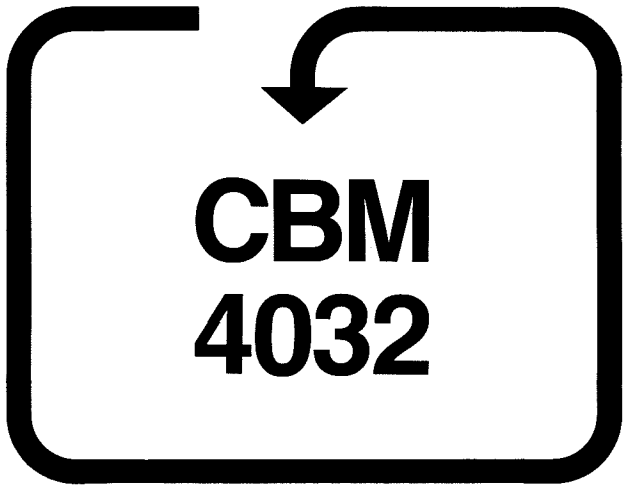


commodore - computer

Bedienungshandbuch

I. Hinweise zur Typographie

Dieses Handbuch wurde vollständig mit Text-Bearbeitungs-Programmen erstellt, die auf dem Markt für CBM-Rechner erhältlich sind und auf einem Typenraddrucker der gehobenen Klasse ausgedruckt, der über den IEC-Bus ganz regulär an den Rechner angeschlossen ist. Daran können Sie die mögliche Leistungsfähigkeit Ihres Computers erkennen.

Leider brachte es der deutsche Zeichensatz mit sich, daß einige Zeichen, die auf dem Bildschirm und der Tastatur Ihres Computers dargestellt werden können, nicht auf dem verwendeten Typenrad enthalten waren.

Es sind dies folgende Zeichen:

Doppelkreuz wird bei PRINT, INPUT und GET verwendet. Dieses Zeichen wird als § (Paragraphenzeichen) dargestellt. Da andererseits das Paragraphenzeichen auf dem Computer nicht zu finden ist, kann es durch diese 'falsche' Darstellung zu keinen Verwechslungen kommen.

Pfeil nach oben ist das Symbol für die Rechenoperation 'Potenzierung'. Dieses Symbol konnte überhaupt nicht dargestellt werden. Da die Potenzierung auch mit 'hoch' bezeichnet wird, haben wir 'Pfeil nach oben' durch 'h' ersetzt.

Spitze Klammer auf bzw. zu wird in BASIC als Symbol für 'größer als' bzw. 'kleiner als' verwendet. Diese Symbole sind aus der Mathematik abgeleitet. Auch diese beiden Zeichen konnten wir nicht verwenden und haben sie deshalb durch 'k' für 'kleiner als' und 'g' für 'größer als' ersetzt.

Eckige Klammern sind durch die Umlaute belegt. Da sie aber auch in der BASIC-Syntax nicht vorkommen, fällt ihr Verlust nicht ins Gewicht.

Wir bitten Sie um Verständnis für diese kleinen Unannehmlichkeiten.

II. Hinweise zum Inhalt

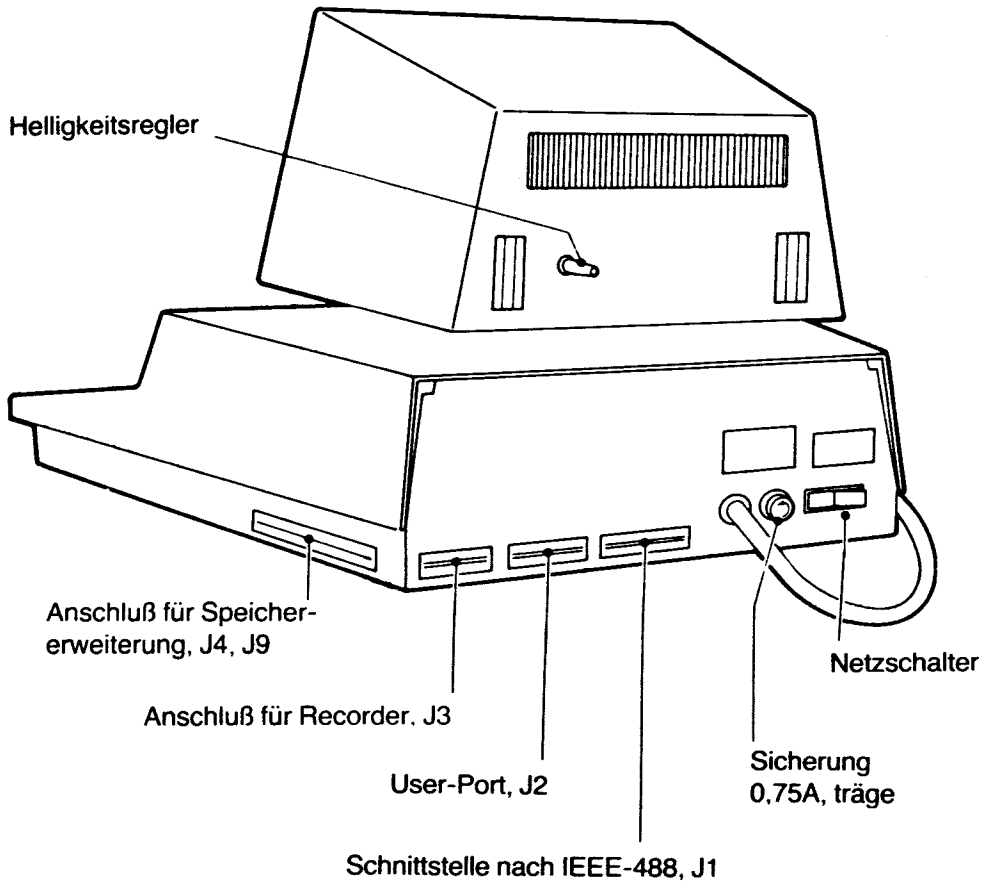
Der Teil A (Anwender-Handbuch) ist nur für Laien gedacht, die noch nicht mit Computern gearbeitet haben.

Teil B ist für den Programmierer geschrieben. Er setzt Programmier- und BASIC-Kenntnisse voraus und vermittelt nur detaillierte Fakten über das spezielle Commodore-BASIC.

Die Teile ab VII sind nur für speziell interessierte Personen gedacht, sie sind keinesfalls 'Pflichtlektüre' oder gehören zur Allgemeinbildung.

Im Übrigen dürfen wir noch auf das Vorwort zum Programmierer-Teil verweisen.


commodore
CBM
Model 4032


Helligkeitsregler
Anschluß für Speicher-
erweiterung, J4, J9
Anschluß für Recorder, J3
User-Port, J2
Schnittstelle nach IEEE-488, J1
Sicherung
0,75A, träge
Netzschalter

A ANWENDER - HANDBUCH

1. Inbetriebname

Ihr CBM 4032 ist ein hochwertiges elektronisches Gerät, konstruiert und gefertigt nach dem neuesten Stand der Microcomputer-Technologie. Wie bei allen technischen Anlagen, hängt auch bei diesem Computer ein großer Teil der Funktionsfähigkeit von der richtigen Bedienung durch den Anwender ab. Deshalb sollten Sie bei der Inbetriebnahme des Gerätes die folgenden Anweisungen genau beachten.

1.1 Verpackung

Der CBM wird anschlußfertig mit Bedienungshandbuch ausgeliefert. Die Spezialverpackung garantiert einen sicheren Transport und sollte deshalb aufbewahrt werden. Können Sie trotzdem an der Verpackung oder am Gerät Transportschäden feststellen, so melden Sie diese bitte umgehend Ihrem CBM-Fachhändler!

Fassen Sie das Gerät beim Auspacken nicht am Bildschirmaufbau!

1.2 Installation

Stecken Sie den Netzstecker in eine geerdete Steckdose (220 Volt). Achten Sie bitte darauf, daß am selben Stromkreis möglichst keine Geräte mit hoher Stromaufnahme angeschlossen sind, da sich starke Netzschwankungen störend auf den Rechnerbetrieb auswirken können.

1.3 Ein-/Aus-Schalter

Den Netzschalter finden Sie an der Rückseite des Gerätes. Nach dem Einschalten erscheint auf dem Bildschirm die Art des Betriebs-Systemes und die Anzahl der freien Speicherplätze. Der Rechner ist betriebsbereit:

```
*** COMMODORE BASIC 4.0 ***
 31743 BYTES FREE
READY.
```

Sollte der Bildschirm dunkel bleiben, so überprüfen Sie bitte nochmals den Netzanschluß und eventuell die Helligkeitseinstellung. Führt dies nicht zum Erfolg, so benachrichtigen Sie bitte Ihren Fachhändler.

1.4 Helligkeitsregler

Mit dem Helligkeitsregler an der Rückseite des Bildschirmaufbaues können Sie die Leuchtintensität der Bildröhre individuell nach Ihren Bedürfnissen einstellen. Beachten Sie dabei, daß ein zu hell eingestellter Bildschirm die Augen stark belastet und die Lebensdauer der Bildröhre reduziert!

2. Eigenschaften und Aufbau eines Computersystems

2.1 Einsatz und Eigenschaften

Ein Computer kann Aufgaben, die sich auf immer wiederkehrende Teilaufgaben zurückführen lassen, sehr schnell und fehlerfrei lösen. Seine Vorteile gegenüber der menschlichen Arbeitsweise sind seine hohe Geschwindigkeit und Zuverlässigkeit.

Er eignet sich deshalb hervorragend für langwierige Sortier- und Vergleichsarbeiten oder umfangreiche Berechnungen, kann also den Menschen von nervtötenden Routinearbeiten befreien, oder ihn wenigstens dabei unterstützen. Damit kann der Computer gleichzeitig Arbeitsvorgänge rationalisieren und humanisieren.

Trotzdem kann er das menschliche Gehirn nicht ersetzen, da ihm die Fähigkeit des kreativen Denkens fehlt. Er kann also nicht denken, sondern nur rechnen und die Ergebnisse mit vorgegebenen Mustern vergleichen und aufgrund von vorprogrammierten Entscheidungen bestimmte Reaktionen auswählen. Er ist deshalb immer auf die Funktionen beschränkt, auf die er von Menschen programmiert wurde.

Diese Funktionen sind dem Computer aber nicht fest eingebaut, sondern können fast beliebig ausgewechselt werden. Das Computersystem ist dabei nur eine Maschine (Hardware), die zusammen mit den Programmen (Software) die vom Anwender gewünschte Leistung erbringt.

Als Vergleich könnte man eine Stereoanlage nehmen, die aus mehreren Geräten zusammengestellt ist. Diese Geräte entsprechen dem Computersystem (Hardware). Damit diese Maschinen aber Ihren Zweck erfüllen können, müssen Sie zusätzlich Schallplatten oder Tonbänder kaufen und in die Anlage 'einlegen'. Auf diesen Tonträgern, die den Datenträgern beim Computer entsprechen, befindet sich die austauschbare geistige Leistung (Software), die Sie eigentlich haben wollen.

Zuerst wollen wir nun allgemein die Bestandteile der Hardware vorstellen und hinterher etwas auf Software aus Anwendersicht eingehen.

2.2 Die Hardware

In der konventionellen EDV (Elektronische Daten Verarbeitung) unterscheidet man beim Computer folgende Hauptbestandteile:

ZENTRALEINHEIT	Hier werden die Informationen verarbeitet und die Steuerung des Informationsflusses zwischen Rechner, Speicher und Peripherie geregelt.
ARBEITSSPEICHER	Hier werden die Informationen abgelegt, die der Zentraleinheit unmittelbar zur Verfügung stehen müssen (Programm und Daten).
EINGABEGERÄTE	Sie dienen der Informationseingabe in den Rechner. (Tastatur, Meßgeräte)
AUSGABEGERÄTE	Ausgabe von Daten, Meldungen und Ergebnissen an den Benutzer (Bildschirm, Drucker).
MASSENSPEICHER	Langzeitspeicher für Programme und Daten. (Magnetband, Floppy-Disk, Magnet-Platte)

Während bei der Groß-EDV die einzelnen Einheiten meist in eigenen Gehäusen von beachtlicher Größe untergebracht sind, sind beim CBM Zentraleinheit, Arbeitsspeicher, Bildschirm und Eingabetastatur in einem Gerät mit der Stellfläche einer elektrischen Schreibmaschine vereint. Ermöglicht wurde dies durch konsequenten Einsatz der modernen Microprozessor-Technik, zu deren Entwicklung Firmen der COMMODORE-Unternehmensgruppe einen wesentlichen Beitrag geleistet haben.

2.2 Die Software

Wir unterscheiden zwischen Betriebs- und Anwender-Software.

Betriebs-Software sind jene Programme, die die technische Struktur des Computers ergänzen und seine Möglichkeiten erschließen. Die Betriebssoftware ist beim CBM fest in ROMs (Read Only Memory) im Gerät installiert. Dies hat den Vorteil, daß sie sofort nach dem Einschalten des Gerätes zur Verfügung steht und außerdem vor versehentlichem Löschen oder Überschreiben geschützt ist. Die Betriebssoftware wird auch Betriebssystem genannt. Für den Anwender ist allerdings weder ersichtlich noch wichtig, wo die Hardware aufhört und das Betriebssystem beginnt.

Anwender-Software sind die Programme, die als Bindeglied zwischen dem Computer und dem Anwender fungieren. In ihnen sind anwenderspezifische Arbeitsabläufe so festgelegt, daß sie vom Computer in der richtigen Reihenfolge abgearbeitet werden können. Diese Programme erst bringen den Computer dazu, die Funktionen zu übernehmen, die Sie von ihm erwarten (z.B. Buchhaltung, Lagerverwaltung).

Anwenderprogramme können Sie entweder selbst erstellen, oder erstellen lassen. Ausserdem bietet sich die Möglicheit, fertige Standardprogramme zu kaufen. Da Ihr Computer aber nur so gut ist wie das Programm, mit dem er gefüttert wird, sollten Sie beim Softwarekauf besonders auf Qualität achten. Der Preis ist dabei nicht unbedingt ein aussagekräftiges Kriterium, denn auch gute Software kann bei entsprechend hoher Auflage günstig angeboten werden. Deshalb hier einige Bedingungen, die ein gutes Programm erfüllen sollte:

* Übersichtliche Bedienungsanleitung
* Übersichtlicher Bildschirmaufbau
* Gute Bedienerführung im Dialog
* Absicherung gegen Fehlbedienung (Plausibilitätskontrollen)
* Einfache Fehlerberichtigung

Ihr Commodore-Fachhändler ist gerne bereit, Sie in Softwarefragen zu beraten.

3. Aufbau Ihres Commodore-Systems

3.1 Der Computer

Wie erwähnt, enthält Ihr Computer bereits die Zentraleinheit und den Arbeitsspeicher, sowie die beiden wichtigsten Ein- / Ausgabe - 'Geräte', nämlich die Tastatur und den Bildschirm. Lediglich Massenspeicher und Drucker müssen zusätzlich durch Kabel angeschlossen werden.

3.2 Der Bildschirm

Ihr CBM ist ein dialogorientiertes Gerät. Die Kommunikation mit dem Rechner erfolgt über Tastatur und Bildschirm. Dem Bildschirm fallen dabei zwei wesentliche Aufgaben zu.

Zum Ersten dient er als Kontrollmedium für Eingaben. Das heißt, jedes über die Tastatur eingegebene Zeichen erscheint sofort auf dem Bildschirm. Dabei werden die einzelnen Zeichen folgerichtig in einer Zeile ausgegeben, ähnlich, wie es bei einer Schreibmaschine der Fall wäre. Dies ermöglicht eine einfache Fehlerkorrektur.

Aber auch dem Rechner dient der Bildschirm als Instrument, um Meldungen an den Benutzer auszugeben. So werden zum Beispiel Fehlermeldungen unmittelbar auf den Bildschirm geschrieben.

Bildschirmaufbau

Eine Bildschirmseite umfaßt 25 Zeilen mit je 40 Zeichen. Es lassen sich also maximal 1000 Zeichen gleichzeitig auf dem Bildschirm darstellen.

Am Zeilenende erfolgt ein automatischer Überlauf zum Anfang der nächsten Zeile. Am Ende der unteren Bildschirmzeile wird der gesamte Text auf dem Bildschirm um eine Zeile hochgerückt. Dadurch verschwindet die erste Zeile. Dieses Durchrollen des Bildes nennt man SCROLLING.

