen wird. Dies wird oft bei der Verwendung des 'TRAP'-Befehls zur Behandlung eines Fehlers der Fall sein (s. dort). Durch den 'DISPOSE'-Befehl wird somit vermieden, daß nicht mehr benötigte Informationen auf dem Stack angesammelt werden und diesen somit unnötig belasten.

Einer der beiden möglichen Parameter muß angegeben werden !

Defaultwerte: -

Wertebereich: p: FOR oder GOSUB

Abkürzung: diS

Beispiel:

```
10 trap 120
20 for i=1 to 10
30 for j=1 to 10
40 x=20/(15-i-j)
50 print x
60 next j
70 next i
80 print "okay"
90 end
100 print "Fehler"
110 end
120 print "Zeile: ";el;" Fehler: ";err$(er)
130 print "i=";i;" j=";j
140 dispose for: dispose for
150 goto 100
160 rem *** Beim Auftreten eines Fehlers werden nach
170 rem *** der Ausgabe einer Fehlermeldung beide
180 rem *** (geschachtelten) Schleifen verlassen.
```

3.2.6 END

Format: END

Modus: Programm

Zweck: Beenden eines Programms

Voraussetzung:-

Beschreibung: Die Anwendung dieses Kommandos bewirkt das Beenden des laufenden Programms. Der Rechner kehrt dadurch in seinen Grundzustand zurück. Am Ende eines Programmtextes kann dieser Befehl fehlen.

Defaultwerte: -

Wertebereich: -

Abkürzung: eN

Beispiel: 10 for i=1 to 10
 20 input x
 30 if x=0 then dispose for: end
 40 print sin (1/x)
 50 next
 60 rem *** Es werden 10 Werte eingelesen und
 70 rem *** der Sinus vom Kehrwert der eingelesenen
 80 rem *** Zahl gedruckt. Falls der eingelesene
 90 rem *** Wert jedoch Null ist, so wird das
 100 rem *** Programm jedoch (vorzeitig) verlassen.

3.2.7 FOR

Format: FOR p1 p2 [p3]
 p1: variable = anfangswert
 p2: TO endwert
 p3: STEP schrittweite

Modus: Direkt / Programm

Zweck: Bilden einer Programmschleife

Voraussetzung:-

Beschreibung: Wenn eine bestimmte Folge von Anweisungen mehrfach wiederholt werden soll, dann ist die Verwendung einer 'FOR'-Schleife angeraten. Sie bewirkt folgendes:

Der einfachen reellen Variablen 'variable' wird der Wert 'anfangswert' zugewiesen. Diese Variable wird Schleifenvariable genannt. Sie darf keine dimensionierte Variable sein. Dann werden die folgenden Anweisungen ausgeführt, bis die zu dieser 'FOR'-Schleife gehörende 'NEXT'-Anweisung (s. dort) erreicht wird. Jetzt wird die Schleifenvariable um den Wert 'schrittweite' erhöht und dieser neue Wert mit dem 'endwert' verglichen. Dieser Vergleich wird Endekriterium genannt. Ist der Wert der Schleifenvariablen nun größer als dieser Endwert (bei positiver Schrittweite; bei negativer Schrittweite kleiner als der Endwert), so wird die Schleife verlassen und das Programm hinter der zugehörigen 'NEXT'-Anweisung fortgesetzt. Anderenfalls werden die Anweisungen zwischen der 'FOR'- und der 'NEXT'-Anweisung nochmals ausgeführt.

Es können mehrere 'FOR'-Anweisungen ineinander verschachtelt werden. Dabei wird eine 'äußere' Schleife erst dann fortgesetzt, wenn die 'innere' Schleifen beendet sind.

Die Schleifenvariable kann im Programm wie jede normale Variable verwendet werden. Es können ihr also auch Werte zugewiesen werden; die Konsequenzen einer solchen Zuweisung müssen jedoch sorgfältig überlegt werden.

Defaultwerte: schrittweite: 1

Wertebereich: anfangswert: gültiger numerischer Wert
 endwert: gültiger numerischer Wert
 schrittweite: gültiger numerischer Wert

Abkürzung: f0
 stE

Beispiel:

```

10 rem ***** Programm 1
20 dim a(20,15),b(15,10),c(20,10)
30 for i1=1 to 20
40 for i2=1 to 10
50 su=0
60 for i3=1 to 15
70 su=su+a(i1,i3)*b(i3,i2)
80 next i3
90 c(i1,i2)=su
100 next i2,i1
110 rem *** Matrizenmultiplikation: Matrix a mal
120 rem *** Matrix b = Matrix c.
130 rem *** Dieses Programm bewirkt dasselbe wie das
140 rem *** das folgende Programm:

1010 rem ***** Programm 2
1020 dim a(20,15),b(15,10),c(20,10)
1030 i1=1
1040 i2=1
1050 su=0
1060 i3=1
1070 su=su+a(i1,i3)*b(i3,i2)
1080 i3=i3+1
1085 if i3 =15 then 1070
1090 c(i1,i2)=su
1100 i2=i2+1
1105 if i2 =10 then 1050
1110 i1=i1+1
1115 if i1 =20 then 1040

```

3.2.8 GET

Format: GET (p)
 p: stringvar

Modus: Programm

Zweck: Lesen einzelner Zeichen von der Tastatur

Voraussetzung:-

Beschreibung: Die Ausführung dieses Befehls bewirkt, daß das nächste Zeichen, welches von der Tastatur eingegeben wurde, an die Stringvariable 'stringvar' übergeben wird. Die eingegebenen Zeichen erscheinen dabei nicht auf dem Bildschirm. War zum Zeitpunkt der Ausführung des Befehls noch kein Zeichen eingegeben worden, so wird der Stringvariablen ein leerer String übergeben. Im Gegensatz zum 'INPUT'-Befehl verschwindet der Cursor während der Ausführung des 'GET'-Befehls.

Werden mehrere Stringvariablen angegeben, so erhalten sie Werte in der gleichen Reihenfolge, wie die Zeichen eingegeben wurden.

Grundsätzlich ist zu bemerken, daß anstelle einer Stringvariablen auch eine Zahlenvariable angegeben werden kann. Dabei werden jedoch nur eingegebene Ziffern richtig übertragen; die Eingabe von '.', '+', '-', 'e' oder ' ' liefert Null als Zahlenwert, ein Komma bzw. der Doppelpunkt bewirkt die Ausgabe der Meldung 'extra ignored'. Jede andere Taste führt zu der Fehlermeldung 'syntax error'.

Mit 'GET' können alle Tasten, die auf der Tastatur sind, eingegeben werden (außer der STOP-Taste). Somit können z.B. auch die Cursor-Steuertasten eingelesen werden. Bei der Eingabe einer Funktionstaste wird die unter ihr abgelegte Zeichenkette verarbeitet.

Defaultwerte: -

Wertebereich: stringvar: gültiger Stringvariablenname

Abkürzung: gE

Beispiel:

```
10 get a$: ifa$="" then 10
20 rem *** Warten, bis ein Zeichen eingegeben wird
30 print a$,asc(a$)
40 goto 10
50 rem *** Eingelesene Zeichen werden auf dem Bild-
60 rem *** schirm mit ihrem ASCII-Wert ausgegeben
70 rem *** (auch Steuertasten und Funktionstasten!)
```

3.2.9 GET#

Format: GET# p1 (p2)
 p1: dateinummer
 p2: stringvar

Modus: Programm

Zweck: Lesen von Zeichen von einem beliebigen Eingabegerät

Voraussetzung: Eingabegerät eröffnet

Beschreibung: Grundsätzlich wirkt dieser Befehl genauso wie der 'GET'-Befehl. Im Gegensatz zu diesem kann der 'GET#'-Befehl aber von einem beliebigen Eingabegerät lesen (Datei, Bildschirm). Wird als Eingabegerät die Tastatur gewählt (Primäradresse Null), so ist die Wirkung identisch mit dem einfachen 'GET'-Befehl. Bei der Eingabe vom Bildschirm (Primäradresse 3) wird ab der nächsten Druckposition gelesen. Wird beim Lesen von einer Datei über das Dateiende hinaus weitergelesen, so wird der Code chr\$(13) zurückgeliefert.

Defaultwerte: -

Wertebereich: stringvar: gültiger Stringvariablenname

Abkürzung: gE#

Beispiel:

```

10 open 2,8,2,"0:testdat,s,r"
20 get#2,a$
30 if a$="" then a$=chr$(0)
40 fs=st
50 print a$;
60 if fs=0 then 20
70 close 2
80 rem *** Der Inhalt der Datei 'testdat' wird
90 rem *** auf dem Bildschirm aufgelistet.
```

3.2.10 GOSUB

Format: GOSUB p
 p: zeilennummer

Modus: Direkt / Programm

Zweck: Aufruf eines Unterprogramms

Voraussetzung:-

Beschreibung: Die Ausführung dieses Befehls bewirkt den Aufruf eines Unterprogrammes. Das bedeutet, daß der momentane Programmablauf unterbrochen wird; es wird mit der ersten Anweisung fortgefahren, die in der durch 'zeilennummer' genannten Zeile steht. Sobald nun eine 'RETURN'-Anweisung erreicht wird, wird das Unterprogramm beendet und mit der nächsten Anweisung hinter dem 'GOSUB'-Befehl fortgefahren.

Defaultwerte: -

Wertebereich: zeilennummer: 0 bis 63999, ganzzahlig

Abkürzung: goS

Beispiel: .
 .
 200 input#1,r
 210 input#1,a1
 220 gosub 1000
 .
 .
 1000 x=r*cos(a1)
 1010 y=r*sin(a1)
 1020 return
 1030 rem *** Dieses Unterprogramm rechnet eine
 1040 rem *** Reihe von 'Polar'-Koordinaten in
 1050 rem *** Standard-Koordinaten um

3.2.11 GOTO -----

Format: GOTO p
 p: zeilennummer

Modus: Direkt / Programm

Zweck: Ausführen eines Programmsprunges

Voraussetzung:-

Beschreibung: Die Ausführung dieses Befehls bewirkt, daß die Ausführung des Programmes an einer anderen Stelle fortgesetzt wird. Es wird mit der ersten Anweisung fortgefahren, die in der durch 'zeilennummer' gekennzeichneten Zeile steht.

Wird dieser Befehl im Direkt-Modus gegeben, so wird das Programm mit der angegebenen Zeile begonnen. Im Gegensatz zum 'RUN'-Kommando werden jedoch bereits vorhandene Variablen weiterverwendet.

Defaultwerte: -

Wertebereich: zeilennummer: 0 bis 63999, ganzzahlig

Abkürzung: g0

Beispiel: 10 open 2,8,2,"0:testdat,s,r"
 20 get#2,a\$
 30 if a\$="" then a\$=chr\$(0)
 40 fs=st
 50 print a\$;
 60 if fs 0 then close 2: end
 70 goto 20
 80 rem *** Der Inhalt der Datei 'testdat' wird
 90 rem *** auf dem Bildschirm aufgelistet.
 100 rem *** Wenn Zeile 70 erreicht wird, dann
 110 rem *** wird das Programm in Zeile 20 fortgesetzt.

3.2.12 IF

Format: IF p1 p2 [p3]
 p1: bedingung
 p2: THEN anweisungen
 THEN zeilennummer
 GOTO zeilennummer
 p3: :ELSE anweisungen
 :ELSE zeilennummer

Modus: Direkt / Programm

Zweck: Bedingte Ausführung von Anweisungen

Voraussetzung:-

Beschreibung: Beim Erreichen einer 'IF'-Anweisung wird zuerst der logische Wert berechnet, der an der Stelle 'bedingung' steht. Dieser Wert muß eine Aussage sein, welche den logischen Wert 'wahr' oder 'falsch' liefert. Es sind nun zwei Fälle zu unterscheiden:

1. Der Teil ':ELSE ...' wird nicht verwendet:
 Ist die Bedingung hinter der 'IF'-Abfrage logisch wahr, so wird (werden) die Anweisung(en) ausgeführt, die hinter 'THEN' stehen. Steht an dieser Stelle nur eine Zeilennummer oder anstatt 'THEN' die Anweisung 'GOTO', so ist die Wirkung wie bei 'THEN GOTO ...'. Steht hinter 'THEN' keine Sprunganweisung, so wird anschließend in der nächsten Zeile fortgefahren. Liefert der Ausdruck hinter 'IF' den logischen Wert falsch, so wird der Rest der Zeile übersprungen und gleich mit der nächsten Zeile fortgefahren.
2. Der Teil ':ELSE ...' wird verwendet:
 Ergibt der logische Ausdruck hinter 'IF' den Wert 'wahr', so werden die Anweisungen hinter 'THEN' bis zu dem Teil ':ELSE' ausgeführt und anschließend mit der nächsten Zeile fortgefahren, falls zwischen 'THEN' und 'ELSE' keine Sprunganweisung steht. Ergibt sich dagegen der logische Ausdruck zu 'falsch', so werden die Anweisungen hinter 'ELSE' ausgeführt und dann in der nächsten Zeile fortgefahren.

Zu bemerken ist dazu folgendes:
 Stehen in einer Zeile mehrere 'IF'-Anweisungen mit

mehreren ':ELSE'-Teilen, so wird wenn eine 'IF'-Abfrage den Wert 'falsch' liefert, immer mit dem ersten ':ELSE'-Teil fortgefahren. Die Anweisungen hinter einem zweiten ':ELSE'-Teil werden niemals ausgeführt.