Die jeweilige Schreibposition für das nächste Zeichen wird durch ein blinkendes Feld angezeigt, dem CURSOR.

3.3 Die Tastatur

Die Tastatur bietet dem Benutzer die Möglichkeit, dem Computer Befehle, Daten oder ganze Programme zu übermitteln. Deshalb ist es von Vorteil, sich mit diesem wichtigen Bestandteil des Gerätes vertraut zu machen. Die Tastenfunktionen können ohne Bedenken ausprobiert werden.

Die Tastatur des CBM 4032 besteht aus zwei getrennten Tastenfeldern, der **Tastatur mit Buchstaben und Sonderzeichen** und dem **numerischen Zehnerblock** mit Rechenzeichen und Funktionstasten (rechts).

Neben den schon von der Schreibmaschine her bekannten Tasten findet man beim CBM noch zusätzliche **Funktionstasten.** Diese Tasten liefern keine Zeichen, sondern haben Steuer- und Kontrollfunktionen. Eine Übersicht dieser Funktionen finden Sie einige Seiten weiter bei der **Bedienung der Tastatur.**

3.4 Die Peripheriegeräte

Beachten Sie bitte, daß der Anschluß externer Geräte nur im ausgeschalteten Zustand erfolgen darf.

3.4.1 Datasette C2N

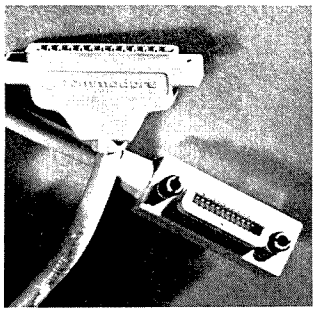
Der Rekorder ist ein preiswerter Langzeitspeicher für Programme und kleine Datensätze. Er arbeitet mit normalen, handelsüblichen Bandkassetten. Die Anschlußbuchse finden Sie an der Rückseite des CBM. Der codierte Stecker kann nicht falsch angeschlossen werden.

3.4.2 Geräte am IEC-BUS

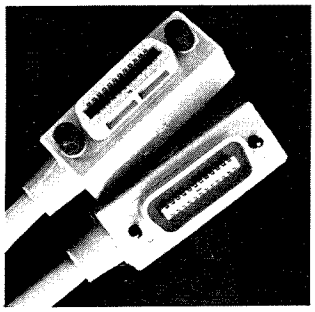
Der IEC-BUS, konzipiert nach der Norm IEEE 488, ist der wichtigste Anschluß am CBM. An ihm lassen sich bis zu 12 Geräte anschließen. Die Steckerleiste ist ebenfalls an der Geräterückseite des CBM zu finden.

3.4.3 Kabeltypen für den IEC-Bus-Anschluß

Das erste IEC-Gerät wird durch das Interfacekabel **Computer-Peripherie** mit dem CBM verbunden. Dieses Kabel hat an der einen Seite einen Flachstecker, der mit der Beschriftung nach oben in die Steckerleiste des CBM paßt. Der IEEE 488 Normstecker am anderen Ende des Kabels wird mit dem anzuschließenden Peripheriegerät verbunden. Sollen weitere IEC-Geräte angeschlossen werden, so kann das mittels des Verbindungskabels **Peripherie-Peripherie** geschehen. Dieses Kabel hat an beiden Enden Normstecker und wird einfach "huckepack" auf den Stecker am ersten Peripheriegerät aufgesteckt. Auf diese Weise lassen sich alle Geräte am Bus anschließen.



Kabel Computer-Peripherie



Kabel Peripherie-Peripherie

3.4.4 Floppy-Disk

Floppy-Laufwerke bieten im Gegensatz zum Recorder einen wesentlich schnelleren Zugriff auf Daten und Programme. Als Speichermedium dienen hier dünne, flexible Magnetplatten. Die Diskettenorganisation erfolgt über Spuren und Sektoren und wird über das fest eingebaute DOS (Disk Operating System) verwaltet. Der CBM Computer kann mit den folgenden Floppystationen gekoppelt werden:

TYP	Anz. Laufwerke	KB/Diskette	Diskettengröße
CBM 3040	2	170	5.25"
CBM 4040	2	170	5.25"
CBM 8050	2	512	5.25"

Bei der Arbeit mit Disketten sollten Sie folgendes beachten:

* Nur Qualitätsdisketten verwenden!
* Gerät nie mit geschlossenen Laufwerksklappen aus- oder einschalten!
* Disketten vor dem Einlegen von Hand vorzentrieren!
* Disketten nur an der Schutzhülle anfassen!
* Disketten nie in die Nähe von Magnetfeldern bringen.

3.4.5 Drucker

Als Ausgabeeinheiten werden von COMMODORE verschiedene Drucker angeboten. Auch diese werden am IEC-BUS betrieben:

CBM 4022	Matrixdrucker mit Traktorführung 80 Zeichen pro Zeile max. 150 Zeichen pro Sekunde
CBM 8024	Matrixdrucker mit Traktorführung 132 Zeichen pro Zeile 170 Zeichen pro Sekunde mit Druckwegoptimierung
CBM 8026	Typenraddrucker, auch als Schreibmaschine verwendbar 140 Zeichen pro Zeile ca. 25 Zeichen pro Sekunde

Ihr Commodore-Händler ist Ihnen gerne bei der Auswahl des für Sie optimalen Druckers behilflich.

4. Bedienung der Tastatur

4.1 Tastenfunktionen der linken Seite

Taste	ohne SHIFT	mit SHIFT
OFF RVS	Umschaltung auf negative Zeichendarstellung. (Dunkle Schrift auf hellem Grund)	Umschaltung auf normale Zeichendarstellung. (Helle Schrift auf dunklem Grund)
	außerdem: Reduzierte Geschwindigkeit bei Bildschirmausgabe (z.B. Listing)	wie ohne SHIFT
SHIFT LOCK	Feststeller für SHIFT-Taste	wie ohne SHIFT

4.2 Tastenfunktionen der rechten Seite (Haupttastenfeld)

Taste	ohne SHIFT	mit SHIFT
RETURN	Die eingegebene Information wird an den Rechner übergeben. Der Cursor wird an den Anfang der nächsten Zeile gesetzt.	Die Information wird **nicht** vom Rechner übernommen. Der Cursor wird an den Anfang der nächsten Zeile gesetzt
RUN STOP	Unterbricht den Programmablauf	Laden und starten des ersten Programmes vom Laufwerk 0. (nicht bei 3040)

4.3 Tastenfunktionen über dem Zehnerblock

Taste	ohne SHIFT	mit SHIFT
CLR HOME	Cursor wird in die linke obere Ecke des Bildschirmes gebracht.	Bildschirm wird zusätzlich gelöscht.
CNO CNU	Bewegt den Cursor um eine Zeile nach unten. Ab der letzten Zeile wird der Bildschirminhalt um eine Zeile hochgeschoben.	Bewegt der Cursor um eine Zeile nach oben.
CNR CNL	Bewegt den Cursor um ein Zeichen nach rechts. Am Zeilenende wird der Cursor an den Anfang der nächsten Zeile gesetzt.	Bewegt den Cursor um ein Zeichen nach links. Am Zeilenanfang wird der Cursor an das Ende der darüberliegenden Zeile gesetzt.
INST DEL	Löschen des Zeichens links neben dem Cursor. Die restliche Zeile rechts neben dem Cursor wird um ein Zeichen nach links gezogen.	Einfügen eines Leerzeichens an der Stelle des Cursors. Der rechte Teil der Zeile wird um ein Zeichen nach rechts geschoben.

4.4 Arbeiten mit der Tastatur

Um etwas mehr mit der Wirkungsweise der einzelnen Tasten vertraut zu werden, sollten Sie folgende Beispiele nachvollziehen.

Schreiben

Versuchen Sie Ihren Namen auf den Bildschirm zu schreiben. Hierbei ist zu beachten, daß im nach dem Einschalten des Rechners nur Großbuchstaben und Graphikzeichen auf den Bildschirm schreiben kann. Die Umschaltung auf Großund Kleinbuchstaben wird unter 'Text-Modus' beschrieben.

Alle Zeichen, die Sie mit gleichzeitig gedrückter SHIFT-Taste schreiben, werden als Graphikzeichen dargestellt, die auf der Vorderseite jeder Taste angegeben sind.

Drücken Sie jetzt die Taste SHIFT LOCK. Die SHIFT-Taste ist jetzt festgestellt, Sie können nur noch Graphikzeichen auf den Bildschirm schreiben. Das Lösen des SHIFT LOCK geschieht durch nochmaliges Drücken der Taste.

Achtung:

SHIFT kann beim Programmablauf zu Fehlern führen, wenn z.B. Eingaben mit einem SHIFT-RETURN abgeschlossen werden.

Drücken Sie nun die Taste OFF RVS und schreiben Sie einen neuen Text. Dieser Text erscheint nun in Negativdarstellung (REVERSE) auf dem Bildschirm (Dunkle Schrift auf hellem Grund). Durch erneutes Betätigen der Taste gemeinsam mit SHIFT wird der Reverse-Modus wieder abgeschaltet.

Cursor horizontal bewegen (rechts / links)

Drücken Sie kurz die Taste CRSR (Cursor rechts). Der Cursor wandert im selben Moment eine Spalte nach rechts. Nach Erreichen der 40. Stelle setzt der Cursor automatisch am Anfang der nächsten Zeile auf. Betätigen Sie jetzt gleichzeitig die SHIFT-Taste, so bewegt sich der Cursor genau in die entgegengesetzte Richtung. Bewegt man den Cursor über beschriebene Stellen des Bildschirmes, so wird der bestehende Text vom Cursor nicht gelöscht.

Cursor vertikal bewegen (unten / oben)

Betätigen Sie die Taste CRSR (Cursor nach unten). Sie werden feststellen, daß sich der Cursor in der Spalte, in der er sich gerade befindet, senkrecht nach unten über den Bildschirm bewegt. In Verbindung mit SHIFT kann er auch hier in die entgegengesetzte Richtung dirigiert werden. Um zu verdeutlichen, was geschieht, wenn der Cursor den unteren Bildschirmrand erreicht, tippen Sie einige Zahlen oder Buchstaben und fahren den Cursor anschließend in die untere Bildschirmzeile. Wenn Sie die Taste noch öfter drücken, werden Sie sehen, daß der geschriebene Text Zeile für Zeile nach oben wandert, bis er am oberen Bildschirmrand verschwindet. Dieses zeilenweise Durchrollen des Textes nennt man SCROLLEN. Der Scroll-Modus wirkt nur in einer Richtung. Der Bildschirminhalt wandert also nicht nach unten, wenn der Cursor oben anstößt.

4.4 Arbeiten mit der Tastatur

Um etwas mehr mit der Wirkungsweise der einzelnen Tasten vertraut zu werden, sollten Sie folgende Beispiele nachvollziehen.

Schreiben

Versuchen Sie Ihren Namen auf den Bildschirm zu schreiben. Hierbei ist zu beachten, daß im nach dem Einschalten des Rechners nur Großbuchstaben und Graphikzeichen auf den Bildschirm schreiben kann. Die Umschaltung auf Großund Kleinbuchstaben wird unter 'Text-Modus' beschrieben.

Alle Zeichen, die Sie mit gleichzeitig gedrückter SHIFT-Taste schreiben, werden als Graphikzeichen dargestellt, die auf der Vorderseite jeder Taste angegeben sind.

Drücken Sie jetzt die Taste SHIFT LOCK. Die SHIFT-Taste ist jetzt festgestellt, Sie können nur noch Graphikzeichen auf den Bildschirm schreiben. Das Lösen des SHIFT LOCK geschieht durch nochmaliges Drücken der Taste.

Achtung:

SHIFT kann beim Programmablauf zu Fehlern führen, wenn z.B. Eingaben mit einem SHIFT-RETURN abgeschlossen werden.

Drücken Sie nun die Taste OFF RVS und schreiben Sie einen neuen Text. Dieser Text erscheint nun in Negativdarstellung (REVERSE) auf dem Bildschirm (Dunkle Schrift auf hellem Grund). Durch erneutes Betätigen der Taste gemeinsam mit SHIFT wird der Reverse-Modus wieder abgeschaltet.

Cursor horizontal bewegen (rechts / links)

Drücken Sie kurz die Taste CRSR (Cursor rechts). Der Cursor wandert im selben Moment eine Spalte nach rechts. Nach Erreichen der 40. Stelle setzt der Cursor automatisch am Anfang der nächsten Zeile auf. Betätigen Sie jetzt gleichzeitig die SHIFT-Taste, so bewegt sich der Cursor genau in die entgegengesetzte Richtung. Bewegt man den Cursor über beschriebene Stellen des Bildschirmes, so wird der bestehende Text vom Cursor nicht gelöscht.

Cursor vertikal bewegen (unten / oben)

Betätigen Sie die Taste CRSR (Cursor nach unten). Sie werden feststellen, daß sich der Cursor in der Spalte, in der er sich gerade befindet, senkrecht nach unten über den Bildschirm bewegt. In Verbindung mit SHIFT kann er auch hier in die entgegengesetzte Richtung dirigiert werden. Um zu verdeutlichen, was geschieht, wenn der Cursor den unteren Bildschirmrand erreicht, tippen Sie einige Zahlen oder Buchstaben und fahren den Cursor anschließend in die untere Bildschirmzeile. Wenn Sie die Taste noch öfter drücken, werden Sie sehen, daß der geschriebene Text Zeile für Zeile nach oben wandert, bis er am oberen Bildschirmrand verschwindet. Dieses zeilenweise Durchrollen des Textes nennt man SCROLLEN. Der Scroll-Modus wirkt nur in einer Richtung. Der Bildschirminhalt wandert also nicht nach unten, wenn der Cursor oben anstößt.

CLR / HOME

Drücken Sie jetzt die Taste CLR HOME. Der Cursor geht in die linke obere Ecke des Bildschirms (Home-Position).

5. Rechenfunktionen

Ihr CBM läßt sich auch als Tischrechner verwenden. Die Rechenfunktionen sind vom Betriebssystem auch ohne Programm abrufbar.

Dabei gilt es zu beachten, daß in BASIC eine besondere Schreibweise für Rechenoperationen vorgeschrieben ist. So werden hier an Stelle eines Kommas die Dezimalstellen durch einen Punkt getrennt.

Für die Darstellung arithmetischer Operationen gilt:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
	Potenzierung (Pfeil nach oben)

Um eine Ausgabe des Ergebnisses auf dem Bildschirm zu erreichen, ist es nötig, an den Anfang der Rechenoperation ein ? einzugeben.

Beispiel:

```
? 4*5          (RETURN drücken)
 20
ready
```

Der Rechner schreibt sofort das Ergebnis (20) unter die Eingabe. Die Meldung READY besagt, daß der Computer für neue Anweisungen bereit ist.

Weitere mathematische Funktionen müssen in BASIC- Schreibweise eingegeben werden. Die Funktion wird in diesem Fall als BASIC-Wort, der zu behandelnde Wert (Argument) in Klammern dahinter eingetippt.

Beispiel:

'Berechne Quadratwurzel aus 16' wird eingegeben mit

```
? sqr(16)        (RETURN drücken)
  4
ready
```

Folgende Funktionen stehen zur Verfügung:

sqr(x)	Quadratwurzel von x
sin(x)	Sinus von x (Bogenmaß)
cos(x)	Cosinus von x
tan(x)	Tangens von x
atn(x)	Arcustangens von x (Umkehrfunktion des Tangens)
log(x)	Natürlicher Logarithmus zur Basis e=2,7182818
exp(x)	Exponent zur Basis e
int(x)	Integer (Schneidet Nachkommastellen der Zahl x ab)
abs(x)	Absolutwert der Zahl x (Vorzeichenunterdrückung)
sgn(x)	Signum (Ermittelt das Vorzeichen der Zahl x)

6. Systembefehle

6.1 Einführung

Systembefehle sind Kommandos, die der Rechner direkt annimmt und ausführt. Diese Befehle sind notwendig, um Programme laden, speichern, oder verändern zu können.

Befehle werden grundsätzlich **ohne SHIFT** geschrieben (nur klein im Textmodus oder nur groß im Graphikmodus) und mit der Taste RETURN zur Ausführung an den Rechner übergeben.

Als Anwender müssen Sie natürlich nicht alle Systembefehle beherrschen. Wir haben hier nur die allerwichtigsten zusammengestellt. Sie werden sehen, daß alle hier aufgeführten Befehle voraussetzen, daß Sie ein Floppy-Laufwerk angeschlossen haben. Bitte beachten Sie, daß beim CBM 3040 geringfügige Abweichungen von den beschriebenen Methoden nötig sind, sie gelten also nur für 4040 und 8050.

Sollten Sie nur über einen Rekorder verfügen, so schlagen Sie bitte bei **Rekorderverwaltung** nach, dort sind alle benötigten Befehle speziell für den Rekorder zusammengestellt.

Eine kurze Übersicht soll Ihnen zeigen, welche Befehle im folgenden erklärt werden:

SHIFT/RUN	erstes Programm laden und starten
DLOAD	beliebiges Programm laden
RUN	Programm starten
DIRECTORY	Inhaltsverzeichnis der Diskette lesen
BACKUP	Diskette duplizieren (Sicherungsduplikat)
HEADER	Neue Diskette anlegen
DSAVE	Programm abspeichern
COPY	Datei kopieren
SCRATCH	Datei löschen

Die Befehle sind in der Reihenfolge erklärt, wie Sie von Ihnen am wahrscheinlichsten benötigt werden. D.h. die weiter hinten beschriebenen Befehle benötigen Sie eventuell am Anfang gar nicht.

6.2 Erstes Programm laden und starten

Wenn Sie die Tasten **SHIFT und RUN** drücken, nachdem Sie die Anlage eingeschaltet haben, wird das erste Programm der Diskette im rechten Laufwerk geladen und automatisch gestartet.

Bei qualitativ hochwertiger Software ist dieses erste Programm ein sogenannter Systemstart, der erstens alle benötigten Programme automatisch lädt und zweitens ab sofort mit Ihnen in deutschem Dialog in Verbindung tritt. In der Regel können Sie dann durch Drücken einer Taste ein bestimmtes Programm aus einer ganzen Anzahl von angebotenen Programmen laden und starten.

6.3 Beliebiges Programm laden

Falls Sie unabhängig von einem 'Systemstart-Programm' ein bestimmtes Programm einer Diskette laden wollen, können Sie dies durch

DLOAD "NAME"

tun. Dieser Befehl lädt das Programm 'NAME' vom **rechten Laufwerk** in den Arbeitsspeicher. Er teilt den Ladevorgang durch 'LOADING' mit und meldet sich mit 'READY.' zurück.

Falls das gewünschte Programm auf der eingelegten Diskette nicht vorhanden ist oder irgendein Diskettenfehler auftritt, wird

FILE NOT FOUND ERROR

gemeldet. Überprüfen Sie in diesem Fall die richtige Schreibweise des Namens und ob die richtige Diskette eingelegt ist.

Wollen Sie ein Programm vom **linken Laufwerk** laden, so fügen Sie an den oben beschriebenen Befehl noch durch Komma getrennt 'D1' an:

DLOAD "NAME" , D1

6.4 Programm starten

Ein Programm, das Sie durch DLOAD in den Speicher geladen haben, müssen Sie noch durch

RUN

starten, damit es für Sie tätig werden kann.

6.5 Inhaltsverzeichnis der Diskette lesen

Wenn Sie wissen wollen, welche Programme oder Dateien auf einer Diskette sind, können Sie durch

DIRECTORY

die Inhaltsverzeichnisse beider Disketten nacheinander auf den Bildschirm bringen. Denken Sie dabei an die Funktion der beiden Tasten ':/*' bzw. 'Pfeil nach links'.

Durch Anfügen von 'D0' oder 'D1' können Sie gezielt das Inhaltsverzeichnis der rechten oder linken Diskette lesen:

DIr D0
DIr D1

Wenn Sie den dritten Buchstaben von DIRECTORY **mit SHIFT** eingeben, versteht dies der Rechner als Abkürzung und Sie müssen das furchtbar lange Wort nicht immer ausschreiben.

6.6 Diskette duplizieren

Von Datendisketten kann nie zu oft ein Sicherungsduplikat erstellt werden, um bösen Überraschungen durch Verlust wertvoller Daten vorzubeugen. Sie können Disketten vollständig duplizieren mit

```
    BACKUP D0 TO D1  rechte Diskette auf linke übertragen
(   BACKUP D1 TO D0  linke Diskette auf rechte übertragen     )
```

Die Form des BACKUP ist:

BACKUP (dupliziere von) D (Laufwerk) 0 TO (nach) D (Laufwerk) 1

Links von 'TO' steht also das Quellaufwerk, rechts das Ziellaufwerk.

Wenn Sie immer die gleiche Richtung (D0 TO D1) verwenden, laufen Sie weniger Gefahr, irgendwann die Richtung zu verwechseln.

Die Diskette in dem Laufwerk, das rechts von TO aufgeführt ist, wird völlig gelöscht!

Da eine falsche Anwendung dieses Befehls die Vernichtung der Daten zur Folge haben kann, die Sie eigentlich sichern wollten, beachten Sie bitte die folgenden Hinweise genau:

Schützen Sie die 'Quelldiskette' immer mit einem 'Schreibschutzaufkleber'!

Vergewissern Sie sich, daß die 'Zieldiskette' wirklich leer ist bzw. gelöscht werden darf!

6.7 Neue Diskette anlegen

Fabrikneue Disketten müssen erst durch einen speziellen Befehl 'sektoriert' werden, ehe sie Programme bzw. Daten aufnehmen können.

Durch die Anweisung

HEADER "NAME" , IXX

wird die Frage

ARE YOU SHURE?

ausgelöst, die sicherstellen soll, daß Sie nicht aus Versehen eine Diskette löschen, die bereits Daten enthält. Diese Frage ist mit 'Y' oder 'YES' zu beantworten, sonst wird die HEADER-Anweisung ignoriert.

'NAME' ist der Diskettenname, er darf maximal 16 Zeichen lang sein. Die beiden XX hinter I stehen stellvertretend für die 'Identität' der Diskette. Diese Kennzeichnung ist zwei Zeichen lang.

Lassen Sie den Zusatz 'IXX' weg, so wird die Diskette nur gelöscht, aber nicht sektoriert. Dies geht nur bei Disketten, die bereits in Gebrauch waren.

Durch die oben angeführte Anweisung wird die Diskette im rechten Laufwerk sektoriert. Wie schon bei den vorhergehenden Befehlen, kann durch Zusatz von 'D1' erreicht werden, daß die Diskette im linken Laufwerk sektoriert wird. Die Reihenfolge der einzelnen Angaben NAME, I oder D ist unwichtig, wichtig ist nur, daß sie jeweils durch Komma getrennt werden.

6.8 Programm abspeichern

Ein im Arbeitsspeicher stehendes BASIC-Programm können Sie mit

DSAVE "NAME"

auf die Diskette im rechten Laufwerk abspeichern. Auch hier kann durch Anfügen von D1 auf das linke Laufwerk abgespeichert werden.

6.9 Datei kopieren

Einzelne Dateien können von einem Laufwerk auf das andere durch

COPY "NAME" , D0 TO "NAME" , D1

kopiert werden. Wie schon bei BACKUP steht links von TO das Quellaufwerk und rechts das Ziellaufwerk. Neu sind die beiden Namen. Auch hier steht links der Name der Datei, die kopiert werden soll und rechts der Name der neuen Datei. Sie können - müssen aber nicht - gleich sein.

Wenn kein Name angegeben wird, werden alle Dateien der Quelldiskette auf die Zieldiskette kopiert. Der Unterschied zu BACKUP besteht darin, daß bei COPY die Zieldiskette nicht gelöscht wird, sie kann also bereits Dateien enthalten, zu denen noch die Dateien der Quelldiskette hinzukopiert werden.

6.10 Datei löschen

Einzelne Dateien können Sie durch die Anweisung

SCRATCH "NAME"

löschen. Wenn der Zusatz 'D1' fehlt, wird auf dem rechten Laufwerk gelöscht. Wie bei HEADER wird mit

ARE YOU SHURE?

nachgefragt, ob Sie dies ernst meinen. Antworten Sie auch hier durch 'Y'. Nach erfolgreichem Löschen wird gemeldet:

01 FILES SCRATCHED 01 00

Die zweite Zahl von rechts sagt Ihnen, daß 1 Datei gelöscht wurde.

Eine gelöschte Datei kann nicht wieder zurückgeholt werden, wenden Sie diese Anweisung also ähnlich sorgfältig an wie BACKUP und HEADER.

B. PROGRAMMIERER - HANDBUCH

Vorwort

Dieser Abschnitt soll kein Leitfaden zum Erlernen von allgemeinen Programmiertechniken sein und auch nicht zum Erlernen von BASIC, sondern ein Nachschlagewerk, in dem alle wissenswerten Fakten über die Programmierung Ihres COMMODORE - Computers zusammengestellt sind.

Lassen Sie sich von der Informationsfülle nicht abschrecken, Sie haben ja ein Nachschlagewerk vor sich. Die Informationen wurden sehr sorgfältig ausgewählt. Grundlage war die große Erfahrung des SOFTWAREVERBUND-MICROCOMPUTER in der Programmierung von Commodore-Rechnern und die Erfahrung aus vielen Anwenderseminaren, die vom Autor durchgeführt wurden.

Der Abschnitt B. gliedert sich in einen Teil über Datenarten und Interpretereigenschaften und einen Teil, der alle BASIC-Anweisungen behandelt. Zuerst sollten Sie unbedingt die ersten Kapitel über Datenarten usw. gelesen haben, da in den Befehlsbeschreibungen auf diese Definitionen zurückgegriffen wird.

Wenn Sie schon irgendeine Programmiersprache beherrschen, werden Sie sich sicher in diesem Handbuch schnell zurechtfinden.

Kurze Einführungen sollen aber auch dem Anfänger ermöglichen, sich in die Materie einzuarbeiten.

Für Anfänger wollen wir noch zwei Hinweise geben:

(1) Haben Sie Mut zur Lücke. Das soll heißen, daß Sie sich nicht beim ersten Durchlesen an irgendwelchen Einzelheiten verbeißen sollen, die Ihnen unverständlich erscheinen.

Viele Details werden erst im Zusammenhang mit anderen Fakten verständlich, so daß Sie also mindestens einmal alles gelesen haben müssen, um beim nächsten Mal mehr verstehen zu können.

Einige Details und auch ganze Anweisungen werden Sie am Anfang nicht benötigen, vielleicht auch nie. Merken Sie sich bei solchen vernachlässigten Stellen aber an, daß Sie sie nicht gelesen oder nicht verstanden haben. Vielleicht schmökern Sie später nochmal alle Stellen durch, die Sie so gekennzeichnet haben und finden doch noch den einen oder anderen wertvollen Hinweis.

(2) Erkundigen Sie sich bei Ihrem Commodore-Händler nach Schulungskursen. In der Regel ist es leichter, sich nach oder während einer Einführung in Seminarform mit dem Computer anzufreunden, als sofort ins kalte Wasser zu springen.

Wir hoffen, daß dieses Handbuch Sie möglichst effektiv mit den außerordentlich vielfältigen Möglichkeiten Ihres Computers vertraut macht und daß Sie an seiner Programmierung genausoviel Spaß und Befriedigung haben werden wie der Autor und seine Mitarbeiter.

Neu-Isenburg, im Februar 1981

COMMODORE Büromaschinen GmbH

Hannes Schießl & Josef Steiner

I. DATEN - INTERPRETER - CODES

1. Datenarten

1.1 Einführung

Da Ihr Computer nicht nur rechnen, was 'to compute' eigentlich bedeutet, sondern auch nichtnumerische Zeichenfolgen behandeln kann, unterscheidet er zwischen zwei grundverschiedenen Datenarten, nämlich zwischen numerischen und nichtnumerischen Daten.

- numerische Daten:

Numerische Daten sind ganz einfach Zahlen. Ihr Computer kann mit Zahlen rechnen, also alle arithmetischen und mathematischen Funktionen und Verknüpfungen damit durchführen.

- nichtnumerische Daten:

Dies sind Zeichenfolgen, die für den Computer keinerlei Bedeutung haben, die er aber aufgrund Ihrer Anweisungen zerteilen oder zusammensetzen kann. Diese Zeichenfolgen ('Strings') werden oft nicht ganz exakt als 'Text' bezeichnet, obwohl sie wesentlich mehr enthalten können, als Text.

Zusätzlich zur Datenart ist eine Unterscheidung nötig, wie die Daten dargestellt werden, bzw. wo sie stehen. Hier sind Konstante und Variablen zu trennen.

Die einfachste Form sind **Konstante.** Sowohl Zahlen als auch Strings können als Konstante direkt im Programmtext auftreten.

Was dem Computer aber erst seine Leistungsfähigkeit gibt, sind **Variable.** Variable sind Namen von Speicherplätzen, in denen Zahlen oder Strings stehen können.

Nach dieser kurzen Einführung werden im folgenden alle verwendeten Datenarten und Darstellungsformen definiert und erläutert.

1.2 Numerische Daten

Ihr CBM unterscheidet sechs verschiedene numerische Datenarten, von denen fünf in jeder Form im Programm verwendet werden können.

1.2.1 Gleitkomma-Zahlen

Die Gleitkommadarstellung ist die mächtigste Zahlenart, da sie alle anderen drei Formen numerischer Daten als Sonderfälle enthält. In der allgemeinsten Form besteht eine Gleitkommazahl aus der **Mantisse** mit ihrem Vorzeichen und aus dem **Exponenten** zur Basis 10 mit seinem Vorzeichen.

Beispiel

```
 -1.2345 E -3   =    -0.0012345
  1.2345 E  3   =  1234.5
    .057        =     0.057
 125.03         =   125.03
 -3             =    -3
```

Der darstellbare Zahlenbereich liegt für den **absoluten Betrag** der Zahl zwischen

1.7 E 38 und 2.94 E -39

Die exakten Werte liegen

bei 2 h +127 = 1.70141183 E +38
und 2 h -128 = 2.93873594 E -39

('h' steht für 'Pfeil nach oben', lies 'hoch')

Wird der absolute Betrag **über**schritten, wird **OVERFLOW ERROR** gemeldet, wird er **unter**schritten, erfolgt **keine Fehlermeldung,** aber das Ergebnis ist **Null.**

WERT **größer als** 1.7 E +38 oder WERT **kleiner als** -1.7 E +38
ergibt OVERFLOW ERROR

WERT **kleiner als** 2.9 E -39 oder WERT **größer als** -2.9 E -39
ergibt 0

Die Mantisse wird mit einer Genauigkeit von 9 Dezimalstellen verarbeitet und gespeichert.

(Exakt kann die Mantisse +/- 2 exp 32 = 4294967296 Möglichkeiten darstellen, das sind fast 9.5 Dezimalstellen.)

Positive Vorzeichen müssen nicht geschrieben werden und werden nicht ausgegeben. Der Exponent kann entfallen, wenn man Zahlen hat, die sich durch 9 Stellen darstellen lassen. Wann bei der Ausgabe von der Darstellung ohne Exponent in die normalisierte Exponentialdarstellung (scientific notation) übergegangen wird, ist bei PRINT beschrieben. Mögliche Eingabeformate sind bei INPUT bzw. STR$ beschrieben.

Das 'E' für Exponent muß immer **ohne SHIFT** getippt werden, unabhängig davon, ob dann ein großes oder kleines E auf dem Bildschirm steht.

Beachten Sie bitte, daß der **Punkt** in 1.234 nicht versehentlich zu einem **Komma** wird, ein SYNTAX ERROR bzw. EXTRA IGNORED wäre die Folge.

Alle Zwischenergebnisse in arithmetischen und mathematischen Ausdrücken werden intern im **Gleitkommaformat** ausgedrückt, unabhängig davon, ob das Ergebnis vielleicht einen Integer- oder Bytewert darstellen muß!

Intern wird eine Gleitkommazahl in 5 Bytes dargestellt, wobei 1 Byte den Exponenten und 4 Bytes die Mantisse enthalten.

1.2.2 Integer (ganze Zahl)

Der **Integerbereich** ist definiert von

-32767 bis **+32767**

In einer Integerzahl kann kein Punkt vorkommen, weil nur ganze Zahlen zugelassen sind.