Defaultwerte: -

Wertebereich: bedingung: gültiger logischer Ausdruck
 anweisungen: gültige BASIC-Anweisung(en)
 zeilennummer: 0 bis 63999, ganzzahlig

Abkürzung: tH
 g0
 eL

Beispiel:

```

10 input a,b,c
20 if a < b then print "a kleiner b"
30 if a*a+b*b=c*c then print "rechtwinklig"
40 if a+b=c then 100: else print "ungleich"
50 l1=a b
60 l2=b c
70 if l1 then if l2 then 200: else 300: else 400
100 print "100"
200 print "200"
300 print "300"
350 end
400 print "400"
500 rem *** Die Anweisung 400 wird von Zeile 70 aus
510 rem *** nie erreicht, da (falls l1 oder l2
520 rem *** logisch falsch sind) immer der Teil
530 rem *** ':else 300' ausgeführt wird.

```

3.2.13 INPUT

Format: INPUT p1 (p2)
 p1: string;
 p2: variable

Modus: Programm

Zweck: Einlesen von Werten

Voraussetzung:-

Beschreibung: Dieser Befehl dient zum Einlesen von Werten von der Tastatur. Es können sowohl Zahlen als auch Strings eingelesen werden. Der Parameter 'string' muß eine Stringkonstante sein. Wird diese Anweisung ausgeführt, dann wird dieser 'string', gefolgt von einem Fragezeichen, ausgegeben. Der Benutzer kann nun seine Eingabe tätigen. Die Eingabedaten werden erst übernommen, wenn die 'RETURN'-Taste gedrückt wird. Sollen mehrere Werte eingegeben werden, so müssen Zahlen durch ein nicht-numerisches Zeichen (i.A. durch Komma) getrennt werden. Strings können durch Komma oder Doppelpunkt voneinander getrennt werden. Wenn ein Komma oder ein Doppelpunkt mit eingelesen werden soll, so ist die gesamte einzulesende Zeichenfolge in Stringbegrenzer (Gänsefüßchen oben) einzuschließen.

Die gesamte Eingabe für eine 'INPUT'-Anweisung darf maximal 160 Zeichen lang sein. Wird 'string' angegeben, so muß dahinter ein Strichpunkt (';') anstelle eines Kommas stehen!

Wird eine leere Eingabe gemacht, so behalten die Variablen, welche einzulesen sind, ihre alten Werte. Werden weniger Werte angegeben, als die 'INPUT'-Anweisung erwartet, so werden die fehlenden Werte erneut angefragt. Dieser Zustand ist durch die Ausgabe von zwei Fragezeichen gekennzeichnet. Der 'string' wird dabei jedoch nicht wieder ausgegeben. Werden Daten eingegeben, die nicht mit den entsprechenden Variablen kompatibel sind (wenn also

z.B. Strings eingegeben werden, wo Zahlen erwartet werden), so gibt der Rechner die Meldung
 'redo from start'
 aus und wiederholt die gesamte 'INPUT'-Anweisung.

Defaultwerte: string: leerer String

Wertebereich: string: durch Gänsefüßchen eingeschlossene
 Zeichenfolge
 variable: gültiger BASIC-Variablenname

Abkürzung: -

Beispiel: 10 input a,b
 20 print a,b
 30 input "zweite eingabe";c,d
 40 print c,d
 run
 ? 12
 ?? 34
 12 34
 zweite eingabe? 47,x
 ?redo from start
 zweite eingabe? 47,145
 47 145

3.2.14 INPUT#

Format: INPUT# p1 (p2)
 p1: dateinummer
 p2: variable

Modus: Programm

Zweck: Einlesen von Werten

Voraussetzung: Angesprochenes Eingabegerät ist zum Lesen eröffnet

Beschreibung: Dieser Befehl dient zum Einlesen von Werten von einem beliebigen Eingabegerät. Es können (wie beim normalen 'INPUT') sowohl Zahlen als auch Strings eingelesen werden. Sollen mehrere Werte eingelesen werden, so müssen Zahlen durch ein nicht-numerisches Zeichen (i.A. durch Komma) getrennt werden. Strings können durch Komma oder Doppelpunkt voneinander getrennt werden. Wenn ein Komma oder ein Doppelpunkt mit eingelesen werden soll, so ist die gesamte einzulesende Zeichenfolge in Stringbegrenzer (Gänsefüßchen oben) einzuschließen.

Die gesamte Eingabe für eine 'INPUT#'-Anweisung darf maximal 160 Zeichen lang sein.

Werden Daten eingegeben, die nicht mit den entsprechenden Variablen kompatibel sind (wenn also z.B. Strings eingegeben werden, wo Zahlen erwartet werden), so gibt der Rechner die Meldung 'file data error in ...' aus und beginnt die Fehlerbehandlung.

Es ist zu beachten, daß als Trennzeichen zwischen Zahlen (oder Strings) ein Komma, ein Doppelpunkt oder Zeilenende (CR) gelten. Leerzeichen sind keine Trennzeichen! Dies ist insbesondere bei 'PRINT#' zu beachten (s. Beispiel), wenn die hiermit ausgegebenen Werte später wieder mit 'INPUT#' eingelesen werden sollen.

Werden nicht alle Werte, die in einer Zeile enthalten sind eingelesen, so wird der Rest der Zeile beim nächsten 'INPUT#' überlesen. Sollen auch Trennzei-

zeichen eingelesen werden, so ist beim Schreiben der Daten mit 'PRINT#' an Anfang und Ende die Zeichen 'Gänsefüßchen oben' zu erzeugen (chr\$(34)).

Defaultwerte:

Wertebereich: variable: gültiger BASIC-Variablenname

Abkürzung: iN

Beispiel:

```

10 open 1,8,2,"0:test.01,s,w"
20 print#1,12
30 print#1,34,56
40 print#1,"abc,def" :rem das sind zwei strings!
50 print#1,chr$(34);"uvw,xyz";chr$(34)
60 close 1
70 open 1,8,2,"0:test.01,s,r"
80 input a: print a
90 input b: print b
100 input a$: print a$
110 input b$: print b$ :rem 'def' wird überlesen!
120 close 1
run
12
3456
abc
uvw,xyz

```

3.2.15 LET

Format: LET p1 p2
 p1: variable
 p2: = ausdruck

Modus: Direkt / Programm

Zweck: Zuweisen von Werten an Variablen

Voraussetzung:-

Beschreibung: Durch die 'LET'-Anweisung werden Variablen Werte zugewiesen. Auf der linken Seite des Gleichheitszeichens muß der Name einer BASIC-Variablen stehen. Dies kann eine einfache oder eine indizierte Variable sein. Die Zeichenfolge 'LET' am Anfang dieser Anweisung kann entfallen, sie ist nur von historischer Bedeutung.

Auf der rechten Seite des Gleichheitszeichens muß ein Ausdruck stehen, der mit dem Typ der Variablen links des Gleichheitszeichens übereinstimmen muß. D.h., es ist nicht möglich, links eine Stringvariable und rechts ein arithmetischer Ausdruck einzusetzen. Eine genaue Beschreibung der zulässigen Ausdruckformen s. im Kapitel Datenmanipulationen.

Diese Anweisung bewirkt nun, daß der Ausdruck auf der rechten Seite berechnet wird und dieser Wert der Variablen auf der linken Seite zugewiesen wird.

Defaultwerte: -

Wertebereich: variable: gültiger BASIC-Variablenname
 ausdruck: gültiger BASIC-Ausdruck

Abkürzung: 1E

Beispiel:

```
10 let a=5
20 rem *** Die Variable 'a' erhält den Wert 5
30 c=sqr(a*a+b*b)
40 rem *** Der Variablen 'c' wird die Wurzel aus
50 rem *** der Summe der Quadrate von 'a' und 'b'
60 rem *** zugewiesen.
70 input x$,y$
80 a$=x$+"---"+y$
90 rem *** Der String, bestehend aus der ersten so-
100 rem *** eben eingelesenen Zeichenfolge, drei Minus-
110 rem *** zeichen und der zweiten Zeichenfolge wird
120 rem *** der Stringvariablen 'a$' zugewiesen.
130 b$=right$(str$(c),len(str$(c))-1)
140 rem *** Als Ergebnis wird ein String zurückge-
150 rem *** liefert, der nur aus der Zahl 'c' (als
160 rem *** String!) ohne führende Leerzeichen besteht.
```

3.2.16 NEXT

Format: NEXT [(p)]
 p: schlvar

Modus: Direkt / Programm

Zweck: Endemarkierung einer Programmschleife

Voraussetzung:-

Beschreibung: Diese Anweisung grenzt die Anweisungen ein, aus denen eine 'FOR'-Schleife besteht (s. 'FOR'-Anweisung)

Wird bei der 'NEXT'-Anweisung keine Schleifenvariable ('schlvar') angegeben, so gehört dieses 'NEXT' zu der innersten 'FOR'-Schleife. Wird eine Schleifenvariable angegeben, so muß es diejenige der innersten 'FOR'-Schleife sein. Sollen mehrere Schleifen mit einer 'NEXT'-Anweisung beendet werden, so sind alle Schleifenvariablen anzugeben, und zwar in der Reihenfolge: Schleifenvariable der innersten Schleife, Schleifenvariable der zweitinnersten Schleife, ..., Schleifenvariable der äußersten Schleife.

Defaultwerte: schlvar: Schleifenvariable der innersten Schleife

Wertebereich: schlvar: Relle Variable

Abkürzung: nE

```

Beispiel:      10 rem ***** Programm 1
                20 dim a(20,15),b(15,10),c(20,10)
1-----      30 for i1=1 to 20
I             33 rem *** Diese Schleife (1) ist die äußerste
I             37 rem *** Schleife.
I   2-----  40 for i2=1 to 10
I   I         42 rem *** Diese Schleife (2) ist eine innere
I   I         44 rem *** Schleife im Verhältnis zur Schleife (1)
I   I         46 rem *** und eine äußere Schleife im Ver-
I   I         48 rem *** hältnis zur Schleife (3).
I   I         50 su=0
I   I   3--  60 for i3=1 to 15
I   I   I    63 rem *** Diese Schleife (3) ist die innerste
I   I   I    67 rem *** Schleife.
I   I   I    70 su=su+a(i1,i3)*b(i3,i2)
I   I   ---  80 next i3
I   I       90 c(i1,i2)=su
-----      100 next i2,i1
                110 rem *** Matrizenmultiplikation: Matrix a mal
                120 rem *** Matrix b = Matrix c.
                130 rem *** Dieses Programm bewirkt dasselbe wie das
                140 rem *** das folgende Programm:

1010 rem ***** Programm 2
1020 dim a(20,15),b(15,10),c(20,10)
1030 i1=1
1040 i2=1
1050 su=0
1060 i3=1
1070 su=su+a(i1,i3)*b(i3,i2)
1080 i3=i3+1
1085 if i3 =15 then 1070
1090 c(i1,i2)=su
1100 i2=i2+1
1105 if i2 =10 then 1050
1110 i1=i1+1
1115 if i1 =20 then 1040

```

3.2.17 ON

Format: ON p1 p2 p3
 p1: arithausdruck
 p2: GOTO
 GOSUB
 p3: zeilenliste

Modus: Direkt / Programm

Zweck: Einfaches Verzweigen aufgrund einer Sprungliste

Voraussetzung:-

Beschreibung: Zuerst wird der arithmetische Ausdruck 'arithausdruck' berechnet. Das Ergebnis wird in eine ganze Zahl umgerechnet (Nachkommastellen werden abgeschnitten). Der so erhaltene Wert dient als Index in der Liste 'zeilenliste', welche als Liste von (existierenden) Zeilennummern, welche durch Kommata zu trennen sind, interpretiert wird. Dann wird an der hierdurch bestimmten Zeile fortgefahren (falls p2 = 'GOTO') bzw. das in jener Zeile beginnende Unterprogramm aufgerufen (falls p2 = 'GOSUB'). Im letzteren Fall wird nach Beendigung des Unterprogramms mit der der 'ON'-Anweisung folgenden Anweisung fortgefahren. Das Programm wird auch dann mit dieser Anweisung fortgesetzt, wenn weniger Zeilennummern vorhanden sind, als der berechnete Wert verlangt.

Defaultwerte: -

Wertebereich: arithausdruck: 1 bis 255
 zeilenliste: Liste von zeilennummern

Abkürzung: -

Beispiel:

```
10 input "Modus";a
20 on a goto 50,90,120
30 print "a="; a; " ist zu gross!"
40 end
50 open 1,8,2,"0:testdat,s,w"
60 rem *** falls a=1, dann wird 'testdat' zum
70 rem *** Schreiben eröffnet.
80 goto 200
90 open 1,8,2,"0:testdat,s,r"
100 rem *** falls a=2, dann wird 'testdat' zum
110 rem *** Lesen eröffnet.
120 scratch "testdat",d0
130 rem *** Bei a=3 wird 'testdat' gelöscht.
```

3.2.18 POKE

Format: POKE p1 p2
p1: adresse
p2: wert

Modus: Direkt / Programm

Zweck: Eintragen von Werten an bestimmte Speicheradressen

Voraussetzung:-

Beschreibung: Dieser Befehl ermöglicht es, errechnete Werte an beliebigen Speicherstellen einzutragen (falls diese Speicherstelle kein schreibgeschützter Bereich ist). Das Speichersegment, das zu bearbeiten ist, muß vorher durch das 'BANK'-Kommando ausgewählt werden. Der errechnete Wert 'wert' wird dann in das Byte mit der Speicheradresse 'adresse' eingetragen.