Wird der Integerbereich über- oder unterschritten, so wird ILLEGAL QUANTITY ERROR gemeldet. Dagegen wird kein Fehler gemeldet, falls das Ergebnis eines Ausdrucks Nachkommastellen enthält:

Zahlen mit Nachkommastellen werden in die nächstkleinere Zahl umgewandelt. Bei positiven Zahlen ist dies gleichbedeutend mit dem Abschneiden des Nachkommateils, bei negativen bedeutet es betragsmäßig aufrunden!

```
 123.456   ergibt   123
-123.456   ergibt  -124
```

Intern wird eine Integerzahl in 2 Bytes dargestellt, die in der Reihenfolge HIGH, LOW abgespeichert sind: WERT = 256*HIGH + LOW

1.2.3 Bytezahl

In vielen BASIC-Befehlen kommen Parameter vor, die nur die ganzzahligen Werte von

0 bis 255

annehmen dürfen. Wir bezeichnen sie als **Byte-Werte,** weil durch ein Byte genau dieser Wertebereich dargestellt werden kann.

Auch hier ist ILLEGAL QUANTITY ERROR die Folge, wenn negative Zahlen oder Zahlen größer als 255 verwendet werden. Etwaige Nachkommastellen werden abgeschnitten.

1.2.4 Logische Daten

Das Ergebnis von logischen Ausdrücken wird als **wahr** oder **falsch** bezeichnet, kann also nur zwei verschiedene Werte annehmen. Im Gegensatz zu anderen Sprachen wird in BASIC **falsch** als **0** und **wahr** als **-1** dargestellt. Die Darstellung logischer Werte erfolgt also durch Zahlenwerte!

Während das Ergebnis von logischen Ausdrücken nur die beiden Werte 0 und -1 annehmen kann, bedeutet für IF THEN nur der Wert **0** **falsch** und **jeder andere** Wert **wahr**.

1.2.5 Adressdaten

Bei den Befehlen PEEK, POKE, SYS, USR und WAIT müssen absolute Maschinenadressen angegeben werden. Diese 16-Bit-Adressen können Werte annehmen zwischen

0 und **65535** (2 h 16 - 1)

Verläßt man diesen Bereich, so wird ILLEGAL QUANTITY ERROR gemeldet.

1.2.6 BASIC-Zeilennummern

BASIC Zeilennummern können ganze Zahlen sein zwischen

0 und 63999

Versucht man eine größere Zeilennummer als 63999 oder eine negative einzugeben, wird SYNTAX ERROR gemeldet!

Ein **wesentlicher Unterschied** zwischen BASIC Zeilennummern und den fünf vorher besprochenen Zahlenarten ist zu beachten:

BASIC-Zeilennummern können nicht durch arithmetische Ausdrücke berechnet werden und auch nicht in Variablen stehen. Sie müssen in den entsprechenden BASIC-Befehlen also immer als Konstante angegeben werden.

1.3 Strings

Ein String ist wörtlich übersetzt eine (Zeichen-) Kette. Ein String kann beliebige alphanumerische Zeichen enthalten. Meistens handelt es sich dabei um Text. Allgemeiner formuliert handelt es sich um druckbare Zeichen (-Codes). Allerdings kann der Inhalt einer Stringvariablen auch wesentlich allgemeiner definiert werden, nämlich als Bytefeld, in dem jedes Byte z.B. einen Dezimalwert von 0 bis 255 repräsentieren kann.

Ein sehr wichtiger Unterschied zwischen String- und Zahlendaten ist, daß der Computer den 'Sinn' eines Strings nicht verstehen kann, während er ja den Zahlenwert von Zahlen 'versteht', also damit rechnen kann.

Strings und Zahlen sind also grundverschiedene Datentypen, die nicht verwechselt werden dürfen. Passiert es doch, wird TYPE MISMATCH ERROR gemeldet.

2. Darstellungsweisen von Daten

2.1 Variable

Variable sind Namen für Speicherplätze, in denen sich der Computer Zwischenergebnisse merken kann. Es gibt zwei Unterscheidungskriterien bei Variablen.

Einerseits gibt es Variablen für **Gleitkomma-** und **Integer**zahlen, sowie für **Strings.**

Andererseits gibt es **einfache** Variable, die nur 1 Zahlenwert oder 1 String enthalten können und im Gegensatz dazu **indizierte** Variable oder Variablen**felder,** die mehrere Werte unter dem gleichen Namen enthalten. Die Werte sind dann einfach 'durchnummeriert'.

2.1.1 Variablennamen

Die Regeln für die Variablennamen sind unabhängig vom Variablentyp folgende:

Ein Name besteht aus maximal zwei Zeichen. Das erste Zeichen ist immer ein Buchstabe, das zweite kann entfallen oder ist ein Buchstabe oder eine Ziffer.

Beispiel: AB
X1
C

Um Mißverständnissen vorzubeugen, sei auch hier erwähnt, daß Variablennamen **immer ohne SHIFT** eingegeben werden müssen, auch wenn sie dann im Textmodus als kleine Buchstaben auf dem Bildschirm erscheinen!

Grundsätzlich können Namen auch länger als zwei Zeichen sein, aber erstens werden nur die ersten zwei zur Unterscheidung herangezogen, so daß VAR1 und VAR2 denselben Speicherplatz bezeichnen. Zweitens gibt es Konflikte mit BASIC-Befehlen, wenn Befehlsworte im Namen enthalten sind, z.B. erzeugt 'UNSINN = 7' einen 'SYNTAX ERROR', weil SIN (Sinus) enthalten ist.

Da man bei längeren Namen oft nicht merkt, daß ein Befehl enthalten ist, und es nur Platz und Zeit kostet, aber keinen Gewinn bringt, wird empfohlen, nur Namen mit maximal zwei Zeichen zu verwenden.

Folgende **Variablen** sind für die angegebenen Zwecke reserviert und dürfen mit der Ausnahme TI$ nicht neue Werte zugewiesen bekommen:

DS	Disk Status (Nummer)
DS$	Disk Status (Text)
ST	Peripherie-Status
TI	TIME (Zeit in 1/60 Sekunden)
TI$	TIME (Zeit in Stunden, Minuten, Sekunden)

Die folgenden **Namen** sind für Variable nicht zugelassen (ergibt SYNTAX ERROR)!

FN	Funktion
IF	IF THEN
ON	ON GOTO / GOSUB
OR	ODER
TO	FOR ... TO

2.1.2 Variablentypen

Wie erwähnt, gibt es die drei Variablentypen **Gleitkomma, Integer und String.** Wenn nach dem Namen ein **%** (Prozentzeichen) steht, handelt es sich um eine Integervariable, wenn ein **$** (Dollarzeichen) steht, um eine Stringvariable. Wenn keines dieser beiden Zeichen folgt, ist immer eine Gleitkommavariable gemeint.

Beispiel:	AC	Gleitkomma
	AC%	Integer
	AC$	String

Diese drei Variablen verweisen auf drei verschiedene Speicherplätze, obwohl sie denselben Namen haben!

Eine Gleitkommavariable kann als Inhalt jede Zahl haben, die der Definition der Gleitkommazahl genügt.

Entsprechend kann eine Integervariable nur eine Integerzahl aufnehmen.

Eine Stringvariable kann Strings bis zu einer maximalen Länge von 255 Zeichen (Bytes) aufnehmen. Dabei paßt sich die Variable der jeweils aktuellen Stringlänge automatisch an. Man muß nicht wie in anderen BASIC-Dialekten den benötigten Platz vorher belegen.

Wird die Länge von 255 überschritten, so wird STRING TOO LONG ERROR gemeldet und der entsprechende Befehl nicht ausgeführt.

Eine Stringvariable kann auch gar kein Zeichen enthalten, wir bezeichnen ihren Inhalt dann als den **leeren** String.

2.1.3 Indizierte Variable

Jede der drei Variablenarten kann nicht nur als einfache Variable auftreten, sondern auch als indizierte Variable. Andere Bezeichnungen dafür sind: Matrix, Vektor, Feld.

Gekennzeichnet sind indizierte Variable durch die Indizes, die in Klammern hinter dem Namen folgen.

Beispiel:	AB (3)	eindimensionales Gleitkommafeld
	AB% (3,7)	zweidimensionales Integerfeld
	AB$ (I)	eindimensionales Stringfeld

Einzelheiten darüber sind bei DIM zu finden. Die Anzahl der Dimensionen ist nur durch den Platz in der BASIC Zeile begrenzt, die Anzahl der Elemente pro Dimension und insgesamt nur durch den Platz im Arbeitsspeicher.

2.1.4 Platzbedarf von Variablen

Jede einfache Variable belegt unabhängig vom Typ **7 Bytes** im Arbeitsspeicher. Bei den Zahlenvariablen (Gleitkomma und Integer) ist darin schon der Zahlenwert enthalten. Bei Strings hingegen kommen noch die Anzahl der Zeichen hinzu, die die Variable gerade enthält, sowie 2 Bytes zusätzlicher Verwaltungsbedarf für jeden nicht leeren String.

Bei Feldern hängt der Platzbedarf vom Typ ab:

Typ	Bytes pro Element
Integer	2
Gleitkomma	5
String	3 + 2 + Stringinhalt

Für die Verwaltung der Felder sind zwar auch einige Bytes nötig, aber nur einmal pro Feld, so daß man diesen Bedarf gegenüber dem Bedarf für jedes Element vernachlässigen kann.

Variable können jederzeit eingeführt, aber nicht mehr einzeln gelöscht werden. Vor allem bei Feldern muß diese Eigenschaft schon während der Planung der Datenstruktur berücksichtigt werden.

2.1.5 Zugriffszeiten auf Variable

Einfache Variable werden in einer Variablentabelle abgelegt. Jede angesprochene Variable, die noch nicht in der Tabelle steht, wird am Tabellenende angehängt. Da Variablen bei jedem Aufruf vom Tabellenanfang her gesucht werden, erfolgt der Zugriff umso schneller, je früher (zeitlich) die Variable im Programm angesprochen wurde.

Die Tabelle der Feldvariablen kommt unmittelbar anschließend an die einfachen Variablen. Sind bereits Felder definiert wenn noch zusätzliche einfache Variable im Programm auftauchen, so müssen alle Felder mit ihrem Inhalt nach hinten verschoben werden. Der Zeitbedarf dafür kann bei großen Feldern beträchtlich sein. Bei Zeitproblemen sollten also große Felder erst dimensioniert werden, nachdem alle einfachen Variablen im Programm angesprochen wurden.

2.2 Konstante

Sowohl Zahlen als auch Strings können als Konstante direkt im Programmtext auftauchen.

Beispiel:
```
A = 3
B%= 15
A$= "TEXT"
D$= ""
```

Stringkonstante sind in der Regel durch Anführungszeichen einzurahmen. Eine spezielle Stringkonstante sind zwei unmittelbar aufeinanderfolgende Anführungszeichen. Sie definieren den leeren String.

Bitte beachten Sie folgenden Unterschied zwischen Zahlen- und Stringkonstanten: Während jeder mögliche Zahlenwert als Konstante ausgedrückt werden kann, sind bei Stringkonstanten einige Einschränkungen zu beachten:

Länge

In einer String-**Variablen** können maximal 255 Zeichen stehen, aufgrund der Beschränkungen der Bildschirmverwaltung können aber in einer String-**Konstanten** nur maximal ca. 70 Zeichen stehen.

Codes

Nur die 128 ASC-Codes, die die nicht-reversen druckbaren Zeichen darstellen (32-95 und 160-223) und einige Steuerzeichen (0-31 und 128-159) können in Stringkonstanten dargestellt werden.

3. Umwandlungen zwischen Datenarten

Durch verschiedene Operationen sind Umwandlungen von einem Datentyp in einen anderen möglich.

Die Umwandlungsbefehle zwischen Strings und Zahlen sind

ASC / CHR$
VAL / STR$

Sie sind unter diesen Namen beschrieben.

Hier soll nur auf die Umwandlung zwischen Gleitkomma- und Integerzahlen eingegangen werden:

3.1 Integer in Gleitkomma

Die Umwandlung von Integer in Gleitkomma ist **immer möglich** und hat auf den Wert keine Auswirkungen.

Beispiel: A = A%

3.2 Gleitkomma in Integer

Die Umwandlung von Gleitkomma in Integer ist nur möglich, wenn die Gleitkommazahl im Integer-Bereich liegt:

```
Beispiel:   10 A = 20156
            20 A%= A
```

Ist dies nicht der Fall, wird ILLEGAL QUANTITY ERROR gemeldet.

```
Beispiel:   10 A = 50000
            20 A%= A
            RUN ergibt ILLEGAL QUANTITY ERROR IN 20
```

Wenn der (positive) Gleitkommawert Nachkommastellen enthält, werden diese **abgeschnitten,** es wird also **nicht gerundet:**

```
Beispiel:   10 A = 123.256
            20 A%= A
            30 PRINT A%
            RUN ergibt 123
```

Exakter ausgedrückt, wird die **nächstkleinere ganze Zahl** gebildet. Dies ist bei negativen gebrochenen Zahlen zu beachten:

Beispiel:
```
10 A = -123.256
20 A%= A
30 PRINT A%
RUN ergibt -124
```

3.3 Rundungsfehler und Integerzahlen

(s. dazu auch den Abschnitt über numerische Ausdrücke)

Jede arithmetische und mathematische **Operation wird auf Gleitkommabasis** ausgeführt, es existiert also kein gesondertes Integer-Arithmetik-Paket!

Viele BASIC-Anweisungen haben Parameter, die keine gebrochenen Zahlen sein dürfen. Diese ganzzahligen Parameter können Integer- oder Byteparameter sein. Da aber in BASIC jeder Parameter als beliebig komplexer Ausdruck auftreten kann, und Zahlenausdrücke auf Gleitkommobasis ausgewertet werden, wird das Ergebnis solcher Ausdrücke automatisch in Ganzzahl-Format umgewandelt. Da nun grundsätzlich nicht auszuschließen ist, daß in umfangreichen Gleitkommaoperationen sich die zwangsläufigen Rundungsfehler aufgrund der endlichen Mantisse bis in die neunte Stelle schleppen, können intern Ergebnisse wie

123.999999 anstatt 124

auftreten. Beim Abschneiden auf Integerformat wird aus einem Fehler von 1E-6, der zu vernachlässigen wäre, ein Fehler von 1, der einen Algorithmus 'zu Fall bringen kann'.

Am Beispiel des Indexes von Feldvariablen können Sie dies leicht einsehen.

Sobald also ein Algorithmus bei Zwischenergebnissen die ganze Genauigkeit der Mantisse von 9 Stellen benötigt, muß an geeigneten Stellen durch Rundung (s. INT) dafür gesorgt werden, daß sich Fehler nicht fortpflanzen können!

In den meisten Fällen wird dies allerdings nicht nötig sein, weil Algorithmen, die ein maximal fünfstelliges Ergebnis liefern, selten mit neunstelligen Zwischenergebnissen arbeiten.

4. Ausdrücke und Operatoren

Bis jetzt sind die verschiedenen Datenarten (Zahlen, Strings) vorgestellt worden, sowie ihre Erscheinungsformen (Konstante, Variable). Um bei den BASIC Befehlen möglichst einfach die einzelnen Parameter beschreiben zu können, wollen wir den Begriff des Ausdrucks einführen.

Ein Ausdruck kann bestehen aus Konstanten, Variablen, Verknüpfungsoperatoren und Funktionen.

Beispiel:	SIN(7*Z/5)	numerischer Ausdruck
	"abc"+A$+LEFT$(Z$,3)	Stringausdruck
	A 3 AND B 5	logischer Ausdruck

Wir unterscheiden folgende Ausdrücke nach ihren Ergebnissen:

numerisch	Gleitkomma-Ausdrücke
	Integer-Ausdrücke
	Byte-Ausdrücke
logisch	logische Ausdrücke
String	String-Ausdrücke

Entsprechend den Datendefinitionen ergibt sich aus einem Gleitkomma-Ausdruck ein Gleitkomma-Ergebnis usw.

Äußerlich unterscheiden sich die drei numerischen Ausdrücke nicht. Dagegen unterscheiden sich numerische, logische und Stringausdrücke sehr stark.

Allen Ausdrücken ist gemeinsam, daß sie sehr komplexe Formen annehmen können. Allgemein kann ein Ausdruck so definiert werden:

Ausdruck := Ausdruck1 Operator Ausdruck2

Ein Ausdruck kann also seinerseits wieder aus Ausdrücken zusammengesetzt sein.

Die einfachste Form eines Ausdrucks ist nur eine Konstante oder Variable oder Funktion:

Ausdruck1:= 5
Ausdruck2:= AC
Ausdruck3:= SIN(5)

Die nächstkomplexere Form ist die Verknüpfung zweier Konstanten oder Variablen oder Funktionen

Ausdruck1:= 5+A
Ausdruck2:= AC*BC
Ausdruck3:= 2*SIN(A)

Im einfachsten Fall kann ein Ausdruck also aus nur einer Konstanten bestehen, im komplexesten aus mehreren geschachtelten Ausdrücken. Die Schachtelungstiefe hängt auch von der Anzahl der GOSUBs und FOR NEXT Schleifen ab (s. Stack).