Defaultwerte: -

Wertebereich: adresse: 0 bis 65535, ganzzahlig
wert: 0 bis 255, ganzzahlig

Abkürzung: p0

Beispiel: 10 bs=13*4096
20 for i=0 to 255
30 poke bs+i,i
40 next
50 rem *** Alle 256 Zeichen des Bildschirmcodes
60 rem *** werden auf den Bildschirm ausgegeben
70 rem *** (Bildschirm-Anfangsadresse = 13*4096)

3.2.19 PRINT

Format: PRINT (p)
 p: wert

 PRINT USING p1 (p2)
 p1: format;
 p2: wert

Modus: Direkt / Programm

Zweck: Ausgabe von Daten

Voraussetzung:-

Beschreibung: Es sind zwei Fälle zu unterscheiden: format-
freies und formatiertes Ausgeben. Zuerst er-
folgt die Beschreibung bei der formatfreien
Ausgabe:

Auszugebende Werte werden immer linksbündig
ausgegeben. Der Abstand, der bei der Ausgabe
zwischen zwei Werten steht, ist von dem Tren-
nungszeichen abhängig. Wird ein Strichpunkt
als Trennzeichen verwendet, so werden die Daten
'dicht' hintereinander geschrieben; d.h., zwei
Zahlen werden durch nur ein Leerzeichen (plus
die Stelle für das Vorzeichen der zweiten Zahl)
voneinander getrennt. Strings werden ohne Zwischen-
raum ausgegeben. Dient ein Komma als Trennzeichen,
so ist der Abstand anders: Werte werden immer ab
eine Position gedruckt, die durch zehn ohne Rest
teilbar ist: also in den Positionen 0, 10, 20, ...
Dabei werden zwischen zwei Zahlen jedoch mindestens
zwei Leerzeichen eingefügt; belegt eine Zahl also
z.B. die Druckpositionen Null bis Acht (einschließ-
lich), so beginnt die nächste Zahl nicht in Posi-
tion 10, sondern in Position 20. Strings werden
durch mindestens ein Leerzeichen voneinander ge-
trennt.

Wird eine 'PRINT'-Anweisung mit einem Komma oder
einem Strichpunkt abgeschlossen, so werden die Werte,
die bei der nächsten 'PRINT'-Anweisung ausgegeben

werden, in die gleiche Zeile, hinter die bereits vorhandenen, geschrieben. Für den Abstand gilt das oben gesagte.

Grundsätzlich ist zu sagen: Zahlen werden immer im Stringformat ausgegeben. Es ist also nicht möglich, mit der 'PRINT'-Anweisung Zahlen in der internen Darstellung auszugeben.

Die formatierte Ausgabe erfordert eine etwas ausführlichere Erklärung:

Zunächst einmal die grundsätzlichen Abweichungen gegenüber der normalen 'PRINT'-Anweisung:

- Die Funktionen 'TAB' und 'SPC' sind nicht zulässig.
- Werte können nicht durch ';' voneinander getrennt werden.
- Das Komma dient als Trennzeichen zwischen den Werten. Die bei 'PRINT' dadurch erzwungene Ausrichtung auf bestimmte Druckpositionen entfällt.

Das Format, in welchem die Werte auszudrucken sind, wird durch die Formatierungsvorschrift 'format' festgelegt. 'format' besteht aus mindestens einem 'Formatfeld'. Formatfelder werden durch Zeichen getrennt, welche keine Formatierungsfunktion haben. Diese Zeichen werden so ausgedruckt, wie sie angegeben werden.

Zur Formatierung dienen im einzelnen folgende Zeichen:

1. # (Doppelkreuz):

Dieses Zeichen reserviert eine Druckposition für genau ein Zeichen oder eine Ziffer. Jedes Formatfeld muß mindestens ein Doppelkreuz beinhalten.

2. + (Plus) oder - (Minus):

Diese Zeichen können entweder am Anfang oder am Ende eines Formatfeldes stehen. An der hierdurch festgelegten Position wird das Vorzeichen einer Zahl ausgedruckt. Man sieht: diese Zeichen sind nur bei Zahlen zulässig, nicht bei Strings. Die Angabe von 'Plus' oder 'Minus' unterscheidet sich folgendermaßen: bei 'Plus' wird das Vorzeichen in jedem Fall ausgegeben; bei 'Minus' wird bei positiven Zahlen eine Leerstelle, bei negativen Zahlen das Minuszeichen ausgegeben.

3. . (Dezimalpunkt):

Dieses Zeichen definiert die Position des Dezimalpunktes bei Zahlen.

4. , (Komma):

Definiert die Position eines Kommas in Zahlen; dient lediglich der besseren Lesbarkeit der Zahlen.

5. \$ (Dollarzeichen):

Wird dieses Zeichen in einem Zahlenfeld angegeben, so erscheint es bei der Ausgabe vor der ersten gültigen Ziffer.

6. (vier Pfeile nach oben):

Werden diese Zeichen angegeben, so wird die auszugebende Zahl in Exponentialdarstellung ausgegeben.

7. = (Gleichheitszeichen):

Strings werden normalerweise linksbündig ausgegeben. Durch Angabe eines Gleichheitszeichens im zugehörigen Formatfeld wird der String in die Mitte des Feldes zentriert.

8. (Größerzeichen)

Strings werden rechtsbündig ausgegeben.

Sind für die Ausgabe einer Zahl oder eines Strings mehr Zeichen nötig, als im Formatfeld dafür vorgesehen sind, so werden bei Zahlen statt der Zahl '*' (Sternchen) ausgegeben, Strings werden abgeschnitten.

Defaultwerte: -

Wertebereich:	format:	Formatstring entsprechend obigen Angaben
	wert:	arithmetischer oder String-Ausdruck

Abkürzung:	?
	usI

Beispiel:

```

print using "+###";1
+ 1
print using "####";-101.5
-101
print using "###,###,###.##"; 123456.789
123,456.79
print using "+#.#####";1
+1.0e+00
print using "#####";"cbm"
cbm
print using "####=#";"cbm"
cbm
print using "## #####";"cbm"
cbm
print using "###$-";-1
$1-
```

Zahlen werden gerundet!

3.2.20 PRINT#

Format: PRINT#p1 (p2)
p1: dateinummer
p2: wert

PRINT# p1,USING p2 p3
p1: dateinummer
p2: format
p3: wert

Modus: Direkt / Programm

Zweck: Ausgabe von Daten auf beliebiges Ausgabegerät

Voraussetzung: Ausgabegerät wurde vorher zum Schreiben
eröffnet

Beschreibung: Die Funktion ist die gleiche wie bei der einfachen
'PRINT (USING) '-Anweisung. Der einzige Unterschied
besteht darin, daß die Ausgabe auf ein beliebiges
Ausgabegerät erfolgt. Das Ausgabegerät wird durch
'dateinummer' spezifiziert. Zur genaueren Erläu-
terung s. im Kapitel 3.2.19, 'PRINT '-Anweisung.

Defaultwerte: -

Wertebereich: dateinummer: 0 bis 255, ganzzahlig
format: Formatstring entsprechend obigen Angaben
wert: arithmetischer oder String-Ausdruck

Abkürzung: pR
usI

Beispiel: s. Kap. 3.2.19

3.2.21 PUDEF

Format: PUDEF p
 p: string

Modus: Direkt / Programm

Zweck: Verändern von Zeichen für die 'PRINT USING'-
 Anweisung

Voraussetzung:-

Beschreibung: Mit diesem Kommando können Standardwerte, die durch die 'PRINT USING'-Anweisung erzeugt werden können, verändert werden. Diese Standardwerte sind das Leerzeichen, das Komma, der Dezimalpunkt und das Währungssymbol. Normalerweise sind das die Zeichen ' ', ',', '.' und '\$'. Die neuer Werte, die diese Funktionen übernehmen sollen, müssen in 'string' angegeben werden, und zwar in der oben aufgeführten Reihenfolge. Ist der angegebene String länger als vier Zeichen, so werden nur die ersten vier ausgewertet. Ist er kürzer, so werden nur die vorhandenen ausgewertet.

Die 'PUDEF'-Anweisung funktioniert nun so, daß in den darauf folgenden 'PRINT USING'-Anweisungen überall dort, wo eines der oben genannten Zeichen erzeugt würde, stattdessen das durch 'PUDEF' definierte Zeichen erzeugt wird.

Es ist zu bemerken, daß durch 'PUDEF' nur die Zeichen verändert werden, die durch 'PRINT USING' ERZEUGT werden. Die Zeichen in dem Formatstring der 'PRINT USING'-Anweisung werden davon NICHT berührt.

Defaultwerte: -

Wertebereich: string: String

Abkürzung: pU

Beispiel:

```
10 print using "###,###,###.##";-1234.567
20 pundef "*.,,"
30 rem *** das leerzeichen wird durch "*" ersetzt
40 rem *** das komma wird durch "." ersetzt
50 rem *** der dezimalpunkt wird durch "," ersetzt
60 print using "###,###,###.##";-1234.567
run
      -1,234.57
*****-1.234,57
```

3.2.22 READ

Format: READ (p)
 p: variable

Modus: Direkt / Programm

Zweck: Einlesen von Variablenwerten, deren Werte im
 Programm enthalten sind

Voraussetzung:-

Beschreibung: Mit dieser Anweisung werden Werte an Variable
 übergeben, die in einer 'DATA'-Anweisung im
 Programm enthalten sind.

Die Zuordnung von Variablen zu den Werten der
'DATA'-Zeile(n) geschieht folgendermaßen:

Der ersten Variablen, die in der ersten durch-
laufenen(!) 'READ'-Anweisung steht, wird der erste
Wert der ersten gültigen (!) 'DATA'-Zeilen
zugewiesen, der zweiten Variablen der zweite Wert
usw. Als erste gültige 'DATA'-Zeile ist die
erste vorhandene 'DATA'-Zeile bzw. bei Verwendung
der 'RESTORE'-Anweisung die dort angegebene 'DATA'-
Zeile zu verstehen.

Es ist darauf zu achten, daß Werte in den 'DATA'-
Anweisungen mit den zugehörigen Variablen verträglich
sind, daß also z.B. nicht einer REAL-Variablen
ein String zugeordnet wird.

Wird die 'READ'-Anweisung im Direktmodus gegeben, so
werden die DATA-Anweisungen des gerade geladenen
Programms verwendet.

Defaultwerte: -

Wertebereich: variable: gültiger BASIC-Variablenname

Abkürzung: rE

Beispiel:

```
10 data a,b,"c,d",1,100
20 read x$,y$,z$,v1,v2
30 print x$
40 print y$
50 print z$
60 print v1
70 print v2
run
a
b
c,d
1
100

ready.
```

3.2.23 REM

Format: REM p
 p: text

Modus: Direkt / Programm

Zweck: Kommentieren von Programmen

Voraussetzung:-

Beschreibung: Dieses Kommando ermöglicht das Kommentieren von Programmen. Als Kommentar ist dabei 'text' zu verwenden. Für den Programmablauf hat dieser Text keinerlei Bedeutung, er erhöht jedoch die Lesbarkeit des Programms. Der Text kann aus allen vorhandenen Zeichen bestehen.

Zu bemerken ist, daß Anweisungen, die hinter einer 'REM'-Anweisung (in der gleichen Zeile) stehen, nicht ausgeführt werden, da sie als Kommentar gelten.

Defaultwerte: text: leere Zeichenkette

Wertebereich: text: jede Zeichenkette aus vorhandenen Zeichen

Abkürzung: -

Beispiel: 10 a=20: rem *** Anfangswert ***
 20 rem nichtssagend: print a
 30 rem *** 'print a' wird niemals ausgeführt!

3.2.24 RESTORE

Format: RESTORE **[p]**
 p: zeilennummer

Modus: Direkt / Programm

Zweck: Verändern des Zeigers auf die aktuelle DATA-Zeile

Voraussetzung: angegebene Zeilennummer existiert

Beschreibung: Dieser Befehl setzt den DATA-Zeiger auf die angegebene 'zeilennummer'. Das bedeutet, daß bei der nächsten 'READ'-Anweisung die Daten von dieser DATA-Zeile beginnend eingelesen werden.

Fehlt die Angabe 'zeilennummer', so wird der DATA-Zeiger auf die erste vorhandene DATA-Zeile gesetzt.

Defaultwerte: zeilennummer: erste vorhandene DATA-Zeile

Wertebereich: zeilennummer: 0 bis 63999, ganzzahlig

Abkürzung: reS

Beispiel: 10 data 1,2,3
 20 data 101,102,103
 30 read a: print a
 40 z=20: restore z
 50 read a: print a
 60 rem *** In Zeile 30 erhält 'a' den Wert '1',
 70 rem *** in Zeile 50 den Wert '101'

3.2.25 RESUME

Format: RESUME [p]
 p: zeilennummer
 NEXT

Modus: Programm

Zweck: Programmfortsetzung nach Fehlerbehandlung

Voraussetzung:-

Beschreibung: Dieser Befehl ist nur wirksam im Zusammenhang mit dem 'TRAP'-Befehl. Tritt während des Programmlaufs ein Fehler auf, der durch die 'TRAP'-Anweisung abgefangen wird, so wirkt die Fehlerbehandlung zunächst wie eine 'GOTO'-Anweisung zur Fehlerbehandlung. Soll danach das Programm jedoch wieder an der Fehlerstelle (oder einer anderen Stelle) weitergeführt werden, so ist die 'RESUME'-Anweisung zu verwenden.

Der Parameter 'p' hat dabei folgende Bedeutung:

Fehlt dieser Parameter, so wird das Programm mit der Anweisung fortgesetzt, welche den Fehler hervorrief. In der Fehlerbehandlungsroutine muß daher dafür gesorgt werden, daß dieser Fehler behoben ist.

Wird der Parameter 'NEXT' angegeben, so wird das Programm bei der nächsten Anweisung hinter der fehlerverursachenden Anweisung fortgesetzt.

Wird als Parameter eine 'zeilennummer' angegeben, dann wird das Programm mit der ersten Anweisung in dieser Zeile weitergeführt.