Bei der Behandlung der einzelnen BASIC Befehle werden die Parameter nur durch die Art des Ausdrucks definiert. Es wird nicht jedesmal erwähnt, daß ein Ausdruck beliebig einfach (Konstante) oder sehr komplex sein kann. Anders ausgedrückt dürfen nur BASIC Zeilennummern nur als Konstante auftreten (z.B. LIST 10 - 20), während alle anderen Parameter von Befehlen allgemeine Ausdrücke des jeweiligen Typs sind!

4.1 Numerische Ausdrücke und Operatoren

Numerische Operatoren

Operation	Operator	Beispiel
Potenz	h (Pfeil n. oben)	X h Y
Negation (unäres Minus)	-	-X , A*(-3)
Multiplikation	*	X*Y
Division	/	X/Y
Addition	+	X+Y
Subtraktion	-	X-Y

Entsprechend der mathematischen Definition bindet die Potenzierung stärker als die Multiplikation/Division und diese stärker als Addition/Subtraktion. Innerhalb der gleichen Hirarchiestufe werden Ausdrücke von links nach rechts ausgewertet. Durch Klammern können die Standard-Hierarchien durchbrochen werden.

Beispiel:

Ausdruck	Zwischen-Ergebn.		Erläuterung
2+3 *5	= 2+15	= 17	'Punkt vor Strich'
(2+3)*5	= 5*5	= 25	Klammern binden am stärksten
2 h 3*2	= 8*2	= 16	Exponentiation bindet stärker als Multiplikation
2 h (3*2)	= 2h6	= 64	Klammern binden stärker als Exponentiation
2 h -3	= .125		2h-3 = 1/(2h3)
2 h (-3)	= .125		überflüssige Klammern schaden nicht
-2 h 3	= -8		-2h3 = (-2)*(-2)*(-2)
-2 h -3	= -.125		-2h-3= (-0.5)*(-0.5)*(-0.5)
-2*3	= -6		minus mal plus ergibt minus
-2*-3	= 6		minus mal minus ergibt plus
--2*3	= 6		--2 = (-1)*(-2)

Maximal kann ein Ausdruck 9 Klammerebenen enthalten. Wird diese Anzahl überschritten, wird OUT OF MEMORY ERROR gemeldet.

Näheres ist unter **Stack** beschrieben. Abhilfe kann leicht geschaffen werden, indem der Ausdruck in mehrere Teilausdrücke zerlegt wird, deren Ergebnisse in mehreren Variablen zwischengespeichert werden. Im nächsten Schritt werden dann diese Variablen in der gewünschten Weise verknüpft. Natürlich kann diese Unterteilung in beliebig viele Teilschritte erfolgen. Dieser Methode sind keine Grenzen gesetzt:

Beispiel

```
A = (B+C) * (D+E)
kann ersetzt werden durch:
Z1= B+C
Z2= D+E
A = Z1 * Z2
```

Die Fehlermeldungen OVERFLOW ERROR und ILLEGAL QUANTITY ERROR sind bei den Datenarten beschrieben. Wird versucht, durch 0 zu dividieren, so wird DIVISION BY ZERO ERROR gemeldet.

4.2 Logische Ausdrücke und Operatoren

Der Computer kann zwei numerische oder String-Ausdrücke vergleichen, um festzustellen, ob sie gleich bzw. ungleich sind, oder ob einer größer oder kleiner als der andere ist.

Mehrere solcher Entscheidungsergebnisse kann er durch die logischen Operatoren NOT, AND und OR verknüpfen. Auf diese Weise können sehr komplexe Entscheidungen getroffen werden, die mit Hilfe von

IF (logischer Ausdruck) THEN

den Programmablauf beeinflussen können. Dies ist bei IF THEN beschrieben.

Darüber hinaus kann aber das Ergebnis einer logischen Entscheidung auch in Variablen zur späteren Verwendung aufbewahrt werden.

Ehe aber diese Feinheiten betrachtet werden, sollen die 3 Vergleichs-Operatoren mit ihren 6 Kombinationsmöglichkeiten vorgestellt werden:

Das Symbol für **kleiner als** ist **spitze Klammer auf,**
und für **größer als** ist **spitze Klammer zu.**

Aus technischen Gründen können diese Symbole hier im Text nicht dargestellt werden, und werden deshalb durch k (kleiner als) bzw. g (größer als) ersetzt.

4.2.1 Vergleichsoperatoren

Operator	**Bedeutung**	**Beispiel**	**Bedeutung:** ist der Inhalt der Variablen A
=	gleich	A = 3	identisch mit dem Wert 3
g	größer als	A g 3	größer als 3
k	kleiner als	A k 3	kleiner als 3
=k oder k=	kleiner/gleich	A =k 3	kleiner oder gleich 3
=g oder g=	größer/gleich	A g= 3	größer oder gleich 3
gk oder kg	ungleich	A gk 3	ungleich 3 (größer oder kleiner als 3)

Bei Zahlen ist die Bedeutung dieser Entscheidung offensichtlich. Die Anwendung auf Strings bedarf aber einer Erklärung:

4.2.2 Vergleich von Strings

'=' stellt fest, ob die Ergebnisse beider Stringausdrücke identisch sind, ob die Strings also in allen Zeichen übereinstimmen.

'gk' stellt fest, ob die Strings nicht in allen Zeichen übereinstimmen. Anders ausgedrückt, wird hier mit 'wahr' geantwortet, wenn mindestens ein Zeichen verschieden ist.

'g' bzw. 'k' prüft die alphabetische Reihenfolge. Ist ein String 'kleiner' als der andere, so steht er im Alphabet weiter vorne.

Einige Beispiele sollen diesen leistungsfähigen Vergleich verdeutlichen. Die angegebenen Vergleiche erzeugen eine **wahre** Aussage:

1) "ABC" = "ABC"
2) "ABC" gk "ABX"
3) "ABC" k "BCD"
4) "ABC" k "BC"
5) "ABC" g "AB"

Allgemein wird der ASC Code der 1., 2., 3. usw. Zeichen beider Strings verglichen:

Solange die Codes identisch sind, werden die Strings als gleich bezeichnet (1).

Wird an einer Stelle ein Unterschied festgestellt, wird abgebrochen und das Ergebnis lautet 'ungleich' (2).

Falls kleiner oder größer gefragt war, wird ebenfalls anhand der **ersten** Ungleichheit entschieden, was zutrifft (3).

Grundsätzlich wird also nur bis zur ersten Ungleichheit verglichen! Ein Sonderfall wird durch diese Definition ebenfalls erklärt: Wenn ein String zu Ende ist, aber beide so weit identisch waren, so ist der kürzere der kleinere, weil das leere Stringelement (für das es keinen ASC Code gibt!) als kleiner definiert ist, als jeder andere ASC Code! (5)

Wie Beispiel (4) zeigt, ist nicht die Länge der Strings ausschlaggebend. Ein längerer String kann im Alphabet durchaus weiter vorne stehen, also 'kleiner' sein als ein kürzerer!

4.2.3 Wahr und falsch

Wenn nur innerhalb von IF ... THEN Entscheidungen getroffen werden, genügt es zu wissen, daß 'wahr' den Programmlauf hinter THEN fortsetzt, 'falsch' dagegen in der nächsten BASIC Zeile (s. IF THEN).

Für kompliziertere Entscheidungen ist es hilfreich zu wissen, wie der Computer die Werte 'wahr' und 'falsch' darstellt. Im Gegensatz zu anderen Computersprachen kennt BASIC keine speziellen logischen (boolean) Werte und Variable, sondern stellt diese beiden Werte durch Integer-Zahlen dar:

Das Ergebnis eines logischen Ausdrucks ist

```
 0  für   'falsch'   (false / nein)
```

und

```
-1  für    'wahr'   ( true / ja    )
```

Beispiel:

```
?    5 g 3    Fünf ist größer als 3 ist eine wahre Aussage.
-1

?    3 g 5    Drei ist größer als 5 ist eine falsche Aussage.
0

A = 5 g 3     Das Ergebnis des logischen Ausdrucks kann in eine
? A           Zahlenvariable übernommen werden.
-1
```

Intern wird ein logischer Wert wie jede Integer Zahl als 16-Bit-Zahl dargestellt. Bei '-1' sind alle Bits gesetzt, bei '0' entsprechend alle gelöscht.

4.2.4 Verknüpfung von logischen Ausdrücken durch NOT, AND und OR

NOT, AND und OR führen im Grunde bitweise logische Verknüpfungen mit zwei 16-Bit-Zahlen durch. Dies ist aber nicht wichtig zu wissen, wenn man nur logische Ausdrücke verknüpfen will. Für diesen Fall kann man diese drei Operatoren leicht verständlich definieren:

Werden zwei logische Ausdrücke durch AND verknüpft, so ist das Ergebnis genau dann 'wahr' (-1), wenn **beide** Ausdrücke 'wahr' sind.

Bei OR ist das Ergebnis schon dann 'wahr', wenn **wenigstens einer** der beiden Ausdrücke 'wahr' ist.

NOT verknüpft nicht zwei Ausdrücke, sondern ist ähnlich wie das arithmetische Vorzeichen (unäres Minus) einem logischen Ausdruck vorangestellt und kehrt seinen Wert ins Gegenteil um.

In der folgenden Tabelle sind alle Kombinationsmöglichkeiten nochmal zusammengestellt.

Verknüpfung			**Ergebnis**
falsch	AND	falsch	falsch
falsch	AND	wahr	falsch
wahr	AND	falsch	falsch
wahr	AND	wahr	wahr
falsch	OR	falsch	falsch
falsch	OR	wahr	wahr
wahr	OR	falsch	wahr
wahr	OR	wahr	wahr
NOT wahr			falsch
NOT falsch			wahr

Ergänzende Informationen zu den logischen Ausdrücken sind bei IF THEN, NOT, AND, OR und WAIT zu finden.

4.2.5 Hierarchie

Bei der Auswertung von logischen Ausdrücken bzw. von Zahlenausdrücken gelten gewisse Hierarchiestufen oder Bindungsregeln. In der folgenden Tabelle sind nochmal alle Operatoren geordnet von der stärksten zur schwächsten Bindung aufgeführt:

Mathematische Operatoren

```
Funktionen   (SIN, FN, EXP usw)
-            (Vorzeichen, unäres Minus)
h            (Exponentiation)
* /
+ -
```

Vergleichsoperatoren

```
= g k        (alle Vergleichsoperatoren sind gleichberechtigt)
```

Logische Operatoren

```
NOT
AND
OR
```

Diese Hierarchien werden vom Computer sehr konsequent eingehalten. Da solche Strukturen erfahrungsgemäß Vorstellungsschwierigkeiten bereiten, und deshalb zu Programmierfehlern führen, sollten Sie folgenden Tip beherzigen: Solange Ihnen diese Regeln nicht wirklich geläufig sind, spendieren Sie lieber einige (überflüssige) Klammern, können aber dann sicher sein, daß der Rechner das tut, was Sie eigentlich wollen. Einige Beispiele sollen die Problematik verdeutlichen:

4.2.6 Beispiele zu logischen Operationen

Automatische Hierarchie	**Durch Klammern verdeutlicht/erzwungen**	
A=B AND C=D	(A=B) AND (C=D)	(1)
A=B OR C=D AND E=F	(A=B) OR((C=D) AND(E=F))	(2)
-	((A=B) OR (C=D))AND(E=F)	
A = B AND C	(A= B)AND C	(3)
-	A=(B AND C)	

Fall (1) ist problemlos. In dieser Entscheidung wird lediglich gefragt, ob gleichzeitig A=B und C=D ist. Wenn man die jeweils zwei denkbaren Fälle (gleich oder nicht gleich) gegenüber stellt, ergeben sich insgesamt 4 Kombinationsmöglichkeiten. In der folgenden Art kann man diese Möglichkeiten sehr übersichtlich darstellen.

```
1100 A=B
1010 C=D
1000 AND
```

Die beiden Fälle wahr und falsch sind hier durch die Werte 1 und 0 repräsentiert. Wenn man sich zu der 1 noch das Minus denkt (-1) stellen die Tabellen gleich die internen Werte im Computer dar.

Etwas komplizierter werden die Überlegungen zum Fall 2. Je nach Klammersetzung werden die Operationen in verschiedener Reihenfolge durchgeführt. Wie die Tabellen zeigen, führt dies zu verschiedenen Ergebnissen:

A=B OR (C=D AND E=F)		(A=B OR C=D) AND E=F	
11001100	C=D	11110000	A=B
10101010	E=F	11001100	C=D
10001000	AND	11111100	OR
11110000	A=B	10101010	E=F
11111000	OR	10101000	AND

An diesem Beispiel sehen Sie schon sehr deutlich, daß eine unüberlegt gesetzte (oder weggelassene) Klammer das Ergebnis grundlegend verändern kann.

Das Beispiel 3 zeigt Ihnen, daß der Computer sogar mit Ausdrücken etwas anfangen kann, die mit der 'Alltagslogik' nicht mehr zu verstehen sind. Im Sinne der Boolschen Algebra sind diese Ausdrücke aber exakt definiert und deshalb hat der Computer damit keine Schwierigkeiten:

(A=B) AND C		A = (B AND C)	
11110000	A	11001100	B
11001100	B	10101010	C
11000011	=	10001000	AND
10101010	C	11110000	A
10000010	AND	10000111	=

4.3 String-Ausdrücke und -Operatoren

Stringausdrücke sind wesentlich einfacher aufgebaut, als numerische und logische Ausdrücke, da sie nur einen Operator kennen, also ohne Hierarchieregeln einfach von links nach rechts abgearbeitet werden.

Der Operator für Stringausdrücke ist + und bewirkt die Aneinanderreihung von Teilstrings von links nach rechts. Diese Reihenfolge kann auch durch Klammern nicht geändert werden.

Eine häufige Fehlermeldung bei Stringausdrücken ist STRING TOO LONG ERROR, wenn der Ergebnis-String oder ein Zwischenergebnis-String länger als 255 Zeichen werden sollte.

Weiterhin kann es zum TYPE MISMATCH ERROR kommen, wenn anstatt eines Teilstrings ein Zahlenausdruck im Stringausdruck steht.

Beispiele von Stringausdrücken:

```
10 A$="abc"+"123"
20 ?A$
abc123
```

```
10 A$="abc":B$="123":A$=B$+A$+"def"
20 ?A$
123abcdef
```

Beachten Sie die Reihenfolge der Aneinanderreihung!

```
10 A$="abc"+123
TYPE MISMATCH ERROR IN 10
```

Weil 123 nicht in Anführungszeichen eingeschlossen ist, wird es als Zahlen-Konstante und nicht als String-Konstante interpretiert.

```
10 FOR I = 0 TO 255
20 A$ = A$ + "*"
30 NEXT
STRING TOO LONG ERROR IN 20
```

A$ soll mit '*' aufgefüllt werden. Die Schleife durchläuft aber 256 Werte und will deshalb 256 Zeichen in den String schreiben. Anfangswert 1 statt 0 würde z.B. funktionieren.

Weitere Beispiele sind bei den Befehlen LEFT$, RIGHT$ und MID$ zu finden.

5. Interpreter und Organisation

Das BASIC Programm im Arbeitsspeicher wird Zeichen für Zeichen vom Interpreter abgetastet und ausgeführt. Dabei muß der Interpreter im ersten Schritt jeden Befehl erst einmal analysieren. Erst nachdem dies geschehen ist, kann der Befehl ausgeführt werden.

Der Interpreter ist selbst ein Programm, allerdings in Maschinensprache geschrieben. Dieses Programm ist in ROMs (nichtlöschbare Speicherbausteine) eingespeichert und deshalb sofort beim Einschalten verfügbar.

Der Vorteil der Interpretersprache liegt darin, daß man relativ einfach im 'Dialog' mit dem Rechner Programme erstellen kann. Allerdings wird dies durch den Nachteil erkauft, daß Interpreterprogramme relativ langsam ablaufen.

5.1 Geschwindigkeit

Bei manchen Programmteilen wäre es wünschenswert, daß sie schneller laufen. Es ist nicht sinnvoll, exakte Laufzeiten von einzelnen Statements anzugeben, da sie oft von vielen Parametern abhängen. Trotzdem ist es oft hilfreich, wenigstens die Größenordnung von Befehlsausführungszeiten zu kennen: Zwischen

1 ms und 100 ms

können einfache BASIC Statements zur Ausführung brauchen. Sehr umfangreiche Ausdrücke können auch länger als 100 ms dauern.