Defaultwerte: -

Wertebereich: zeilennummer: 0 bis 63999, ganzzahlig

Abkürzung: resU

Beispiel:

```
10 rem ***** Programm 1 *****
20 trap 50: c=50
30 a=5: b=0: c=a/b: print c
40 end
50 b=1: resume
60 rem *** Die fehlerverursachende Anweisung
70 rem *** (c=a/b) wird wiederholt und 'c' (=5!)
80 rem *** ausgegeben.

110 rem ***** Programm 2 *****
120 trap 150: c=50
130 a=5: b=0: c=a/b: print c
140 end
150 b=1: resume next
160 rem *** Die Anweisung (c=a/b) wird übersprungen
170 rem *** und 'c' (=50!) wird ausgegeben.
```

3.2.26 RETURN -----

Format: RETURN

Modus: Programm

Zweck: Rücksprung aus Unterprogrammen

Voraussetzung: Programm befindet sich in einem Unterprogramm

Beschreibung: Mit dieser Anweisung wird ein aufgerufenes Unterprogramm verlassen. Sie ist die letzte im Unterprogramm ausgeführte Anweisung. Das Programm wird hinter der aufrufenden 'GOSUB'-Anweisung fortgesetzt.

Defaultwerte: -

Wertebereich: -

Abkürzung: reT

Beispiel:

```
10 gosub 100
20 for i= aw to ew step sw
30 gosub 200
40 next
50 end
100 input "Anfangswert ";aw
110 input "Schrittweite";sw
120 input "Endwert      ";ew
130 return
200 print "Sinus(";i;"=";sin(i):return
```

3.2.27 STOP

Format: STOP

Modus: Programm

Zweck: Beenden eines Programms

Voraussetzung:-

Beschreibung: Die Anwendung dieses Kommandos bewirkt das Beenden des laufenden Programms. Der Rechner kehrt dadurch in seinen Grundzustand zurück. Wird das Kommando 'STOP' anstelle von 'END' verwendet, so gibt der Rechner die Meldung 'break in (zeilennummer)' aus. Ansonsten sind die Wirkungen dieser beiden Kommandos gleich.

Defaultwerte: -

Wertebereich: -

Abkürzung: sT

Beispiel:

```
10 for i=1 to 10
20 input x
30 if x=0 then dispose for: stop
40 print sin (1/x)
50 next
60 rem *** Es werden 10 Werte eingelesen und
70 rem *** der Sinus vom Kehrwert der eingelesenen
80 rem *** Zahl gedruckt. Falls der eingelesene
90 rem *** Wert jedoch Null ist, so wird das
100 rem *** Programm jedoch (vorzeitig) verlassen.
```

3.2.28 SYS

```
Format:      SYS p1 [p2]
             p1:  adresse
             p2:  parameter
```

Modus: Direkt / Programm

Zweck: Aufruf eines Assembler-Unterprogramms

Voraussetzung:-

Beschreibung: Dieser Befehl ruft ein Assembler-Unterprogramm auf, welches sich in dem Segment 15 (der 'Systembank') befindet. 'adresse' ist die Dezimaladresse, in welcher das Unterprogramm beginnt.

Die Bedeutung des Parameters 'parameter' hängt einzig und allein von dem Unterprogramm ab, das diesen Parameter abarbeiten muß.

Hinweis: Alle Assembler-Unterprogramme müssen mit dem Befehl 'RTS' abgeschlossen werden.

Defaultwerte: -

Wertebereich: adresse: 0 bis 65535, ganzzahlig
parameter: abhängig vom Unterprogramm

Abkürzung: sY

```

Beispiel: 10 for i=1024 to 1029: read a: poke i,a: next
          20 data 169,1,141,0,208,96
          30 rem *** 0400 a9 01      lda #$01
          40 rem *** 0402 8d 00 d0   sta $d000
          50 rem *** 0405 60          rts
          60 sys 1024
          70 rem *** Ein 'a' wird in die linke obere Ecke
          80 rem *** des Bildschirms geschrieben.

```

3.2.29 TRAP

Format: TRAP [p]
 p: zeilennummer

Modus: Programm

Zweck: Fehlerbehandlung durch Programm einleiten

Voraussetzung:-

Beschreibung: Diese Anweisung ermöglicht es, Fehler, die während eines Programmlaufs auftreten, selbst zu behandeln. Ohne Verwendung der 'TRAP'-Anweisung gibt der Rechner eine dem Fehler entsprechende Meldung aus und beendet das Programm. Wurde vor dem Auftreten des Fehlers im Programm eine 'TRAP'-Anweisung mit der Angabe einer Zeile ('zeilennummer') gegeben, so verzweigt das Programm im Fehlerfall zu dieser Zeile. Dort kann der Fehler dann analysiert (durch die Systemvariablen EL, ER und ERR\$) und behoben werden. Das Programm kann anschließend mit dem 'RESUME'-Befehl (s. dort) fortgesetzt werden.

Wird die 'TRAP'-Anweisung ohne Zeilennummer gegeben, dann verhält sich der Rechner wie im Normalfall: Beim Auftreten eines Fehlers wird das Programm abgebrochen.

Defaultwerte: -

Wertebereich: zeilennummer: 0 bis 63999, ganzzahlig

Abkürzung: tR

Beispiel:

```
10 trap 100: c=1
20 a=5: b=0
30 c=a/b
40 print c
50 trap      : c=1
60 goto 20
100 b=1: resume
110 rem *** Das Programm durchläuft folgende Zeilen:
120 rem *** 10,20,30,100,30,40,50,60,20,30
130 rem *** Hier wird nun das Programm mit einer
140 rem *** Fehlermeldung abgebrochen, da in Zeile 50
150 rem *** die Fehlerbehandlung per Programm
160 rem *** rückgängig gemacht wurde.
```


3.2.30 WAIT

Format: WAIT p1 p2 [p3]
 p1: adresse
 p2: maske1
 p3: maske2

Modus: Direkt / Programm

Zweck: Warten auf einen bestimmten Speicherinhalt

Voraussetzung:-

Beschreibung: Dieser Befehl prüft ständig eine Speicherstelle auf deren Inhalt. Die Anwendung dieses Befehls erfordert eine genaue Kenntnis der Funktionsweise des Rechners. Sinnvollerweise können nur solche Speicherstellen abgefragt werden, von denen sichergestellt ist, daß sie auch den erwarteten Wert erhalten. Das ist i. A. nur bei manchen Werten aus der Zero-Page der Systembank oder bei E/A-Bausteinen gewährleistet.

Die Adresse der zu überprüfenden Speicherstelle wird im Parameter 'adresse' angegeben. Der Rechner verhält sich bei diesem Befehl folgendermaßen:

Die Speicherstelle wird ausgelesen. Der Wert dieser Speicherstelle wird bitweise mit dem Wert von 'maske2' mit der logischen Funktion 'exklusiv-oder' verknüpft. (s. Kap. 'Datenmanipulationen') Der so ermittelte Wert wird - ebenfalls bitweise - mit 'maske1' durch die logische 'und'-Verknüpfung zu einem neuen Wert vereint. Ist dieser neue Wert ungleich Null, so wird mit der nächsten Anweisung hinter der 'WAIT'-Anweisung fortgefahren. Ist dies nicht der Fall, wird die Speicherstelle erneut überprüft. Dies geschieht solange, bis der erwünschte Wert erreicht ist.

Vereinfachend läßt sich folgendes sagen:

'maske2' dient zur Definition, ob die zu testenden Bits auf logisch Null oder Eins zu überprüfen sind. Ein gesetztes Bit in 'maske2' prüft das entsprechende Bit auf logisch Null, ein gelöscht Bit demnach auf logisch Eins. 'maske1' dient zum Ausfiltern der

Bits; ein gesetztes Bit in 'maske1' bewirkt die Prüfung des entsprechenden Bits in der Speicherstelle, ein gelöscht Bit in 'maske1' blockiert dieses Bit.

Defaultwerte: maske2: 0

Wertebereich: adresse: 0 bis 65535, ganzzahlig
maske1: 0 bis 255, ganzzahlig
maske2: 0 bis 255, ganzzahlig

Abkürzung: wA

Beispiel: 10 wait 205,255,255
20 rem *** Wartet, bis eine beliebige Taste
30 rem *** gedrückt wird.

3.3 Funktionen

Inhalt

3.3.1	Addition	118
3.3.2	Subtraktion	119
3.3.3	Multiplikation	120
3.3.4	Division	121
3.3.5	Potenzierung	122
3.3.6	Kleiner	123
3.3.7	Gleich	125
3.3.8	Größer	126
3.3.9	Kleiner gleich	128
3.3.10	Ungleich	130
3.3.11	Größer gleich	131
3.3.12	abs	133
3.3.13	and	134
3.3.14	asc	135
3.3.15	atn	136
3.3.16	chr\$	137
3.3.17	cos	138
3.3.18	err\$	139
3.3.19	exp	140
3.3.20	fre	141
3.3.21	instr	142
3.3.22	int	143
3.3.23	left\$	144
3.3.24	len	145
3.3.25	log	146
3.3.26	mid\$	147
3.3.27	not	149
3.3.28	or	150
3.3.29	peek	151
3.3.30	pos	152
3.3.31	right\$	153
3.3.32	rnd	154
3.3.33	sgn	156
3.3.34	sin	157
3.3.35	spc(.....	158
3.3.36	sqr	159
3.3.37	str\$	160
3.3.38	tab(.....	161
3.3.39	tan	162
3.3.40	usr	163
3.3.41	val	164

3.3.1 Addition

```
Format:      p1 + p2
             p1: operand1
             p2: operand2
```

Modus: Direkt / Programm

Zweck: Addition zweier Werte oder Stringverknüpfung

Voraussetzung: p_1 hat gleichen Typ wie p_2

Beschreibung: Sind 'p1' und 'p2' arithmetische Ausdrücke, so werden beide Werte addiert. Handelt es sich dabei jedoch um Strings, so werden sie verkettet, d.h., es wird ein neuer String gebildet, der sich aus den beiden Teilstrings zusammensetzt.

Defaultwerte: -

```
Wertebereich: operand1:  Ausdruck
               operand2:  Ausdruck
```

Abkürzung: -

```
Beispiel:      10 print 12+13
                20 x1$="abc"
                30 x2$="xyz"
                40 xs$=x1$+x2$
                50 print xs$
                run
                25
                abcx123
                ready.
```

3.3.2 Subtraktion

```
Format:      p1 - p2
             p1: operand1
             p2: operand2
```

Modus: Direkt / Programm

Zweck: Subtraktion zweier Werte

Voraussetzung:-

Beschreibung: 'p2' wird von 'p1' subtrahiert.

Defaultwerte: -

```
Wertebereich: operand1: arithmetischer Ausdruck
               operand2: arithmetischer Ausdruck
```

Abkürzung: -

```
Beispiel:      print 12-13
               -1
               ready.
```

3.3.3 Multiplikation

Format: p1 * p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: Multiplikation zweier Werte

Voraussetzung:-

Beschreibung: 'p1' wird mit 'p2' multipliziert.

Defaultwerte: -

Wertebereich: operand1: arithmetischer Ausdruck
 operand2: arithmetischer Ausdruck

Abkürzung: -

Beispiel: print 12*13

3.3.4 Division

Format: p1 / p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: Division zweier Werte

Voraussetzung:-

Beschreibung: 'p1' wird durch 'p2' dividiert.

Defaultwerte: -

Wertebereich: operand1: arithmetischer Ausdruck
 operand2: arithmetischer Ausdruck

Abkürzung: -

Beispiel: print 12/13
 .923076923

 ready.

3.3.5 Potenzierung

Format: $p1 \uparrow p2$
 $p1: \text{operand1}$
 $p2: \text{operand2}$

Modus: Direkt / Programm

Zweck: Potenzieren zweier Werte

Voraussetzung:-

Beschreibung: 'operand1' wird in die durch 'operand2' angegebene Potenz erhoben.

Defaultwerte: -

Wertebereich: operand1: arithmetischer Ausdruck
 operand2: arithmetischer Ausdruck

Abkürzung: -

Beispiel: $\text{print } 12 \uparrow 13$
 $1.06993206e+14$
 ready.

3.3.6 Kleiner

Format: $p1 < p2$
 $p1$: operand1
 $p2$: operand2

Modus: Direkt / Programm

Zweck: Vergleich zweier Werte auf kleiner

Voraussetzung: $p1$ und $p2$ sind vom selben Typ

Beschreibung: ' $p1$ ' wird mit ' $p2$ ' verglichen. Handelt es sich um arithmetische Ausdrücke, so wird der Wert -1 (logisch wahr) zurückgeliefert, wenn $p1$ kleiner als $p2$ ist, ansonsten der Wert 0 (logisch falsch).

Handelt es sich bei ' $p1$ ' und ' $p2$ ' um Strings, dann wird der Vergleich folgendermaßen vorgenommen:

Die ersten Zeichen beider Strings werden verglichen. Sind sie verschieden, so gilt der String als kleiner, dessen ASCII-Wert des ersten Zeichens kleiner ist.

Sind die beiden verglichenen Zeichen identisch, so werden die jeweils nächsten Zeichen verglichen.

Dies geschieht solange, bis zwei verschiedene Zeichen gefunden werden. Wird dabei das Ende eines Strings erreicht, gilt der als kleiner, der kürzer ist.

Defaultwerte: -

Wertebereich: operand1: Ausdruck
 operand2: Ausdruck

Abkürzung: -

Beispiel: 10 if 12<13 then print"kleiner in 10"
 20 if "abc"<"xyz" then print "kleiner in 20"
 30 if "xyz"<"xy" then print "kleiner in 30"
 run
 kleiner in 10
 kleiner in 20

 ready.

3.3.7 Gleich

Format: p1 = p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: Vergleich zweier Werte auf Gleichheit

Voraussetzung: p1 und p2 sind vom selben Typ

Beschreibung: 'p1' wird mit 'p2' verglichen. Handelt es sich um arithmetische Ausdrücke, so wird der Wert -1 (logisch wahr) zurückgeliefert, wenn p1 gleich p2 ist, ansonsten der Wert 0 (logisch falsch).