Es können also sicher nicht mehr als 1000 BASIC Befehle pro Sekunde ablaufen. Als grobe Näherung ist der Wert 100 Befehle pro Sekunde ganz nützlich.

Soweit es sinnvoll ist, werden bei einzelnen Befehlen qualitative Hinweise zur Ausführungszeit gegeben.

Hier sollen einige generelle Hinweise gegeben werden, die sich nicht auf spezielle Befehle beziehen:

Mehrere Befehle pro Zeile

Commodore-BASIC bietet im Gegensatz zu vielen anderen BASIC-Dialekten die Möglichkeit, mehrere Anweisungen durch Doppelpunkt getrennt in eine Zeile zu schreiben. Diese Möglichkeit sollte aus zwei Gründen ausgenützt werden: Erstens dauert der Übergang zu einer neuen Zeile länger, als der Übergang zu einem neuen Statement und zweitens steigt die Zeit für Sprünge mit der Anzahl der Zeilen. Außerdem braucht eine neue Zeile 5 Bytes an Verwaltung, während der Trenn-Doppelpunkt zwischen zwei Befehlen nur 1 Byte benötigt. Pro weiterem Statement in einer Zeile spart man also vier Bytes ein.

Keine überflüssigen SPACEs

Wenn Zeit und Platz knapp sind, sollte nicht die Lesbarkeit des Listings durch Spaces erhöht werden, weil jedes Space 1 Byte benötigt und vom Interpreter gelesen werden muß.

Kurze Variablennamen

In zeitkritischen Schleifen sollten möglichst nur einbuchstabige Variablen verwendet werden. Das bedeutet, daß auch keine Integervariablen verwendet werden sollten, da das '%' auch gelesen werden muß. Außerdem muß erst das Integerformat in Gleitkomma umgewandelt werden, was zusätzlich Zeit kostet.

5.2 Spaces im BASIC Text

Prinzipiell brauchen im BASIC Text keine Spaces als Trennzeichen stehen. Es gibt allerdings einige Ausnahmefälle, wo durch Spaces eindeutige Verhältnisse geschaffen werden müssen:

Beispiel:

```
10 IFSTAND64THEN... (IF ST AND 64 THEN)
20 IF64ANDSTTHEN... (IF 64 AND ST THEN)
30 IFSTOR64THEN.... (IF ST OR 64 THEN)
```

Zeile 10 meldet SYNTAX ERROR, Zeile 20 nicht. Eigentlich müßten beide Zeilen völlig identische Wirkung haben, da die beiden Operanden um AND vertauschbar sein müssen.

Des Rätsels Lösung: Der Interpreter findet nach dem S den Befehl TAN (Tangens), aber dahinter keine Klammer. Und schon kennt er sich nicht mehr aus und bringt die Fehlermeldung. In Zeile 20 hat er keine Möglichkeit zu Fehlinterpretationen In Zeile 30 findet er nach S den Befehl TO (FOR..TO..NEXT). Ein Space nach ST in 10 und 30 behebt den Fehler.

Sollte man sich also mit einem hartnäckigen SYNTAX ERROR herumschlagen, kann eine derartige Mißdeutung die Ursache sein!

5.3 BASIC-Zeilenformat und Platzbedarf

Jede BASIC-Zeile beginnt mit einer Zeilennummer und kann, durch Doppelpunkt getrennt, mehrere Befehle aufnehmen.

Jedes BASIC-Befehlswort wird durch einen 1-Byte-Code komprimiert im Speicher gehalten. Jede andere Information innerhalb der Zeile braucht soviele Bytes, wie sie Zeichen umfaßt.

Jede BASIC-Zeile benötigt 5 Bytes für ihre Verwaltung. Im einzelnen sind das 2 Bytes für die Zeilennummer, 2 Bytes für einen Vorwärtszeiger auf die nächste BASIC-Zeile und der Code 0 als Abschluß der Zeile.

Durch die Beschränkung der Bildschirmeingabe kann eine BASIC-Zeile maximal ca. 80 Zeichen lang sein. Allerdings könnte der Interpreter BASIC-Zeilen bis zu einer internen (Interpretercode) Länge von 250 Bytes verarbeiten.

Es ist möglich, durch Verwendung der Befehlsabkürzungen mehr Information in die Zeile zu bringen, als die 80 Zeichen-Zeile normalerweise gestattet, aber dies sollte man nur in Ausnahmefällen nutzen, da LIST die betreffende Zeile ja nicht mit den Abkürzungen druckt, sondern durch Verwendung der vollen Befehlsworte die Zeile über mehr als 80 Zeilen auslistet. Zum Editieren müssen dann von Hand wieder alle Abkürzungen eingegeben werden, um die Zeile geändert wieder in der vollen Länge abspeichern zu können.

5.4 Speichereinteilung

Der freiprogrammierbare BASIC-Arbeitsbereich des Rechners (34 K) gliedert sich in folgende Teile:

Zero Page
-
Stack
-
Page 2,3
-
BASIC-Text (Programm)
-
Einfache Variable
-
Indizierte Variable (Felder)
-
Strings
-
Bildschirm

Die Zero Page ist ein 256 Bytes langer Bereich, auf den der Prozessor extrem schnell zugreifen kann. Alle besonders häufig benötigten Daten legt das Betriebssystem und der Interpreter dort ab.

Der Stack (Stapel) wird im nächsten Abschnitt mit seinen Konsequenzen für BASIC beschrieben.

Die folgenden 512 Bytes (Page 2,3) enthalten unter anderem die beiden Kassettenpuffer mit zweimal 191 Bytes Länge, den BASIC-Eingabepuffer mit 80 Zeichen Länge, sowie den Tastaturpuffer mit 10 Zeichen Länge.

Ab dem Byte 1024 steht der Text des BASIC-Programms im komprimierten Interpreter-Code.

Sobald ein Programm (mit RUN) gestartet wird, wird automatisch der Anfang der Variablentabelle auf das Ende des BASIC Programms gesetzt. Das ist der Grund, warum andererseits nach einer Änderung im Programm die Variablen 'weg' sind.

Das Ende der Variablentabelle ist gleichzeitig der Anfang der Felder-Tabelle. Dies hat zur Folge, daß der gesamte Feldbereich nach hinten verschoben werden muß, wenn eine neue einfache Variable eingeführt wird.

Wichtig zu wissen ist, daß zwar jederzeit neue einfache und indizierte Variable eingeführt werden können, daß aber nicht einzelne Variable oder Felder gelöscht werden können!

Die Stringinhalte 'wachsen' von der oberen Speichergrenze (32K) her nach 'unten'.

Der Speicher wird also optimal den jeweiligen Gegebenheiten angepaßt. Trotzdem kann er natürlich irgendwann zu klein sein. Dies wird dann durch OUT OF MEMORY ERROR gemeldet. Bei folgenden Gelegenheiten kann diese Meldung auftreten:

Tritt sie bei der Programmeingabe auf, so ist das Programm alleine schon zu groß, könnte also nie laufen, da zu diesem Zeitpunkt noch keine Variable im Speicher steht. In diesem Fall kann nur 'Overlay' (LOAD) helfen.

Tritt sie nach dem Start des Programms auf, ist die häufigste Ursache eine Dimensionierung von Feldern (DIM), da hiermit auf einmal sehr viel Speicherplatz belegt werden kann. Man kann leicht nachprüfen, ob dies der Fall ist: Erstens muß in der gemeldeten Zeile ein DIM stehen und zweitens muß der Platzbedarf des Feldes (s. indizierte Variable) größer sein, als der noch freie Speicherplatz (FRE).

Scheidet DIM aus, kann es ein String sein, da Strings immerhin auch auf einmal 255 Bytes belegen können. In diesem Fall muß in der gemeldeten Zeile eine Stringzuweisung stehen.

Ist auch das nicht der Fall, könnte theoretisch eine neu eingeführte einfache Variable den Speicher gerade um 7 Bytes zu klein gefunden haben.

Wahrscheinlicher ist aber, daß der Stack 'übergelaufen' ist:

5.5 Stack (Stapel)

Der Stack ist ähnlich wie die Zero Page ein bevorzugter Speicherbereich des Prozessors. Der Stack eignet sich sehr gut zur Aufnahme von hierarchischen Strukturen, wie sie die Daten für Unterprogramme (GOSUB/RETURN), Schleifen (FOR NEXT) und Klammerebenen in Ausdrücken darstellen.

Deshalb legt der Interpreter die nötigen Verwaltungsinformationen für Unterprogramme, Schleifen und Ausdrücke im Stack ab. Dies hat den Vorteil, daß der Rücksprung aus Unterprogrammen (RETURN), der Rücksprung in Schleifen, sowie die Überprüfung des Endwertes und die Berechnung von Ausdrücken in sehr kurzer Zeit durchgeführt werden kann. Andererseits ist der Stack ein endlicher Speicherbereich, der relativ schnell 'überläuft'.

Wie schon erwähnt, kann auch dies der Grund für OUT OF MEMORY ERROR sein. Die folgenden Werte sollen Ihnen einen Anhaltspunkt geben, wieviel Ebenen jeweils möglich sind.

GOSUB	26
FOR NEXT	10
Klammerebenen	10

Diese Werte sind exklusiv, d.h. wenn Sie z.B. 26 GOSUB Ebenen haben, kann keine Schleife und kein Ausdruck mehr existieren, ohne daß OUT OF MEMORY gemeldet wird.

Sollten Sie OUT OF MEMORY erhalten, helfen Ihnen vielleicht folgende Überlegungen weiter:

Eine Ausdrucksebene benötigt im Stack genausoviel Platz, wie eine FOR NEXT Schleife (18 Bytes), eine GOSUB Ebene dagegen nur 5 Bytes. Sie können also durch Verzicht auf eine Ausdrucksebene eine Schleife gewinnen oder 3 GOSUB Ebenen gegen eine Schleife eintauschen.

Beispiele für die Stackgrenzen:

```
200 A=2*(2*(2*(2*(2*(2*(2*(2*(2*(2*2)))))))))

301 GOSUB 302:RETURN
302 GOSUB 303:RETURN
.
.
.
325 GOSUB 326:RETURN
326 GOSUB 327:END
327 RETURN

401 FOR A = 0 TO 1
402 FOR B = 0 TO 1
.
.
.
409 FOR I = 0 TO 1
410 FOR J = 0 TO 1
420 PRINT A;B;C;D;E;F;G;H;I;J
430 NEXT  J,I,H,G,F,E,D,C,B,A
```

Die Beispiele in 200, 300 ..., und 400 ... zeigen die jeweils maximal mögliche Schachtelungstiefe, nämlich 10, 26 und 10. Es sei nochmal betont, daß die drei Beispiele nicht kombiniert werden können, weil jedes für sich den Stack vollkommen in Anspruch nimmt.

Wenn Sie jeweils eine Ebene mehr anfügen, wird OUT OF MEMORY IN ... gemeldet.

Lösungsvorschläge

Am leichtesten kann die Schachtelungstiefe von Ausdrücken vermindert werden, da man sie nur in Teilausdrücke zu zerlegen braucht, deren Ergebnisse schrittweise zusammengefaßt werden (s. numerische Ausdrücke).

GOSUB-Ebenen kann man simulieren, indem das Unterprogramm statt mit GOSUB mit GOTO angesprungen wird. Anstatt des RETURN muß dann am Ende des Unterprogramms ein ON .. GOTO Verteiler dafür sorgen, daß das Unterprogramm wieder zum aufrufenden Programm zurückspringt. Das aufrufende Programm muß dem Unterprogramm dann lediglich in einer zusätzlichen Variable mitteilen, wer es gerufen hat (s. ON GOTO).

FOR NEXT Schleifen sind sehr leicht durch IF THEN usw. zu ersetzen.

Natürlich sollte man zu solchen 'Krücken' erst greifen, wenn es wirklich nötig ist, weil sie die Programmstruktur verschlechtern und langsamer sind.

6. Codes

Ihr Computer kann 256 verschiedene Zeichen auf dem Bildschirm darstellen und kennt 256 verschiedene String-Codes. Der Code der Zeichen im Bildschirm ist allerdings nicht identisch mit dem Code im String. Wir unterscheiden deshalb den Bildschirmcode **BSC** und CBM-ASCII Code **ASC.** Zusätzlich soll noch der ASCII Code **ASCII** besprochen werden, da dieser von fast allen Peripheriegeräten verstanden wird, auch wenn sie von anderen Anbietern stammen.

6.1 ASCII

Der **A**merican Standard **C**ode for **I**nformation **I**nterchange (Amerikanischer Standard Code für Informations Austausch) ist ein 7-Bit Code, durch ihn können also 128 verschiedene Zeichen dargestellt werden. Der ASCII-Code ist in 4 Gruppen zu je 32 Zeichen eingeteilt:

Dezimal	Bereich
0 - 31	Steuerzeichen (nichtdruckbar)
32 - 63	Satz- und Sonderzeichen und Ziffern
64 - 95	Große Buchstaben
96 - 127	Kleine Buchstaben

Wie bei ASC erklärt wird, verwendet Ihr Computer den Bereich von 0 bis 95 fast ohne Änderung. Auf einige Unregelmäßigkeiten, vor allem auch in Verbindung mit Druckern, die deutschen Zeichensatz haben, soll jedoch hingewiesen werden. Folgende Codes des ASCII-Zeichensatzes sind u.U. mit verschiedenen Zeichen belegt:

Code	**Standardzeichen**	**Alternativzeichen (Beispiel)**
35	Doppelkreuz	Paragraphenzeichen
60	spitze Klammer auf	kein Zeichen (SPACE)
62	spitze Klammer zu	kein Zeichen (SPACE)
64	Klammeraffe	ß
91	eckige Klammer auf	Ü oder Ä
92	Schrägstrich	Ö
93	eckige Klammer zu	Ä oder Ü
94	Pfeil nach oben	Abwandlungen davon oder SPACE
95	Unterstrich	Pfeil nach links
123	geschweifte Klammer auf	ü oder ä
124	senkrechter Strich	ö
125	geschweifte Klammer zu	ä oder ü
126	Schlange	

Einige Auswirkungen dieser Zeichenalternativen sehen Sie in diesem Handbuch. Das 'Doppelkreuz' (Dateinummer) wird vom Drucker als Paragraphenzeichen dargestellt und 'Pfeil nach oben' (Exponentiation) wird überhaupt nicht gedruckt. Deshalb wurde es im Text durch h ersetzt. Auch die beiden spitzen Klammern werden nicht gedruckt und deshalb durch g (größer als) und k (kleiner als) ersetzt.

ASCII

CONTROL-CODES			
0	BLK	@	`
1	!	A	a
2	"	B	b
3	#	C	c
4	$	D	d
5	%	E	e
6	&	F	f
7	'	G	g
8	(	H	h
9	)	I	i
10	*	J	j
11	+	K	k
12	,	L	l
13	-	M	m
14	.	N	n
15	/	O	o
16	0	P	p
17	1	Q	q
18	2	R	r
19	3	S	s
20	4	T	t
21	5	U	u
22	6	V	v
23	7	W	w
24	8	X	x
25	9	Y	y
26	:	Z	z
27	;	[	{
28	<	\	|
29	=	]	}
30	>	↑	~
31	?	←	⌧

6.2 ASC

Da im Commodore-BASIC 256 verschiedene Stringcodes existieren, kann der ASCII-Code intern nur mit einer Änderung verwendet werden:

Die kleinen Buchstaben werden von den großen unterschieden, indem zum ASCII-Code der großen Buchstaben 128 addiert wird, was gleichbedeutend damit ist, daß Bit 7 gesetzt wird. Bit 7 enthält also die Information, ob die SHIFT-Taste gedrückt wurde oder nicht. Dies gilt nicht nur für die Buchstaben, sondern auch für alle Steuertasten.

Damit kann der ASC-Code folgendermaßen eingeteilt werden, was seine Darstellung auf dem Bildschirm betrifft, wenn mit PRINT gedruckt wird:

Zwischen 32 und 95 liegen die Sonderzeichen und großen Buchstaben und zwischen 160 und 223 Graphikzeichen und die kleinen Buchstaben.

Zwischen 0 und 31 liegen Steuerzeichen und zwischen 128 und 159 die entsprechenden SHIFT-Steuerzeichen.