Handelt es sich bei 'p1' und 'p2' um Strings, dann wird der Vergleich folgendermaßen vorgenommen:

Haben beide Strings die gleich Länge und stimmen sie Zeichen für Zeichen überein, so sind sie gleich, ansonsten nicht.

Defaultwerte: -

Wertebereich: operand1: Ausdruck
 operand2: Ausdruck

Abkürzung: -

Beispiel: 10 if 12=13 then print"gleich in 10"
 20 if "abc"="abc" then print "gleich in 20"
 30 if "xyz"="xy" then print "gleich in 30"
 run
 gleich in 20
 ready.

3.3.8 Größer

Format: p1 > p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: Vergleich zweier Werte auf größer

Voraussetzung: p1 und p2 sind vom selben Typ

Beschreibung: 'p1' wird mit 'p2' verglichen. Handelt es sich um arithmetische Ausdrücke, so wird der Wert -1 (logisch wahr) zurückgeliefert, wenn p1 größer als p2 ist, ansonsten der Wert 0 (logisch falsch).

Handelt es sich bei 'p1' und 'p2' um Strings, dann wird der Vergleich folgendermaßen vorgenommen:

Die ersten Zeichen beider Strings werden verglichen. Sind sie verschieden, so gilt der String als größer, dessen ASCII-Wert des ersten Zeichens größer ist.

Sind die beiden verglichenen Zeichen identisch, so werden die jeweils nächsten Zeichen verglichen.

Dies geschieht solange, bis zwei verschiedene Zeichen gefunden werden. Wird dabei das Ende eines Strings erreicht, gilt derjenige als größer, der länger ist.

Defaultwerte: -

Wertebereich: operand1: Ausdruck
 operand2: Ausdruck

Abkürzung: -

Beispiel: 10 if 12>13 then print"groesser in 10"
 20 if "abc">"xyz" then print "groesser in 20"
 30 if "xyz">"xy" then print "groesser in 30"
 run
 groesser in 30

 ready.

3.3.9 Kleiner gleich

Format: $p1 \leq p2$
 $p1 = \leq p2$
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: Vergleich zweier Werte auf kleiner oder gleich

Voraussetzung: p1 und p2 sind vom selben Typ

Beschreibung: 'p1' wird mit 'p2' verglichen. Handelt es sich um arithmetische Ausdrücke, so wird der Wert -1 (logisch wahr) zurückgeliefert, wenn p1 kleiner oder gleich p2 ist, ansonsten der Wert Null (logisch falsch).

Handelt es sich bei 'p1' und 'p2' um Strings, dann wird der Vergleich folgendermaßen vorgenommen:

Die ersten Zeichen beider Strings werden verglichen. Sind sie verschieden, so gilt der String als kleiner, dessen ASCII-Wert des ersten Zeichens kleiner ist.

Sind die beiden verglichenen Zeichen identisch, so werden die jeweils nächsten Zeichen verglichen.

Dies geschieht solange, bis zwei verschiedene Zeichen gefunden werden. Wird dabei das Ende eines Strings erreicht, gilt derjenige als kleiner, der kürzer ist.

Defaultwerte: -

Wertebereich: operand1: Ausdruck
 operand2: Ausdruck

Abkürzung: -

Beispiel: 10 if 12<=13 then print"kleiner gleich in 10"
 20 if "abc"<="xyz" then print "kleiner gleich in 20"
 30 if "xyz"<="xyz" then print "kleiner gleich in 30"
 run
 kleiner gleich in 10
 kleiner gleich in 20
 kleiner gleich in 30

 ready.

3.3.10 Ungleich

Format: p1 <> p2
 p1 >< p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: Vergleich zweier Werte auf Ungleich

Voraussetzung: p1 und p2 sind vom selben Typ

Beschreibung: 'p1' wird mit 'p2' verglichen. Handelt es sich um arithmetische Ausdrücke, so wird der Wert -1 (logisch wahr) zurückgeliefert, wenn p1 ungleich p2 ist, ansonsten der Wert 0 (logisch falsch).

Handelt es sich bei 'p1' und 'p2' um Strings, dann wird der Vergleich folgendermaßen vorgenommen:

Haben beide Strings die gleiche Länge und stimmen sie Zeichen für Zeichen überein, so sind sie gleich, ansonsten nicht.

Defaultwerte: -

Wertebereich: operand1: Ausdruck
 operand2: Ausdruck

Abkürzung: -

Beispiel: 10 if 12<>13 then print"ungleich in 10"
 20 if "abc"<>"abc" then print "ungleich in 20"
 30 if "xyz"<>"xy" then print "ungleich in 30"
 run
 ungleich in 10
 ungleich in 30
 ready.

3.3.11 Größer gleich

Format: p1 > = p2
 p1 = > p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: Vergleich zweier Werte auf größer oder gleich

Voraussetzung: p1 und p2 sind vom selben Typ

Beschreibung: 'p1' wird mit 'p2' verglichen. Handelt es sich um arithmetische Ausdrücke, so wird der Wert -1 (logisch wahr) zurückgeliefert, wenn p1 größer oder gleich p2 ist, ansonsten der Wert Null (logisch falsch).

Handelt es sich bei 'p1' und 'p2' um Strings, dann wird der Vergleich folgendermaßen vorgenommen:

Die ersten Zeichen beider Strings werden verglichen. Sind sie verschieden, so gilt der String als größer, dessen ASCII-Wert des ersten Zeichens größer ist.

Sind die beiden verglichenen Zeichen identisch, so werden die jeweils nächsten Zeichen verglichen.

Dies geschieht solange, bis zwei verschiedene Zeichen gefunden werden. Wird dabei das Ende eines Strings erreicht, gilt derjenige als größer, der länger ist.

Defaultwerte: -

Wertebereich: operand1: Ausdruck
 operand2: Ausdruck

Abkürzung: -

Beispiel: 10 if 12>=13 then print"groesser gleich in 10"
 20 if "abc">="abc" then print "groesser gleich in 20"
 30 if "xyz">="xy" then print "groesser gleich in 30"
 run
 groesser gleich in 20
 groesser gleich in 30

 ready.

3.3.12 ABS

Format: ABS p
 p: (operand)

Modus: Direkt / Programm

Zweck: Bilden des Absolutwertes

Voraussetzung:-

Beschreibung: Von 'operand' wird der Absolutwert gebildet.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: aB

Beispiel: print abs(15.3): print abs (-12.2)
 15.3
 12.2

 ready.

3.3.13 AND

Format: p1 AND p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: logische 'und'-Verknüpfung

Voraussetzung:-

Beschreibung: 'operand1' wird bitweise mit 'operand2' durch
 die logische 'und'-Funktion verknüpft.
 (s. Datenmanipulationen)

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: aN

Beispiel: print 1 and 2, 5 and 4, 3 and 5
 0 4 1
 ready.
 print (2 3) and (2 3)
 0 / bedeutet logisch falsch
 ready.

3.3.14 ASC -----

Format: ASC p
 p: (operand)

Modus: Direkt / Programm

Zweck: Umwandeln eines Zeichens in eine Zahl

Voraussetzung:-

Beschreibung: Der ASCII-Wert des ersten Zeichens im String
 'operand' wird zurückgeliefert. Das Ergebnis
 ist eine ganze Zahl zwischen 0 und 255.

Defaultwerte: -

Wertebereich: operand: nicht-leerer String

Abkürzung: aS

Beispiel: print asc("a"): print asc ("abcdefghijkl")
 65
 65
 ready.

3.3.15 Arcustangens

Format: ATN p
 p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen des Arcustangens

Voraussetzung:-

Beschreibung: Der Wert des Arcustangens von 'operand' wird berechnet. Das Ergebnis ist eine reelle Zahl, welche den Winkel in Radian darstellt.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: aT

Beispiel: 10 pi=3.141592654
 20 print 180/pi*atn(1); "Grad"
 run
 45 Grad

 ready.

3.3.16 CHR\$ -----

Format: CHR\$ p
 p: (operand)

Modus: Direkt / Programm

Zweck: Umwandeln einer Zahl in ein Zeichen

Voraussetzung:-

Beschreibung: Der Wert von 'operand' wird als ASCII-Wert
eines Zeichens interpretiert und dieses
Zeichen zurückgeliefert.

Defaultwerte: -

Wertebereich: operand: 0 bis 255, ganzzahlig

Abkürzung: cH

Beispiel: 10 x\$="abc"+chr\$(13)+"xyz"+chr\$(66)
 20 print x\$
 run
 abc
 xyzb

 ready.

3.3.17 Cosinus

Format: COS p
p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen des Cosinus

Voraussetzung:-

Beschreibung: Der Wert des Cosinus von 'operand' wird berechnet. Das Argument wird als Winkelangabe in Radian interpretiert.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: -

Beispiel: 10 pi=3.141592654
20 print cos(60*pi/180)
run
.5
ready.

3.3.18 ERR\$

Format: ERR\$ p
 p: (operand)

Modus: Direkt / Programm

Zweck: Erstellen der Fehlermeldung im Klartext

Voraussetzung:-

Beschreibung: Das Ergebnis dieser Funktion ist die Fehlermeldung im Klartext, deren Fehlernummer in 'operand' übergeben wird. Sie wird meist im Zusammenhang mit der Systemvariablen 'ER' verwendet (s. Systemvariablen).

Defaultwerte: -

Wertebereich: operand: 0 bis 42, ganzzahlig

Abkürzung: eR

Beispiel: 10 trap40
 20 x= 1/0
 30 end
 40 print "Fehler:"; er; "- "; err\$(er); " in"; el
 run
 Fehler: 30 - division by zero in 20
 ready.

3.3.19 Exponentialfunktion

Format: EXP p
 p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen von e hoch x (e = Eulersche Zahl)

Voraussetzung:-

Beschreibung: Diese Funktion berechnet e hoch 'operand', wobei
 ' e ' die Eulersche Zahl ist.

Defaultwerte: -

Wertebereich: operand: reeller Wert = 88.0296919

Abkürzung: eX

Beispiel: print exp(1),exp(2)
 2.71828183 7.3890561
 ready.

3.3.20 FRE

Format: FRE p
 p: (segment)

Modus: Direkt / Programm

Zweck: Berechnen der noch vorhandenen freien Speicher-
 kapazität

Voraussetzung:-

Beschreibung: Es wird die Anzahl der noch freien Byte für
 das Segment angegeben, dessen Nummer in 'segment'
 angegeben wird.

Defaultwerte: -

Wertebereich: segment: 0 bis 15, ganzzahlig

Abkürzung: fr

Beispiel: 10 fr=0
 20 for i = 0 to 15
 30 fr=fr+fre(i)
 40 print fr;" Bytes free"
 50 end
 run
 128187 Bytes free

 ready.

3.3.21 INSTR

Format: INSTR p1 p2
p1: (string1
p2: string2)

Modus: Direkt / Programm

Zweck: Überprüfen auf Teilstring

Voraussetzung:-

Beschreibung: Es wird untersucht, ob die Zeichenfolge 'string2' in 'string1' enthalten ist. Ist dies der Fall, so wird die Anfangsposition von 'string2' in 'string1' zurückgeliefert. Andernfalls ist das Ergebnis Null.

Defaultwerte: -

Wertebereich: string1: String
string2: String

Abkürzung: iN

Beispiel: 10 print instr("abcd","bc")
20 print instr("abcd","xy")
run
2
0
ready.

3.3.22 INT -----

Format: INT p
 p: (operand)

Modus: Direkt / Programm

Zweck: Umwandeln einer reellen Zahl in eine ganze Zahl

Voraussetzung:-

Beschreibung: Der (reelle) Parameter 'operand' wird in eine ganze Zahl gewandelt, indem der 'operand' zur nächsten ganzen Zahl, die kleiner oder gleich dem Operanden ist, abgerundet wird. Der 'operand' selbst bleibt unverändert, das Ergebnis wird als Funktionswert übergeben.

Defaultwerte: -

Wertebereich: operand: reeller Ausdruck zwischen -32768
 (einschließlich) bis +32768 (aus-
 (schließlich))

Abkürzung: -

Beispiel: print int(3.8): print int(-3.8): print int(100)
 3
 -4
 100

 ready.

3.3.23 LEFT\$

Format: LEFT\$ p1 p2
 p1: (string
 p2: länge)

Modus: Direkt / Programm

Zweck: Bilden eines Teilstrings

Voraussetzung:-

Beschreibung: Der Parameter 'string' ist ein String, der Parameter 'länge' ist eine Zahl. Das Ergebnis ist ein String, der aus 'string' gebildet wird. Dabei werden von 'string', von links beginnend, soviele Zeichen genommen, wie 'länge' angibt. Ist 'länge' größer als die Länge von 'string', so wird der ganze 'string' übertragen.

Defaultwerte: -

Wertebereich: operand1: String
 operand2: reeller Ausdruck größer oder
 gleich Null

Abkürzung: leF

Beispiel: 10 print "*" ; left\$ ("abcdefgh",3) ; "*"
 20 print "*" ; left\$ ("abcdefgh",20) ; "*"
 30 print "*" ; left\$ ("abcdefgh",4.9) ; "*"
 run
 abc
 abcdefgh
 abcd

 ready.

3.3.24 LEN -----

Format: LEN p
 p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen der Länge eines Strings

Voraussetzung:-

Beschreibung: 'operand' ist ein String. Das Ergebnis der Funktion ist die Anzahl der Zeichen, aus denen der String besteht.

Defaultwerte: -

Wertebereich: operand: String

Abkürzung: -

Beispiel: 10 input a\$
 20 print len(a\$)
 30 print len("abcd")
 run
 ? xxxyyyyzzz
 9
 4

 ready.

3.3.25 LOG

Format: LOG p
p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen des natürlichen Logarithmus einer Zahl

Voraussetzung:-

Beschreibung: Diese Funktion berechnet den Logarithmus zur Basis e (Eulersche Zahl, 2.71828183), d.h., den natürlichen Logarithmus, von 'operand'.