Bleiben noch die beiden Bereiche 96-127 und 224-255. Sie enthalten, was die Darstellung auf dem Bildschirm anbelangt, keine neue Information, weil sie genauso dargestellt werden, wie die Bereiche 32-63 und 160-191. Ein Beispiel soll dies verdeutlichen: Sowohl CHR$(33) als auch CHR$(97) wird auf dem Bildschirm als '!' dargestellt.

Im ASC Code sind also 2*64=128 druckbare Codes enthalten, sowie 2*32=64 nichtdruckbare Steuerzeichen. Die restlichen 2*32=64 Codes finden beim Drucken keine Verwendung. Sie werden erst interessant, wenn ein String als Bytefeld (Binärfeld) verwendet wird.

Mit der kommerziellen Tastatur können nicht alle druckbaren Codes eingegeben werden, sondern nur 32-95 und 192-223. Die Graphikzeichen zwischen 160 und 191 können also mit dieser Tastatur nicht eingegeben werden. Durch die Steuertasten können noch einige Codes aus dem Bereich der Steuercodes eingegeben werden. Durch Revers-Darstellung innerhalb von Anführungszeichen können alle Steuercodes (0-31) und (128-159) eingegeben werden.

Einige Codes kommen sehr häufig im weiteren Text vor, oder werden allgemein oft verwendet und sollen deshalb kurz vorgestellt werden:

Code	Name im Text	Bedeutung
0	NUL / CHR$(0)	lauter Nullen, Füllzeichen
10	LF (Line Feed)	Zeilenvorschub
13	CR (Carriage Return)	Wagenrücklauf
27	ESC (Escape)	Kommandos folgen
32	SP / BL (Space / Blank)	Leerzeichen
64	Klammeraffe	amerikanisches 'and'-Symbol
141	SH-CR	CR gleichzeitig mit SHIFT gedrückt
160	SH-BL	SP gleichzeitig mit SHIFT gedrückt

NUL hat in der ASCII-Norm die Funktion eines Füllzeichens, das keinerlei Funktion hat. In Verbindung mit GET und INPUT bzw. mit Binärdateien hat es eine Sonderstellung (s. GET/INPUT).

CR/LF hat (historisch) die Funktion der Wagenrücklaufes bzw. Zeilenvorschubes bei Druckern / Fernschreibern. Innerhalb von Computer-Dateien hat es davon abgeleitet die Funktion eines Trennzeichens zwischen einzelnen Datensätzen oder Datenfeldern. Allerdings wird als Trennzeichen in der Regel nur CR und nicht LF verwendet. Andererseits gibt es Drucker, die entweder bei Empfang von CR automatisch auch einen Zeilenvorschub machen (Auto-Linefeed) oder tatsächlich beide Steuerzeichen benötigen, um die beiden Funktionen ausführen zu können. CR/LF hat also zu tun mit Druckern einerseits und CR alleine hat zu tun mit Dateien und Ein- / Ausgabebefehlen andererseits (s. GET/INPUT/PRINT). Beachten Sie bei INPUT auch die spezielle Funktion von SH-CR.

ESC hat bei intelligenten Peripheriegeräten, die ASCII verstehen, die Funktion einer Kommando-Einleitung. D.h. nach Empfang von ESC erwartet das Gerät eine bestimmte Anzahl von Kommando-Codes, die normalerweise gedruckt werden müssten, aber nach ESC eben nur als Befehl interpretiert werden. Vor allem bei Druckern ist diese Art der Kommandoübermittlung weit verbreitet.

SP / SH-SP wird deshalb erwähnt, weil beide Codes auf dem Bildschirm das gleiche Zeichen produzieren, nämlich einen leeren Fleck, aber für die Eingabe vom Bildschirm mit INPUT trotzdem verschiedene Wirkung haben.

Klammeraffe ist ein Ausdruck aus dem Datenverarbeiter-Jargon für das kleine a mit dem Haken, der es fast ganz umschließt. Dieses Zeichen ist im deutschen Sprachraum meistens nicht bekannt, auch nicht seine diversen Namen, wird aber in Zusammenhang mit der Floppy-Verwaltung evtl. gebraucht.

CONTROL				ZEICHEN			BUCHSTABEN					
										TEXT		GRAPHIK
0	128			32	160		64	192		@	—	@ —
1	129			33	161	! ▮	65	193		a	A	A ♣
2	130			34	162	" ▀	66	194		b	B	B |
3	131	STOP..	RUN	35	163	# ▔	67	195		c	C	C ┼
4	132			36	164	$ _	68	196		d	D	D ┼
5	133			37	165	% ▏	69	197		e	E	E ┼
6	134			38	166	& ▒	70	198		f	F	F ─
7	135	BELL	BELL	39	167	' ▔	71	199		g	G	G |
8	136			40	168	(▒	72	200		h	H	H |
9	137	TAB	TABECL	41	169	) ▚	73	201		i	I	I ┌
10	138			42	170	* ▕	74	202		j	J	J ┌
11	139			43	171	+ ┣	75	203		k	K	K ┌
12	140			44	172	, ▪	76	204		l	L	L └
13	141	CR	SHCR	45	173	- ┗	77	205		m	M	M ╲
14	142	TEXT	GRAPH	46	174	. ┓	78	206		n	N	N ╲
15	143	SETTOP	SETBOT	47	175	/ ▬	79	207		o	O	O └
16	144			48	176	0 ┏	80	208		p	P	P ┌
17	145	CDOWN	CUP	49	177	1 ┻	81	209		q	Q	Q ■
18	146	RVSON	RVSOFF	50	178	2 ┳	82	210		r	R	R —
19	147	HOME	CLR	51	179	3 ┫	83	211		s	S	S ♠
20	148	DEL	INST	52	180	4 ▕	84	212		t	T	T ¦
21	149	DELLIN	INSLIN	53	181	5 ▐	85	213		u	U	U ╱
22	150	ERSEND	ERSBEG	54	182	6 ▐	86	214		v	V	V X
23	151			55	183	7 ▔	87	215		w	W	W ○
24	152			56	184	8 ▀	88	216		x	X	X ♠
25	153	SCRUP	SCRDOW	57	185	9 ▄	89	217		y	Y	Y |
26	154			58	186	: ◤	90	218		z	Z	Z ♠
27	155	ESC		59	187	; ▖	91	219		┼	[	[┼
28	156			60	188	< ▝	92	220		▒	\	\ ▒
29	157	CRIGHT	CLEFT	61	189	= ┘	93	221		|	]	] ¦
30	158			62	190	> ▪	94	222		↑	↑	↑ π
31	159			63	191	? ▚	95	223		←	←	← ▜

6.3 BSC (Bildschirmcode)

Der Vollständigkeit halber wird hier auch der BSC aufgeführt.

Der Bildschirmcode ist im Allgemeinen nur von Interesse, wenn im Assembler gearbeitet wird. Mit PEEK und POKE kann zwar auch von BASIC aus direkt auf den Bild-Wiederhol-Speicher zugegriffen werden, dann wird man aber ausschließlich im BSC arbeiten, so daß die angegebene Tabelle genügt.

Zur Tabelle muß nur noch hinzugefügt werden, daß die angegebenen Werte + 128 das entsprechende Revers-Zeichen auf den Bildschirm bringen.

Der algorithmische Zusammenhang zwischen ASC- und BSC - Code, der nur aus einfachen Bitmanipulationen besteht, ist im Rahmen eines BASIC-Handbuches nicht interessant.

Die Umschaltung zwischen Graphik und Kleinschreibung kann auf folgende Weise erfolgen:

Graphik POKE 59468,12
Text POKE 59468,14

TEXT	GRAPHIK			TEXT	GRAPHIK		
0	@	-	32 BLK	64	-	-	96 SHBL
1	a	A	33 !	65	A	♠	97 ▌
2	b	B	34 "	66	B	┃	98 ▬
3	c	C	35 #	67	C	─	99
4	d	D	36 $	68	D	─	100 _
5	e	E	37 %	69	E	─	101 ▏
6	f	F	38 &	70	F	-	102 ▒
7	g	G	39 '	71	G	▏	103 ‾
8	h	H	40 (	72	H	▏	104 ▒
9	i	I	41)	73	I	╲	105 ▒
10	j	J	42 *	74	J	╲	106 ▕
11	k	K	43 +	75	K	╲	107 ├
12	l	L	44 ,	76	L	╲	108 ▖
13	m	M	45 -	77	M	╲	109 └
14	n	N	46 .	78	N	╲	110 ┐
15	o	O	47 /	79	O	┌	111 ▁
16	p	P	48 0	90	P	┐	112 ┌
17	q	Q	49 1	91	Q	●	113 ┴
18	r	R	50 2	82	R	▬	114 ┬
19	s	S	51 3	83	S	♥	115 ┤
20	t	T	52 4	84	T	▕	116 ▏
21	u	U	53 5	85	U	╱	117 ▌
22	v	V	54 6	86	V	X	118 ▐
23	w	W	55 7	87	W	O	119 ▔
24	x	X	56 8	88	X	♠	120 ▬
25	y	Y	57 9	89	Y	▕	121 ▄
26	z	Z	58 :	90	Z	♦	122 ▗
27	[	[	59 ;	91	+	+	123 ▖
28	\	\	60 <	92	▒	▒	124 ▝
29	]	]	61 =	93	▏	▏	125 ┘
30	↑	↑	62 >	94	π	π	126 ▘
31	←	←	63 ?	95	◥	◥	127 ▚

II. BASIC-Anweisungen

Dieser Abschnitt beschreibt ausführlich die Eigenschaften aller BASIC-Anweisungen. Vorausgesetzt ist, daß Sie die Abschnitte über Datenarten, Darstellungsweisen und Ausdrücke und Operatoren in den wesentlichen Zügen verstanden haben, bzw. daß Sie bei Bedarf in diesen Kapiteln nachschlagen.

Form

Zur Form in den Abschnitten 'Format' sollten Sie einige Hinweise beachten:

'GROSSBUCHSTABEN' kennzeichnen Worte der BASIC-Sprache, die exakt so und in dieser Reihenfolge vom Computer erwartet werden.

'kleinbuchstaben' kennzeichnen als **Platzhalter** Stellen, in die irgendwelche Parameter gehören. Die Art der Parameter ergibt sich entweder aus der Bezeichnung oder aus nachstehenden Erklärungen.

Das Format für die dargestellten Befehlsausdrücke ist folgendes:

Alle Befehle bzw. Zeichen in **Fettdruck** müssen so eingegeben werden, wie sie in der Beschreibung stehen.

Alle Zeichen in **Normaldruck** sind Teile, die unter Umständen wegfallen können. Beachten Sie aber bitte, daß dann die Ersatzwerte gelten, die jeweils angeführt sind.

Blanks (Leerstellen) sind überall nur der Übersichtlichkeit halber eingestreut. Sie brauchen nicht mit eingegeben zu werden.

Abgesehen von Abkürzungen werden alle **Buchstaben** in BASIC-Zeilen **ohne SHIFT** eingegeben, unabhängig davon, ob sie dann auf dem Bildschirm groß oder klein erscheinen. Lediglich innerhalb von Anführungszeichen dürfen auch Buchstaben erscheinen, die mit SHIFT getippt wurden (Stringkonstante).

Eine Tabelle der Befehlsabkürzungen finden Sie im Anhang.

Reihenfolge

Innerhalb der Befehlsbeschreibungen gehen die Erklärungen von vorne nach hinten vom Prinzipiellen zu den Details. Viele dieser Details brauchen Sie sich nicht zu merken, Sie sollten aber eine Ahnung haben, wo sie stehen, um bei Bedarf schnell nachschlagen zu können.

Die Einführungen zu Anweisungsgruppen sollen auf Zusammenhänge und gemeinsame Eigenschaften hinweisen. Solange Sie nicht mit diesen Zusammenhängen vertraut sind, sollten Sie auch beim Nachschlagen diese Einführung jeweils mitlesen.

Die einzelnen Befehlsgruppen und auch die einzelnen Befehlsbeschreibungen innerhalb der Gruppen sind weitgehend unabhängig voneinander gehalten, so daß Sie nicht unbedingt die Gruppen in der Reihenfolge des Inhaltsverzeichnisses lesen müssen und auch innerhalb der Gruppen einzelne Teile auslassen können.

1. Programme laden und speichern

1.1 Einführung

Mit den Befehlen LOAD (DLOAD), SAVE (DSAVE) können Programme von Band oder Diskette geladen, bzw. dorthin abgespeichert werden. Durch VERIFY kann überprüft werden, ob die Aufzeichnung richtig erfolgt ist.

SAVE bezieht sich immer auf BASIC-Programme. LOAD und VERIFY dagegen kann auch beliebige Speicherinhalte behandeln.

Diese Befehle sind prinzipiell Systembefehle, werden also im Direktmodus angewandt. Sie sind allerdings auch im Programm zugelassen, was bei SAVE und VERIFY nur in Ausnahmefällen sinnvoll sein wird, bei LOAD aber das Nachladen von Programmteilen durch 'Overlay' ermöglicht.

Die beiden Disk-Befehle DSAVE und DLOAD unterscheiden sich in der Wirkung kaum von den 'normalen' BASIC-Befehlen. Deshalb wird hier nur das Format vorgestellt. Bitte lesen Sie die Feinheiten im Floppy-Handbuch nach.

Folgende Parameter sind erforderlich:

dn	**Dateiname**	
gn	**Gerätenummer**	
ln	**Laufwerknummer**	(nur bei Floppy)

dn ist ein String-Ausdruck
gn ist ein Byte-Ausdruck
ln ist bei LOAD/SAVE/VERIFY ein Teilstring, der vor dn stehen muß,
bei DLOAD und DSAVE ein Zahlenausdruck, der nur 0 oder 1 ergeben darf.

Der Dateiname darf nicht länger als 16 Zeichen sein.

Die Gerätenummer ist 1 oder 2 für die Rekorder bzw. 8 (Standard) für die Floppy.

Die Laufwerknummer kann nur 0 oder 1 sein.

Bei **DLOAD** und **DSAVE** ist zu beachten, daß Parameter**variablen in Klammern** stehen müssen.

1.2 LOAD, DLOAD (Laden)

Zweck

Mit LOAD bzw. DLOAD können Programme von Band oder Diskette in den Arbeitsspeicher geladen werden.

Format

LOAD ln: **dn** , gn

DLOAD dn , D ln , U gn

gn = 1 (Rekorder 1) ist Ersatzwert für nicht angegebene Gerätenummer bei LOAD
gn = 8 (Floppy 8) ist Ersatzwert für nicht angegebene Gerätennummer bei DLOAD
ln = 0 (Laufwerk 0) ist Ersatzwert für nicht angegebenes Laufwerk bei DLOAD

Beispiele

LOAD "1:PROGRAMM1",8	PROGRAMM1 von Gerät 8, Laufwerk 1 laden
DLOAD "PROGRAMM1",D1,U8	identisch
LOAD DN$,GN	Name (und Laufwerk) steht in DN$, Gerät in GN
DLOAD (DN$),U(GN),D(LN)	Variable in Klammern! DN$ enthält den Programmnamen, GN die Gerätenummer und LN die Laufwerknummer
LOAD DN$	vom Rekorder 1

LOAD vom Floppy

Nach LOAD "NAME",8 oder DLOAD "NAME" meldet der Rechner SEARCHING FOR NAME und LOADING, falls das Programm vorhanden ist, oder FILE NOT FOUND ERROR, falls nicht. FILE NOT FOUND ERROR wird auch gemeldet, wenn wegen eines beliebigen Floppy-Fehlers der Zugriff nicht möglich war.

Wird bei LOAD das Laufwerk nicht angegeben, so sucht das Floppy automatisch auf beiden Laufwerken, falls es das gewünschte Programm nicht auf dem ersten Laufwerk findet. Dagegen sucht es bei DLOAD nur auf Laufwerk 0, falls kein Laufwerk angegeben wurde und meldet dann schon FILE NOT FOUND, falls das Programm nicht existiert.

Die Taste SHIFT/RUN lädt das erste Programm der Diskette im Laufwerk 0 und startet es sofort. (Dies gilt nicht, wenn ein Floppy 3040 angeschlossen ist.)

Der leere Name wird durch SYNTAX ERROR abgelehnt.

Die Behandlung des **Rekorders** erfolgt in einem gesonderten Kapitel.

LOAD im Direktmodus

Im Direktmodus bewirkt LOAD außer dem eigentlichen Laden des BASIC-Programms auch, daß die Zeiger der Speicherverwaltung auf das Ende des Programms eingestellt werden. Das bedeutet, daß vorher vorhandene Programme und Variablen vollständig gelöscht werden!