Defaultwerte: -

Wertebereich: operand: reeller Ausdruck größer als Null

Abkürzung: -

Beispiel: 10 print log(2.71828183)
20 print log(exp(3.5))
30 print exp(log(4.5))
run
1
3.5
4.5

ready.

3.3.26 MID\$

Format: MID\$ p1 p2
p1: (string
p2: position
p3: länge)

Modus: Direkt / Programm

Zweck: Bilden eines Teilstrings

Voraussetzung:-

Beschreibung: Der Parameter 'string' ist ein String, die Parameter 'position' und 'länge' sind Zahlen. Das Ergebnis ist ein String, der aus 'string' gebildet wird. Von 'position' und 'länge' werden jeweils die ganzzahligen Anteile verwendet. Der Ergebnisstring besteht aus so vielen Zeichen, wie 'länge' angibt. Er besteht aus den Zeichen, die in 'string' ab der Position stehen, welche in 'position' angegeben wird. Ist die Summe von 'position' und 'länge' größer als die Länge von 'string', so werden nur so viele Zeichen übertragen, wie die Länge von 'string' zuläßt.

Defaultwerte: -

Wertebereich: string: String
position: reeller Ausdruck größer oder gleich Null
länge: reeller Ausdruck größer oder gleich Null

Abkürzung: mI

Beispiel:

```
10 print "*" ; mid$ ("abcdefgh",3,2); "*"
20 print "*" ; mid$ ("abcdefgh",1,5); "*"
30 print "*" ; left$ ("abcdefgh",5); "*"
40 print "*" ; mid$ ("abcdefgh",2.9,3.1); "*"
run
*cd*
*abcde*
*abcde*
*bcd*

ready.
```

3.3.27 NOT

Format: NOT p
 p: operand

Modus: Direkt / Programm

Zweck: Bitweises Invertieren einer Zahl

Voraussetzung:-

Beschreibung: Der 'operand' wird bitweise invertiert. Ist der 'operand' ein logischer Ausdruck, so wird sein logischer Wert (wahr oder falsch) in sein Gegenteil verändert. Ist 'operand' ein arithmetischer Ausdruck, so ist das Ergebnis das gleiche wie bei der Operation "-('operand'+1)".

Defaultwerte: -

Wertebereich: operand: arithmetischer oder logischer Ausdruck

Abkürzung: n0

Beispiel: 10 if not (5 3) then print "so, so!"
 20 print not 13
 30 print not - 26
 40 print - not 26
 run
 so, so!
 -14
 25
 27

 ready.

3.3.28 OR

Format: p1 OR p2
 p1: operand1
 p2: operand2

Modus: Direkt / Programm

Zweck: logische 'oder'-Verknüpfung

Voraussetzung:-

Beschreibung: 'operand1' wird bitweise mit 'operand2' durch
 die logische 'oder'-Funktion verknüpft.
 (s. Datenmanipulationen)

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: -

Beispiel: print 1 or 2, 5 or 4, 3 or 5
 3 5 7

 ready.
 print (2 3) or (2 3)
 -1 / bedeutet logisch wahr

 ready.

3.3.29 PEEK

Format: PEEK p
 p: (adresse)

Modus: Direkt / Programm

Zweck: Auslesen von Speicherinhalten

Voraussetzung:-

Beschreibung: Das Ergebnis ist eine ganze Zahl zwischen Null und 255, welche den Inhalt der Speicherstelle angibt, die an der Position 'adresse' liegt. Das Speichersegment, auf das sich diese Funktion bezieht, kann mit der 'BANK'-Anweisung eingestellt werden (Standardwert ist 15, d.h. Systembank)

Defaultwerte: -

Wertebereich: adresse: 0 bis 65535, ganzzahlig

Abkürzung: pE

Beispiel: 10 bank 1
 20 for i=0 to 255
 30 print peek(i);
 40 next i
 50 rem *** Die ersten 255 Byte von Segment '1'
 60 rem *** (d.h. dem Programmtext-Segment) werden
 70 rem *** auf dem Bildschirm als Zahlen ausgegeben.

3.3.30 POS

Format: POS p
 p: (dummy)

Modus: Direkt / Programm

Zweck: Bereitstellen der nächsten Druckposition

Voraussetzung:-

Beschreibung: Das Ergebnis ist eine ganze Zahl, welche die Position auf dem Bildschirm angibt, ab welcher die nächste 'PRINT'-Anweisung ihre Werte ausgibt. Als Position ist hierbei die Spalte zu verstehen, nicht die Zeile auf dem Bildschirm. Das Argument 'dummy' muß zwar angegeben werden, hat jedoch keine Bedeutung.

Defaultwerte: -

Wertebereich: dummy: irgendetwas

Abkürzung: -

Beispiel: 10 print "abcdefg";
 20 if pos(x) 10 then print chr\$(13);
 30 rem *** Dann neue Zeile
 40 goto 10
 run
 abcdefgabcdefg
 abcdefgabcdefg
 abcdefgabcdefg
 abcdefgabcdefg
 .
 .
 .

3.3.31 RIGHT\$

Format: RIGHT\$ p1 p2
 p1: (string
 p2: länge)

Modus: Direkt / Programm

Zweck: Bilden eines Teilstrings

Voraussetzung:-

Beschreibung: Der Parameter 'string' ist ein String, der Parameter 'länge' ist eine Zahl. Das Ergebnis ist ein String, der aus 'string' gebildet wird. Dabei werden von 'string', von rechts beginnend, so viele Zeichen genommen, wie 'länge' angibt. Ist 'länge' größer als die Länge von 'string', so wird der ganze 'string' übertragen.

Defaultwerte: -

Wertebereich: operand1: String
 operand2: reeller Ausdruck größer oder
 gleich Null

Abkürzung: rI

Beispiel: 10 print "*" ; right\$("abcdefgh",3) ; "*"
 20 print "*" ; right\$("abcdefgh",20) ; "*"
 30 print "*" ; right\$("abcdefgh",4.9) ; "*"
 run
 fgh
 abcdefgh
 efgh
 ready.

3.3.32 RND

Format: RND p
 p: (operand)

Modus: Direkt / Programm

Zweck: Erzeugen von Zufallszahlen

Voraussetzung:-

Beschreibung: Diese Funktion liefert gleichverteilte Zufallszahlen zwischen Null und Eins. Durch 'operand' kann die Berechnung der Zufallszahlen beeinflusst werden:

1. 'operand' ist positiv:

Es wird, abhängig vom Anfangswert, immer die gleiche Reihe von Zufallszahlen nach einem bestimmten Algorithmus erzeugt. Der Anfangswert wird beim Einschalten des Rechners erstellt. Die Größe von 'operand' spielt dabei keine Rolle.

2. 'operand' ist negativ:

Es wird ein Anfangswert erstellt, der von der Größe von 'operand' abhängig ist. Dieser Anfangswert ist der erste zurückgelieferte.

3. 'operand' gleich Null:

Es wird ein Anfangswert erstellt, der aus verschiedenen internen Registern (Uhr) berechnet wird. Dieser Anfangswert ist der erste zurückgelieferte.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: rN

Beispiel:

```
10 input "Anfang, Ende"; anfang, ende
20 a=rnd(0)
30 for i=1 to 100
40 print (ende-anfang)*rnd(1)+anfang
50 rem *** Es wird eine Reihe von Zufallszahlen
60 rem *** erzeugt, deren erster Wert nur von
70 rem *** internen Registern des Rechners (d.h.,
80 rem *** von der Uhr) abhängt. Sie liegen in
90 rem *** dem Zahlenbereich zwischen 'anfang'
100 rem *** und 'ende'.
```

3.3.33 SGN

Format: SGN p
 p: (operand)

Modus: Direkt / Programm

Zweck: Bestimmen des Vorzeichens eines Wertes

Voraussetzung:-

Beschreibung: Als Ergebnis dieser Funktion wird der Wert 1 zurückgeliefert, wenn 'operand' größer als Null ist, der Wert -1, falls 'operand' kleiner als Null ist, bzw. der Wert Null, wenn 'operand' gleich Null ist.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: sG

Beispiel: 10 input x
 20 on sgn(x)+2 gosub 40,60,80
 30 goto 10
 40 print "x 0"
 50 return
 60 print "x 0"
 70 return
 80 print "x = 0"
 90 return
 100 rem *** Für jeden eingelesenen Wert wird
 110 rem *** ausgedruckt, ob er positiv, negativ
 120 rem *** oder Null war.

3.3.34 Sinus -----

Format: SIN p
 p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen des Sinus

Voraussetzung:-

Beschreibung: Der Wert des Sinus von 'operand' wird berechnet. Das Argument wird als Winkelangabe in Radian interpretiert.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: sI

Beispiel: 10 pi=3.141592654
 20 print sin(30*pi/180)
 run
 .5

 ready.

3.3.35 SPC

Format: SPC (p
 p: operand)

Modus: Direkt / Programm

Zweck: Positionieren des Cursors

Voraussetzung:-

Beschreibung: Diese Funktion ist nur wirksam in Verbindung mit einer 'PRINT'-Anweisung. Es werden dabei so viele Druckpositionen übersprungen (von der letzten Druckposition aus gesehen), wie 'operand' angibt.

Defaultwerte: -

Wertebereich: operand: 0 bis 255, ganzzahlig

Abkürzung: sP

Beispiel: 10 print "aa"; spc(5); "bb"; spc(2);
 20 print "ccc"
 run
 aa bb ccc

 ready.

3.3.36 SQR -----

Format: SQR p
 p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen der Quadratwurzel

Voraussetzung:-

Beschreibung: Diese Funktion berechnet die Quadratwurzel von
 'operand'.

Defaultwerte: -

Wertebereich: operand: positiver arithmetischer Ausdruck

Abkürzung: sQ

Beispiel: 10 for i=1 to 100
 20 print i,sqr(i)
 30 next
 40 rem *** Es wird eine Tabelle erstellt, welche
 50 rem *** die Quadratwurzeln der ersten 100
 60 rem *** Zahlen enthält.

3.3.37 STR\$

Format: STR\$ p
 p: (operand)

Modus: Direkt / Programm

Zweck: Umwandeln einer Zahl in einen String

Voraussetzung:-

Beschreibung: Diese Funktion wandelt den Zahlenwert, der durch
 'operand' dargestellt wird, in einen String um.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: str

Beispiel: 10 a\$=""
 20 for i=0 to 39
 30 test\$=right\$(str\$(i),1)
 40 rem *** 'test\$' enthält die Einerstelle von 'i'
 50 if test\$="0" then a\$=a\$+" ":else a\$=a\$+test\$
 60 next
 70 print a\$
 run
 123456789 123456789 123456789 123456789

 ready.

3.3.38 TAB -----

Format: TAB (p
 p: operand)

Modus: Direkt / Programm

Zweck: Positionieren des Cursors

Voraussetzung:-

Beschreibung: Diese Funktion ist nur wirksam in Verbindung mit einer 'PRINT'-Anweisung. Es werden dabei so viele Druckpositionen übersprungen (von der letzten Druckposition aus gesehen), wie 'operand' angibt.

Defaultwerte: -

Wertebereich: operand: 0 bis 255, ganzzahlig

Abkürzung: tA

Beispiel: 10 print "aa"; tab(5); "bb"; tab(2);
 20 print "ccc"
 run
 aa bb ccc
 ready.

3.3.39 Tangens

Format: TAN p
 p: (operand)

Modus: Direkt / Programm

Zweck: Berechnen des Tangens

Voraussetzung:-

Beschreibung: Der Wert des Tangens von 'operand' wird berechnet. Das Argument wird als Winkelangabe in Radian interpretiert.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: -

Beispiel: 10 pi=3.141592654
 20 print tan(45*pi/180)
 run
 1
 ready.

3.3.40 USR

Format: USR p
 p: (operand)

Modus: Direkt / Programm

Zweck: Aufrufen einer Benutzerfunktion

Voraussetzung:-

Beschreibung: Die 'USR'-Funktion ist eine Funktion, die der Benutzer selbst erstellen muß. Die Adresse, ab der die 'USR'-Funktion beginnt, muß in die Speicherstellen 3 und 4 der Systembank geschrieben werden. Der Wert von 'operand' wird vom Betriebssystem in den Fließkomma-Akkumulator des Rechners eingetragen, bevor die 'USR'-Funktion aufgerufen wird. Das Ergebnis, welches die 'USR'-Funktion erstellt, wird ebenfalls im Fließkomma-Akkumulator bei der Rückkehr zu BASIC erwartet.

Defaultwerte: -

Wertebereich: operand: arithmetischer Ausdruck

Abkürzung: uS

Beispiel: 10 poke 3,0: poke 4,4
 20 x=usr(20)
 30 print x
 40 rem *** In der Systembank steht ab Adresse 1024
 50 rem *** eine USR-Funktion. Der Wert '20' wird in
 60 rem *** den Fließkomma-Akkumulator eingetragen,
 70 rem *** die USR-Funktion berechnet irgendetwas
 80 rem *** daraus, und dieser Wert wird an die
 90 rem *** Variable 'x' übergeben.

3.3.41 VAL

Format: VAL p
p: (operand)

Modus: Direkt / Programm

Zweck: Umrechnen eines Strings in eine Zahl

Voraussetzung:-

Beschreibung: Die Funktion 'VAL' erwartet als 'operand' einen Zahlenstring, d.h. einen String, der eine gültige Zahlendarstellung repräsentiert. Dieser Zahlenstring wird in die entsprechende Zahl umgewandelt.

Defaultwerte: -

Wertebereich: operand: Zahlenstring

Abkürzung: vA

Beispiel: 10 input x\$
20 x = val(x\$+"0")
30 print "10 * "; x\$; " ="; x
run
? 123
10 * 123 = 1230
ready.

IEEE Connector

Pin	Bez.	Baustein	intern	Adresse	
1	D1	CIA 6526	PRA 0	dc00	56320
2	D2	CIA 6526	PRA 1	dc00	56320
3	D3	CIA 6526	PRA 2	dc00	56320
4	D4	CIA 6526	PRA 3	dc00	56320
5	EOI	TPI 6525	PRA 5	de00	56832
6	DAV	TPI 6525	PRA 4	de00	56832
7	NRFD	TPI 6525	PRA 7	de00	56832
8	NDAC	TPI 6525	PRA 6	de00	56832
9	IFC	TPI 6525	PRB 0	de01	56833
10	SRQ	TPI 6525	PRB 1	de01	56833
11	ATN	TPI 6525	PRA 3	de00	56832
12	SHIELD				
A	D5	CIA 6526	PRA 4	dc00	56320
B	D6	CIA 6526	PRA 5	dc00	56320
C	D7	CIA 6526	PRA 6	dc00	56320
D	D8	CIA 6526	PRA 7	dc00	56320
E	REN	TPI 6525	PRA 2	de00	56832
F	GND				
H	GND				
J	GND				
K	GND				
L	GND				
M	GND				
N	GND				

RS 232 C Connector

Pin	Bez.	Baustein	intern	Adresse
1	SHIELD			
2	TxD			
3	RxD			
4	RTS			
5	CTS			
6	DSR			
7	GND			
8	DCD			
9	N.C.			
10	N.C.			
11	+ 5 V DC			
12	- 12 V DC			
13	N.C.			
14	N.C.			
15	N.C.			
16	N.C.			
17	N.C.			
18	N.C.			
19	N.C.			
20	DTR			
21	N.C.			
22	N.C.			
23	N.C.			
24	RxC			
25	N.C.			

USER Connector

Pin	Bez.	Baustein	intern	Adresse	
1	GND				
2	PB2	TPI 6525	PRB 2	de01	56833
3	GND				
4	PB3	TPI 6525	PRB 3	de01	56833
5	NOT PC	CIA 6526	-PC		
	(Handshake PRB I/O, Output)				
6	NOT FL. Cass-Read		-FLAG		
	(Interrupt, Input)				
7	2D7	CIA 6526	PRB 7	dc01	56321
8	2D6	CIA 6526	PRB 6	dc01	56321
9	2D5	CIA 6526	PRB 5	dc01	56321
10	2D4	CIA 6526	PRB 4	dc01	56321
11	2D3	CIA 6526	PRB 3	dc01	56321
12	2D2	CIA 6526	PRB 2	dc01	56321
13	2D1	CIA 6526	PRB 1	dc01	56321
14	2D0	CIA 6526	PRB 0	dc01	56321
15	1D7	CIA 6526	PRA 7	dc00	56320
16	1D6	CIA 6526	PRA 6	dc00	56320
17	1D5	CIA 6526	PRA 5	dc00	56320
18	1D4	CIA 6526	PRA 4	dc00	56320
19	1D3	CIA 6526	PRA 3	dc00	56320
20	1D2	CIA 6526	PRA 2	dc00	56320
21	1D1	CIA 6526	PRA 1	dc00	56320
22	1D0	CIA 6526	PRA 0	dc00	56320
23	NOT CNT	CIA 6526	-CNT		
	(Takt v. Timer A			dc04/5	56324/5
24	+ 5 V DC				
25	NOT IRQ	TPI 6525	PRC 5	de02	56834
26	SP	CIA 6526	SP		
	(Serial Port I/O)				

Keyboard Connector

Pin	Bez.	Baustein	intern	Adresse	
1	PA0	TPI 6525	PRA 0	df00	57088
2	PA2	TPI 6525	PRA 2	df00	57088
3	PA4	TPI 6525	PRA 4	df00	57088
4	PA6	TPI 6525	PRA 6	df00	57088
5	PB0	TPI 6525	PRB 0	df01	57089
6	PB1	TPI 6525	PRB 1	df01	57089
7	PB2	TPI 6525	PRB 2	df01	57089
8	PB3	TPI 6525	PRB 3	df01	57089
9	PB4	TPI 6525	PRB 4	df01	57089
10	PB5	TPI 6525	PRB 5	df01	57089
11	PB6	TPI 6525	PRB 6	df01	57089
12	PB7	TPI 6525	PRB 7	df01	57089
13	PC5	TPI 6525	PRC 5	df02	57090
14	PA1	TPI 6525	PRA 1	df00	57088
15	PA3	TPI 6525	PRA 3	df00	57088
16	PA5	TPI 6525	PRA 5	df00	57088
17	PA7	TPI 6525	PRA 7	df00	57088
18	PC0	TPI 6525	PRC 0	df02	57090
19	PC1	TPI 6525	PRC 1	df02	57090
20	PC2	TPI 6525	PRC 2	df02	57090
21	PC3	TPI 6525	PRC 3	df02	57090
22	GND				
23	GND				
24	GND				
25	PC4	TPI 6525	PRC 4	df02	57090

Cartridge Connector

Pin	Bez.	Baustein	intern	Adresse
1	A0		Adresse	
2	A1		Adresse	
3	A2		Adresse	
4	A3		Adresse	
5	A4		Adresse	
6	A5		Adresse	
7	A6		Adresse	
8	A7		Adresse	
9	A8		Adresse	
10	A9		Adresse	
11	A10		Adresse	
12	A11		Adresse	
13	A12		Adresse	
14	+ 5 V	DC		
15	+ 5 V	DC		
A	BD0		Daten	
B	BD1		Daten	
C	BD2		Daten	
D	BD3		Daten	
E	BD4		Daten	
F	BD5		Daten	
H	BD6		Daten	
J	BD7		Daten	
K	GND			
L	GND			
M	S R/W		Modus Read/Write	
N	S02		Takt phi 2	
P	NOT CSBANK 1		Adresse	
R	NOT CSBANK 2		Adresse	
S	NOT CSBANK 3		Adresse	

Co-Processor Connector

Pin	Bez.	Baustein	intern	Adresse
1	EXTMA3			
2	DRAM00			
3	EXTMA2			
4	DRAM01			
5	EXTMA7			
6	DRAM02			
7	EXTMA6			
8	DRAM03			
9	EXTMA5			
10	DRAM04			
11	EXTMA4			
12	DRAM05			
13	EXTMA1			
14	DRAM06			
15	EXTMA0			
16	DRAM07			
17	GND			
18	GND			
19	GND			
20	GND			
21	GND			
22	NOT BUSY 1			
23	GND			
24	NOT P2REFREQ			
25	GND			
26	NOT P2REFGRNT			
27	GND			
28	BP0			
29	GND			
30	BP1			
31	GND			
32	BP2			
33	N.C.			
34	BP3			
35	NOT PROCRES			
36	NOT BUSY2			
37	EXTBUF R/W			
38	NOT ERAS			
39	DRAM R/W			
40	NOT ECAS			

Expansion Connector

Pin	Bez.	Baustein	intern	Adresse
1	+ 5 V DC			
2	+ 5 V DC			
3	+ 5 V DC			
4	+ 5 V DC			
5	GND			
6	GND			
7	GND			
8	GND			
9	GND			
10	GND			
11	NOT BRAS		DRAM: Row Access	
12	IRQ3	TPI 6525	PRC 3	de02 56834
13	- 12 V DC			
14	NOT EXTRES		Reset-Durchlaß	
15	+ 12 V DC			
16	NOT S0	CPU 6509	S0	
17	NOT RES		System Reset	
18	LPEN	CRTC 6845	Light Pen	
19	S R/W		System Read/Write	
20	NOT EXTBUFCS		Adresse \$0800-\$0fff	
21	TODCLK		Takt 50 Hz	
22	NOT DISKROMCS		Adresse \$1000-\$1fff	
23	BDOTCLK		Takt (18 MHz)	
24	N.C.			
25	S02		Takt phi 2	
26	NOT BCAS		DRAM: Column Access	
27	S01		Takt phi 1	
28	NOT CS1		Adresse: \$d900-\$d9ff	
29	BD3		Daten	
30	NOT EXTPRTCS		Adresse: \$db00-\$dbff	
31	BD4		Daten	
32	BD2		Daten	
33	BD5		Daten	
34	BD1		Daten	
35	BD7		Daten	
36	BD0		Daten	
37	BA13		Adresse	
38	BD7		Daten	
39	BA14		Adresse	
40	BA15		Adresse	
41	BA1		Adresse	
42	BA0		Adresse	
43	BA2		Adresse	
44	BA11		Adresse	
45	BA3		Adresse	
46	BA10		Adresse	
47	BA12		Adresse	

48	BA4			Adresse
49	BA9			Adresse
50	BA5			Adresse
51	BA8			Adresse
52	BA6			Adresse
53	BP0			Bank
54	BA7			Adresse
55	BP1			Bank
56	BP2			Bank
57	NOT	NMI	CPU 6509	Non-maskable Interrupt
58	BP3			Bank
59	RDY		CPU 6509	Ready
60	NOT	IRQ	CPU 6509	Interrupt Request

Audio Connector

Pin	Bez.	Baustein	intern	Adresse
1	Lautsprecher	(8 Ohm)		
2	N.C.			
3	Lautsprecher	(8 Ohm)		

Power Connector

Pin	Bez.	Baustein	intern	Adresse
1	50 Hz			
2	- 12 V DC			
3	+ 12 V DC			
4	GND			
5	GND			
6	+ 5 V DC			

Video Connector

Pin	Bez.	Baustein	intern	Adresse
1	VIDEO			
2	GND			
3	VERTICAL SYNC			
4	GND			
5	HORIZONTAL SYNC			
6	KEY			
7	GND			

5 Die RS 232 C - Schnittstelle

5.0 Vorbemerkungen

Die Rechner der Systemfamilie CBM 600 / 700 sind standardmäßig mit einer Schnittstelle ausgerüstet, welche allgemein unter dem Begriff RS 232 C oder V.24 bekannt ist. Diese beiden Bezeichnungen haben ihren Ursprung in der Nomenklatur der Institute, welche diese Schnittstelle unter diesen Bezeichnungen normten. Im amerikanischen Sprachraum ist dies die EIA (Electronic Industries Association) für RS 232 C, in Europa ist es das CCITT (Commitee Consultative Internationale de Telephone et Telegraphe) für V.24. In Deutschland sind die Empfehlungen dieser Schnittstellen in der DIN 66... genormt.

Diese Schnittstelle arbeitet asynchron und seriell. Dies bedeutet, daß alle Informationen in serieller Form, d.h. Bit für Bit, übertragen werden. Asynchron bedeutet hierbei, daß die Übertragungsleitung, wenn keine Information zu übermitteln ist, auf einem bestimmten logischen Pegel gehalten wird; im Gegensatz zur synchronen Datenübertragung, bei welcher ständig ein Füllbyte auf der Leitung gesendet wird.

Im seriellen Datenformat wird bei der Übertragung eines Byte zuerst ein Start-Bit gesendet, dann die Informations-Bits (LSB zuerst), eventuell ein Paritätsbit und ein oder zwei Stoppbits. Da die Datenleitung im Grundzustand (d.h., wenn keine Information zu übertragen ist) im logischen Zustand high ist, muß das Startbit ein logisches low sein. An diesem Wechsel des logischen Pegels der Leitung erkennt der Empfänger den Anfang einer Informationseinheit und synchronisiert sich selbst darauf. Der Zustand eines Stoppbit ist logisch high, also identisch mit dem Grundzustand der Übertragungsleitung.

Diese Schnittstelle wird einerseits dazu benutzt, um Peripheriegeräte zu bedienen, andererseits zur Kommunikation des Rechners über ein Modem mit anderen Rechnern. Im Hinblick auf die Datenfernübertragung mittels eines Modems ist diese Schnittstelle so ausgelegt, daß ein Rechner der Systemfamilie CBM 600/700 als Datenterminal (Daten-End-Einrichtung) zu interpretieren ist.

Ein Beispiel: der Buchstabe "A" soll übertragen werden;
 das Format ist: 1 Stoppbit, gerade Parität, 8 Bit
 Wortlänge ("A" = binär 1100 0011)

```

      ----      -
...XXXXI  IXXXXXXXXI      IXXXXXXXXI  IXXX  ...
      XXXXI  IXXXXXXXXI
-----
      Start 1  1  0  0  0  0  1  1  Par.Stop
      Bit    Bit    Bit    Bit    Bit    Bit    Bit

```


5.1 Extended BASIC 4.0 und RS 232 C

Die RS 232 C-Schnittstelle kann mit den normalen BASIC 4.0-Befehlen bearbeitet werden; lediglich beim Eröffnen dieses Datenkanals sind einige Spezifikationen für diese Schnittstelle zu machen. Dadurch ist ein komfortables Handhaben der Schnittstelle gewährleistet. Das Eröffnen geschieht mit dem normalen 'OPEN'-Befehl. Zur Erinnerung: Dieser Befehl hat folgendes Format:

OPEN lu,pa,sa,fp

lu = Logical unit, das ist die Nummer, unter der in Basic auf das entsprechende Medium zugegriffen wird.

pa = Primäadresse

sa = Sekundäadresse

fp = File-Parameter.

Bei der RS 232 C-Schnittstelle haben die einzelnen Parameter folgende Bedeutung:

lu: wie gewohnt.

pa: Die Primäadresse für die RS 232 C-Schnittstelle ist '2'.

sa: Die Sekundäadresse gibt die Übertragungsrichtung an.

fp: Der File-Parameter ist ein vier Byte langer Character-String, in welchem Angaben über Übertragungsgeschwindigkeit, Parität, Wortlänge und Stoppbits zu machen sind.

Die beiden ersten Parameter haben nichts außergewöhnliches an sich, die beiden anderen bedürfen einer genaueren Erläuterung:

5.1.1 Sekundäadresse (sa)

Die Sekundäadresse ist entsprechend der folgenden Tabelle anzugeben:

sa=1: Der Rechner sendet nur, empfängt aber keine Daten
(z.B. Ausgabe auf einen Drucker)

sa=2: Der Rechner empfängt nur, sendet aber nicht
(z.B. Meßwerterfassung)

sa=3: Der Rechner sendet und empfängt Daten
(z.B. Rechnerkommunikation)

Da der Zeichencode der CBM-Rechner (CBM-ASCII) in einigen Punkten vom Standard-ASCII abweicht, ist aus Gründen der Kompatibilität mit anderen Systemen vorgesehen, an der RS 232 C-Schnittstelle eine Code-Wandlung vorzunehmen.

Bei der Ausgabe wird dadurch das CBM-ASCII in Standard-ASCII umgewandelt, bei der Eingabe entsprechend von Standard-ASCII in CBM-ASCII.

Wenn diese Code-Konvertierung gewünscht wird, so ist der Wert der Sekundäradresse um 128 zu erhöhen. So ist z.B. als Sekundäradresse bei bidirektionalem Datenverkehr mit Code-Wandlung der Wert '131' (3 + 128) anzugeben.

5.1.2 Fileparameter (fp)

Der Fileparameter besteht aus einem 4 Byte langen Characterstring. Wichtig sind dabei die ersten beiden Byte, die letzten beiden Byte spielen keine Rolle, müssen jedoch angegeben werden. Zunächst die Bedeutung des ersten Byte, welches den stärksten Informationsgehalt hat. In diesem Byte sind die Angaben über die Übertragungsgeschwindigkeit, Anzahl der Stoppbits, sowie die Wortlänge enthalten. Den folgenden drei Tabellen ist jeweils der betreffende Wert zu entnehmen und diese drei Werte zu addieren. Dieser so ermittelte Wert stellt das erste Byte des Characterstrings dar.

Tabelle: Übertragungsgeschwindigkeit

Übertr.-geschw.	Wert	
16 * Ext. Takt	0	(nur zulässig beim Empfangen)
50 Baud	17	
75 Baud	18	
109.92 Baud	19	
134.58 Baud	20	
150 Baud	21	
300 Baud	22	
600 Baud	23	
1200 Baud	24	
1800 Baud	25	
2400 Baud	26	
3600 Baud	27	
4800 Baud	28	
7200 Baud	29	
9600 Baud	30	
19200 Baud	31	

Tabelle: Wortlänge

Wortlänge	Wert
8 Bit	0
7 Bit	32
6 Bit	64
5 Bit	96

Tabelle: Stoppbits

Stoppbits	Wert
1	0
1 (falls Wortlänge = 8 und Parität)	128
1.5 (nur bei Wortlänge = 5 und keine Parität)	128
2	128

Das zweite Byte baut sich nach einem ähnlichen Muster auf. Dieses Byte wird aus zwei Tabellen errechnet. Eigentlich wären zu diesem Byte fünf Tabellen nötig, um eine vollständige Leistungsbeschreibung zu bieten. Drei davon sind jedoch nur für Sonderfälle anwendbar, welche ein umfassendes Verständnis des Betriebssystems sowie der Hardware des Rechners und der Datenübertragung selbst voraussetzen. Sie sind hier nur wegen der Vollständigkeit der Tabellen aufgeführt; der Basic-Benutzer sollte sie ignorieren.

Der Wert des zweiten Byte ermittelt sich demnach aus der Summe jeweils eines Wertes aus den beiden folgenden Tabellen:

Tabelle: Modus

Modus	Wert
Normal	0
Echo	16 (nur in der Betriebsart Empfangen)

Der Modus 'Echo' bewirkt, daß die Daten, die der Rechner empfängt, sofort wieder an den Sender zurückgesendet werden, um dort die richtige Übermittlung der Daten zu kontrollieren (Duplex).

Tabelle: Parität

Parität		Wert
Empfangen	Senden	
keine	keine	0
ungerade	ungerade	32
gerade	gerade	96
keine	mark (log. 1)	160
keine	space (log. 0)	224

Hier nun die drei Tabellen, die für Standardanwendungen zu ignorieren sind:

Tabelle: Data Terminal Ready

Zustand	Wert
Blockieren Senden/Empfangen	0
Freigeben Senden/Empfangen	1

Tabelle: Interrupt freigeben

Interrupt	Wert
Freigeben Interrupt (Bit 0 Status)	0
Sperrern Interrupt	2

Tabelle: Sendekontrolle

Sendeunterbrechung	(not RTS)	Zusatz	Wert
Blockiert	high	-	0
Freigegeben	low	-	4
Blockiert	low	-	8
Blockiert	low	Senden: BRK	16

Zur Kontrolle der Datenübertragung steht die Systemvariable 'ST' zur Verfügung. Der Inhalt dieser Variablen hat folgende Bedeutung:

ST Bedeutung

0	alles okay
1	Paritätsfehler (nur beim Lesen !)
2	Formatfehler (nur beim Lesen; z.B. falsche Wortlänge)
4	Datenverlust (nur beim Lesen; die Daten wurden schneller angeliefert, als sie ausgelesen wurden)
16	Eingabepuffer leer (nur beim Lesen; beim letzten GET# war noch kein gültiges Zeichen im Puffer)
32	Fehler bei 'Data Carrier Detect'
64	Fehler bei 'Data Set Ready'

Eventuell können mehrere Fehler gleichzeitig auftreten. In diesem Fall sind die einzelnen Fehlermeldungen auszublenden (siehe Datenmanipulationen).

Wichtig:

Beim ersten Auslesen der Variablen 'ST' wird ihr Wert auf Null zurückgesetzt. Soll sie später noch verwendet werden, so ist ihr Wert einer anderen Variablen zuzuweisen.

z.B: 10 a=st:b=st: rem 'a' erhält den gewünschten Wert,
 20 rem 'b' erhält den Wert Null!

Die nun eröffnete RS 232 C-Schnittstelle kann mit den normalen BASIC 4.0-Befehlen für die Ein- und Ausgabe bearbeitet werden (GET#, PRINT#, INPUT#). Der Befehl 'INPUT#' sollte jedoch mit Vorsicht verwendet werden, da dieser Befehl am Ende der Eingabe das Zeichen 'Carriage Return' (dez. Wert: 13) erwartet. Trifft dieser Wert nicht ein, so bleibt der Rechner in der Input-Schleife hängen. Dieser Befehl sollte also nur dann verwendet werden, wenn das Eintreffen dieses Zeichens mit Sicherheit gewährleistet ist.

5.2 Die Hardware der Schnittstelle RS 232 C

Die RS 232 C-Schnittstelle wird von dem Baustein ACIA 6551 (Asynchronous Communication Interface Adapter) bedient. Dieser Integrierte Schaltkreis besitzt einen Baudraten-generator auf dem Chip, welcher Übertragungsraten von 50 bis 19200 Baud ermöglicht. Bei Verwendung eines externen Taktes können in der Betriebsart 'Empfangen' nicht-standardisierte Übertragungsgeschwindigkeiten bis zu 125000 Baud erreicht werden.

Dieser Baustein besitzt neben den internen Registern fünf weitere, die von außen erreichbar sind. Es sind dies das Control Register, das Command Register, das Status Register, das Transmitter Register und das Receiver Register. Diese Register liegen unter folgenden Adressen:

Register	Adresse (hex)	Adresse (dez)
Receiver Register	DD00	56576
Transmitter Register	DD00	56576
Status Register	DD01	56577
Control Register	DD02	56578
Command Register	DD03	56579

Zwischen dem Receiver Register und dem Transmitter Register wird durch die Art des Zugriffes unterschieden: lesender Zugriff spricht das Receiver Register an, schreibender Zugriff entsprechend das Transmitter Register.

Der Baustein ist ausführlich in dem entsprechenden Datenblatt beschrieben (xx). Hier sei nur soviel erwähnt, daß das erste Byte des Filestrings im 'OPEN'-Befehl in das Control Register und das zweite Byte in das Command Register übertragen wird. Das Status Register wird teilweise in die Systemvariable 'ST' kopiert.

An der RS 232 C-Schnittstelle werden 10 Anschlüsse ausgewertet. Die Verbindung zum Rechner ist über eine Standard 25-polige Cannon-Buchse (unter Technikern als 'Weibchen' bekannt) realisiert. Die verwendeten Anschlüsse sind hier mit den Anschlußnummern des Steckers und Ihren Bezeichnungen unter den verschiedenen Normen aufgeführt:

Stecker = Cannon	CCITT V.24	EIA RS232C	DIN 66...	Bedeutung (Abk.)
1	1	AA	101	prot. ground
2	2	BA	103	TxD
3	3	BB	104	RxD
4	4	CA	105	RTS
5	5	CB	106	CTS (RFS)
6	6	CC	107	DSR
7	7	AB	102	signal ground
8	8	CF	109	DCD
20	20	CD	108/2	DTR
24	24	??	??	RxC

Bedeutung:

protective ground: Schutz Erde, gemeinsame Masse

TxD (Transmitted Data): Über diese Leitung werden die auszu-
gebenden Daten abgesendet.

RxD (Received Data): Über diese Leitung wird ankommende Infor-
mation eingelesen.

RTS (Request to send): Auf dieser Leitung teilt die Daten-End-
Einrichtung (hier: Rechner CBM 600/700) dem Modem den
Wunsch mit, Daten abzusenden.

CTS (Clear to send; RFS, Ready for send): ist die Antwort des
Modems auf die Anfrage RTS

DSR (Data Set Ready): Mit dieser Leitung teilt das Modem dem
Rechner mit, daß es bereit ist, auf der Leitung vor-
handene Daten an den Rechner zu übermitteln.

signal ground: ist der gemeinsame Nulleiter für alle Signale.

DCD (Data Carrier Detect): Über diese Leitung teilt das Modem
dem Rechner mit, daß auf der Datenübertragungsstrecke
Information vorhanden ist, die der Rechner empfangen soll.

DTR (Data Terminal Ready): Der Rechner teilt dem Modem über diese
Leitung mit, daß es bereit ist, Daten zu empfangen.

RxC (Receive Clock): Auf dieser Leitung wird der Takt übermittelt,
wenn die Übertragungsgeschwindigkeit außerhalb der
genormten Werte liegt. (Nur in der Betriebsart Empfangen!).

Die Rechner der Systemfamilie CBM 600/700 erwarten an allen An-
schlüssen den richtigen logischen Pegel (ausgenommen 'RxC', welcher

nur ausgewertet wird, falls der externe Takt beim Eröffnen der Datenübertragungsleitung ausdrücklich erwünscht wird; siehe hierzu Tabelle 'Übertragungsgeschwindigkeit' im Kapitel 5.1.2).

Da es sehr häufig vorkommt, daß das an den Rechner anzuschließende Gerät nicht alle Kontakte unterstützt, können folgende Lötbrücken hergestellt werden, um ein ordnungsgemäßes Funktionieren zu ermöglichen:

1. Anschlüsse 4 (RTS) und 5 (CTS) verbinden und
2. Anschlüsse 6 (DSR), 8 (DCD) und 20 (DTR) verbinden.

Dies geschieht am einfachsten (und sichersten) durch Lötbrücken auf dem Stecker des Verbindungskabels.

ASCII CODE	CHARACTER
000	SHIFT
001	CTRL
002	C=
003	STOP
004	CE
009	TAB
010	CURSOR RIGHT mit SHIFT
013	ENTER
013	RETURN
014	NORM
017	CURSOR DOWN
018	RVS
019	HOME
020	DEL
025	CURSOR UP mit SHIFT
027	ESC
029	CURSOR RIGHT
032	SPACE
033	!
034	"
035	#
036	\$
037	%
038	&
039	'
040	(
041)
042	*
043	+
044	,
045	-
046	.
047	/
048	0
049	1
050	2
051	3
052	4
053	5
054	6
055	7
056	8
057	9
058	:
059	;
060	<
061	=
062	>
063	?
064	@
065	a
066	b
067	c
068	d
069	e
070	f

ASCII CODE	CHARACTER	
071	g	
072	h	
073	i	
074	j	
075	k	
076	l	
077	m	
078	n	
079	o	
080	p	
081	q	
082	r	
083	s	
084	t	
085	u	
086	v	
087	w	
088	x	
089	y	
090	z	
091	[
092	£	
093]	
094	↑	
095	←	
130	C=	mit SHIFT
131	RUN	mit SHIFT
132		
137	TAB	mit SHIFT
138	CURSOR LEFT	mit SHIFT
141		
142	GRAPH	
145	CURSOR UP	
146	OFF	mit SHIFT
147	CLEAR	mit SHIFT
148	INS	mit SHIFT
153		
155	ESC	mit SHIFT
157	CURSOR LEFT	
160	SPACE	
193	A	
194	B	
195	C	
196	D	
197	E	
198	F	
199	G	
200	H	
201	I	
202	J	
203	K	
204	L	
205	M	
206	N	
207	O	
208	P	
209	Q	
210	R	
211	S	
212	T	
213	U	
214	V	
215	W	
216	X	

ASCII CODE	CHARACTER	
217	Y	
218	Z	
222	π	
224	F1	
225	F2	
226	F3	
227	F4	
228	F5	
229	F6	
230	F7	
231	F8	
232	F9	
233	F10	
234	F1	mit SHIFT
235	F2	mit SHIFT
236	F3	mit SHIFT
237	F4	mit SHIFT
238	F5	mit SHIFT
239	F6	mit SHIFT
240	F7	mit SHIFT
241	F8	mit SHIFT
242	F9	mit SHIFT
243	F10	mit SHIFT
255		
255		

Die ASCII-Codes sind dezimal dargestellt