Hinter LOAD kann kein weiterer Befehl angegeben werden, er würde nicht ausgeführt werden!

LOAD"NAME",8:RUN hat also **nicht** die Wirkung, daß das Programm geladen und sofort gestartet wird!

Die Wirkung von LOAD **innerhalb eines Programms** ist bei **Overlay** beschrieben.

Parametergrenzen / Fehlermöglichkeiten

Bei Gerätenummer 0 (Tastatur) oder 3 (Bildschirm) wird SYNTAX ERROR gemeldet, bei Werten außerhalb der Bytedefinition (gn kleiner als 0 oder größer als 255) ILLEGAL QUANTITY ERROR. Bei Gerätenummern von nicht vorhandenen Geräten wird FILE NOT FOUND ERROR oder DEVICE NOT PRESENT ERROR gemeldet.

1.3 SAVE, DSAVE (Abspeichern)

Zweck

Mit SAVE (DSAVE) können BASIC-Programme auf Band oder Floppy abgespeichert werden.

Format

SAVE ln: **dn** , gn
DSAVE dn , D ln , U gn

Beispiele

SAVE "1:PROGRAMM1",8	PROGRAMM1 auf Gerät 8, Laufwerk 1 absp.
DSAVE "PROGRAMM1",D1,U8	identische Wirkung
SAVE DN$,GN	Name und Laufwerk steht in DN$, Gerät in GN
DSAVE (DN$),U(GN),D(LN)	Variable in Klammern, LN enthält Laufwerk
SAVE "NAME"	auf Rekorder 1

Besonderheiten von SAVE beim Floppy

Im Gegensatz zu LOAD muß bei SAVE das Laufwerk unbedingt angegeben werden! Bei DSAVE wird automatisch auf 0 abgespeichert, wenn kein Laufwerk angegeben wird.

Falls irgendwelche Floppy-Fehler auftreten, wird dies nicht automatisch auf dem Bildschirm gemeldet, sondern lediglich die Fehlerlampe zeigt an, daß ein Fehler aufgetreten ist. Achten Sie deshalb auf die Fehlerlampe des Floppy. Leuchtet sie, können Sie durch

?DS$

fragen, welcher Fehler aufgetreten ist. Zwei Fehlermeldungen, die bei SAVE vorkommen können, wollen wir kurz vorstellen. Wegen anderen Fehlermeldungen schlagen Sie bitte im Floppy-Handbuch nach.

63, FILE EXISTS bedeutet, daß schon ein Programm mit diesem Namen existiert. Ändern Sie deshalb entweder den Namen des neuen Programms oder löschen Sie das alte Programm (SCRATCH).

26, WRITE PROTECT ON weist Sie darauf hin, daß der Schreibschutz geklebt ist.

Der Dateiname muß mindestens ein Zeichen enthalten, sonst wird SYNTAX ERROR gemeldet.

Bei SAVE auf ein nicht vorhandenes Gerät wird **keine** Fehlermeldung gebracht, sondern sofort READY gemeldet. Achten Sie also darauf, ob das Floppy wirklich gelaufen ist!

Fehlermöglichkeiten

Die Gerätenummern 0 und 3 werden durch DEVICE NOT PRESENT ERROR abgelehnt. Soll auf nicht vorhandene Geräte abgespeichert werden, wird entweder DEVICE NOT PRESENT ERROR oder nichts gemeldet. Wenn die Gerätenummer kein Bytewert ist, wird ILLEGAL QUANTITY gemeldet.

1.4 VERIFY (Überprüfen)

Zweck

Durch VERIFY kann die Aufzeichnung eines Programms auf Band oder Floppy überprüft werden. Dazu wird das gerade im Speicher stehende Programm mit dem angegebenen Programm auf Floppy (Band) verglichen.

Format

Die Syntax von VERIFY und die Parameter sind identisch mit der von LOAD. Beachten Sie bitte, daß es für VERIFY keine spezielle Disk-Version gibt.

VERIFY In: **dn** , gn

Beispiele

VERIFY "1:PROGRAMM1",8	PROGRAMM1 auf Laufwerk 1 im Gerät 8
VERIFY	das erste Programm auf Rekorder 1

Anwendung

Die häufigste Anwendung sieht so aus, daß Sie ein Programm mit SAVE abspeichern und unmittelbar danach mit VERIFY überprüfen. Um sich die Arbeit zu erleichtern, merken Sie sich vielleicht folgendes Verfahren:

'Sa"0:NAME",8' oben links auf den Bildschirm schreiben und ausführen. Mit 'HOME' zurückgehen und mit 'Ve' 'Sa' überschreiben und wieder RETURN drücken. Dies können Sie schon tippen, während das Floppy abspeichert. Damit erledigen Sie SAVE und VERIFY in einem Arbeitsgang. Beim Rekorder müssen Sie allerdings warten, bis er fertig ist, weil die Tastatur durch Rekorderoperationen lahmgelegt wird.

VERIFY meldet in jedem Fall VERIFYING und darunter entweder OK oder VERIFY ERROR, falls keine Übereinstimmung festgestellt wurde.

Beim Band sollte bei VERIFY ERROR das Band bzw. die Einstellung des Schreib-/Lesekopfes überprüft werden, falls sicher ist, daß nicht der falsche Programmname angegeben wurde.

Beim Floppy ist normalerweise VERIFY ERROR nur möglich, wenn eine Fehlermeldung des Floppy übersehen wurde.

VERIFY funktioniert aber nicht nur unmittelbar nach SAVE, sondern vergleicht immer das aktuelle BASIC-Programm im Speicher mit dem Inhalt der Programm-Datei auf Floppy oder Band. Wenn Sie also wissen wollen, ob das Programm im Speicher identisch ist mit einem beliebigen auf Floppy oder Band, beantwortet VERIFY diese Frage.

Für Neugierige

VERIFY macht fast das gleiche wie LOAD: Es holt die Bytes eines Programms in den Rechner. Der Unterschied ist nur, daß LOAD diese Bytes dann im Speicher ablegt, während VERIFY die geholten Bytes mit denen im Speicher vergleicht. Die Wirkung auf Floppy oder Rekorder ist daher identisch mit der Wirkung von LOAD.

2. Programme starten und stoppen

2.1 Einführung

Nachdem ein Programm in den Speicher geladen (oder eingetippt) wurde, kann es durch RUN oder GOTO gestartet werden.

Durch den STOP-Befehl kann es an definierter Stelle unterbrochen werden, bzw. durch die STOP-Taste irgendwo, wo es sich zu diesem Zeitpunkt gerade befindet.

Nach einem Abbruch durch STOP bzw. durch END kann das Programm durch CONT an der dem Abbruch folgenden Stelle fortgesetzt werden.

2.2 RUN / GOTO (Programm starten)

Zweck

RUN dient dazu, ein Programm am Anfang oder einer bestimmten Zeile 'kalt' zu starten.

GOTO läßt das Programm an einer bestimmten Stelle 'warm' starten.

Format

RUN zeilennummer

GOTO zeilennummer

Beispiele:

RUN	startet Programm im Speicher bei der ersten Zeile
RUN 100	bei Zeile 100
GOTO 10	startet bei 10, löscht Variable nicht

Anmerkungen

RUN bewirkt immer einen sogenannten 'Kaltstart'. D.h. die gesamten Variablen-Listen werden gelöscht, so daß unmittelbar nach dem Start keine Variable existiert.

GOTO bewirkt prinzipiell einen 'Warmstart', d.h. alle Variablenwerte bleiben erhalten.

Der exakte Unterschied zwischen RUN und GOTO besteht darin, daß RUN die Funktion von CLR beinhaltet, GOTO dagegen nicht.

Wenn hinter RUN keine Zeilennummer angegeben ist, wird das Programm in der ersten Zeile gestartet.

GOTO darf nicht ohne Zeilennummer verwendet werden.

2.3 STOP / CONT (Programm stoppen / fortsetzen)

STOP ist ausführlich bei END/STOP (s. Sprunganweisungen) beschrieben. Danach bewirken sowohl die Anweisung STOP im Programm als auch die Taste STOP während des Programmlaufes einen Abbruch des Programms.

Mit **CONT** kann der Programmlauf nach der Abbruchstelle fortgesetzt werden. CONT setzt bei der nächsten Anweisung auf, also nicht erst am Anfang der nächsten Zeile.

Auch nachdem das Programm verlassen wurde, weil während **INPUT** eine leere Eingabe gemacht wurde, kann mit CONT dieses INPUT wiederholt werden. Man beachte den Unterschied zum Abbruch durch STOP: Nach STOP wird bei der nächsten Anweisung weitergemacht, nach der leeren Eingabe wird die aktuelle Anweisung, nämlich das INPUT wiederholt!

Achten Sie darauf, daß Sie **bei CONT keinen Fehler machen,** sich also nicht verschreiben und auch der Rest der Zeile leer ist, da sonst ein SYNTAX ERROR die Folge sein könnte und dann keine Möglichkeit mehr gegeben ist, mit CONT fortzufahren:

Nach Fehlermeldungen kann das Programm nicht mit CONT fortgesetzt werden. Der Computer meldet diesen Umstand durch CAN'T CONTINUE ERROR (keine Fortsetzung möglich). Der Grund liegt darin, daß nach Fehlermeldungen der Stack leer ist, also Unterprogramm-, Schleifen- und Ausdrucks-Hierarchien zerstört sind.

Dies ist nicht zu verwechseln damit, daß Variableninhalte gelöscht sind, was nämlich nicht der Fall ist. Das bedeutet, daß man nach dem Ausstieg durch eine Fehlermeldung mit GOTO an geeigneter Stelle das Programm warm starten kann. Während der Testphase tut man gut daran, solche **Einstiegspunkte** (Warmstartpunkte) zu schaffen, falls sie nicht ohnehin vorhanden sind.

3. Programm- und Speicherverwaltung

3.1 Einführung

Etwas willkürlich wurden die folgenden vier Anweisungen unter dieser Überschrift zusammengefaßt:

Mit **LIST** können Sie das gerade im Arbeitsspeicher stehende BASIC-Programm ganz oder teilweise auf den Bildschirm oder den Drucker (s. CMD) ausgeben.

Mit **NEW** löschen Sie ein BASIC-Programm aus dem Arbeitsspeicher, mit **CLR** nur alle Variablen.

FRE gibt Auskunft, wieviel Arbeitsspeicher noch frei ist.

Alle vier Anweisungen sind in erster Linie zum Gebrauch im Direktmodus bestimmt, können aber prinzipiell auch in Programmen vorkommen.

3.2 LIST (Programm auslisten)

Zweck

LIST gibt ein Listing des aktuellen BASIC-Programmes auf den Bildschirm, bzw. nach CMD auf einen Drucker aus.

Format

LIST von - bis

An Beispielen sollen die vier Möglichkeiten von LIST gezeigt werden:

LIST	listet das ganze Programm
LIST 20 - 50	listet von Zeile 20 bis 50 einschließlich, sofern vorhanden
LIST 20 -	listet ab 20 bis zum Programmende
LIST - 50	listet von der ersten Zeile bis 50

Bitte beachten Sie, daß LIST 0 ebenfalls das ganze Programm auslistet.

Alle Zeilen, die nach LIST auf dem Bildschirm stehen, können ohne weiteres beliebig verändert werden. Die geänderte Zeile wird aber nur durch Drücken der RETURN-Taste übernommen, nicht aber durch SHIFT/RETURN!

Listing verlangsamen

Nachdem die unterste Bildschirmzeile beschrieben wurde, rollt der Bildschirm hoch. Wenn Sie zu diesem Zeitpunkt die Taste 'RVS' gedrückt halten, wird nach jedem Hochschieben eine kleine Pause gemacht.

Durch Drücken der Taste 'STOP' wird das Listing mit BREAK unterbrochen.

LIST im Programm

LIST kann zwar innerhalb eines Programms verwendet werden, das Programm wird aber nach dem Listing beendet, der Rechner meldet sich mit READY.

3.3 CLR (Variable löschen)

Zweck

CLR löscht alle Variablen.

Format

CLR (nicht mit der HOME/CLR - **Taste** verwechseln)

Anmerkungen für Fortgeschrittene

CLR löscht eigentlich die Variableninhalte nicht, sondern 'zerstört' nur die 4 Zeiger, die die Variablenlisten verwalten.

Im Einzelnen wird (44,45) (Ende einfache Variablen) und (46,47) (Ende indizierte Variable) auf den Wert von (42,43) (Ende BASIC-Text) gesetzt. Entsprechend wird (48,49) und (50,51) (Anfang und Ende des letzten Strings) auf den Wert von (52,53) (Speicherobergrenze) gebracht.

Dadurch finden alle Zugriffsroutinen leere Listen vor.

CLR kann ohne weiteres im Programm verwendet werden. Bedenken Sie aber, daß Sie nicht selektiv löschen können, sondern nur alle Variablen.

Eine spezielle Anwendung von CLR ist bei Overlay beschrieben.

3.4 NEW (BASIC-Programm löschen)

Zweck

NEW löscht das BASIC-Programm im Speicher.

Format

NEW

NEW beinhaltet die Funktion von CLR.

Anmerkungen für Fortgeschrittene

Ähnlich wie CLR löscht auch NEW nicht wirklich, sondern setzt nur (42,43) auf 0,4 (= 1024 = Beginn des BASIC-Speichers)

NEW im Programm bewirkt, daß sich das Programm selbst löscht. Die Meldung danach ist READY wie nach END.

3.5 FRE (freier Speicherplatz)

Zweck

FRE übergibt die Anzahl der noch freien Bytes des BASIC-Arbeitsspeichers.

Format

FRE (0)

Der Parameter in Klammern muß angegeben werden, weil FRE sonst nicht durch die SYNTAX-Prüfung kommt. Er hat aber weiter keinerlei Bedeutung.

Beispiel:

```
? FRE(0)
```

FRE(0) im Direktmodus

FRE wird im Direktmodus oft dann gebraucht, wenn das Programm mit OUT OF MEMORY ERROR abgebrochen wurde. Wie bei Stack beschrieben, kann dann durch FRE festgestellt werden, ob wirklich der Arbeitsspeicher oder aber der Stapel 'übergelaufen' ist.

FRE(0) im Programm

Falls Sie platzkritische Dimensionierungen haben, deren Platzbedarf sich im Dialog mit dem Anwender ergibt, können Sie mit FRE prüfen, ob der Platz noch reicht und dies gegebenenfalls dem Anwender schonender mitteilen als durch OUT OF MEMORY ERROR.

Anmerkungen für Fortgeschrittene

FRE muß erst die Garbage Collect Routine rufen (s. Stringverwaltung). Wenn sehr viele Strings im Speicher stehen, kann diese Routine bis zu einer Sekunde brauchen. Damit kann also auch FRE solange brauchen. Dies ist zu beachten, wenn FRE im Programm verwendet wird.

FRE berechnet die Differenz zwischen (48,49) (Anfang des letzten Strings) und (46,47) (Ende der indizierten Variablen).

4. Wertzuweisungen

4.1 Einführung

Der wohl am häufigsten benötigte Befehl ist die Wertzuweisung an eine Variable. Wir unterscheiden drei Arten der Zuweisung. Die am häufigsten benutzte Art ist die Zuweisung des Ergebnisses eines Ausdrucks an eine Variable (=). Für die Zuweisung von Konstanten kann auch der Befehl READ benutzt werden, der in Zusammenhang mit DATA steht. Schließlich gibt es noch die sehr wichtige Möglichkeit, einer Variable einen Wert zuzuweisen, der von außen kommt, also von einem Peripheriegerät. Dies wird durch INPUT bzw. GET ermöglicht. Diese beiden Befehle haben wir allerdings dem Komplex der Ein- / Ausgabebefehle zugeordnet.

4.2 Allgemeine Zuweisung (LET)

Zweck

An die Variable auf der linken Seite des 'Gleichheitszeichens' wird das Ergebnis des Ausdrucks auf der rechten Seite übergeben.

Format

LET **variable** = **ausdruck**

LET kann entfallen. Da LET nur sinnvoll wäre, wenn Ihr BASIC Programm auch auf einem anderen Rechner laufen muß, der LET verlangt, werden wir LET nicht verwenden.

Trotzdem soll der Sinn kurz erklärt werden, weil dadurch gleichzeitig eine Fehlinterpretation des '=' vermieden wird: Die Zuweisung mit LET könnte übersetzt werden mit: Lasse den Inhalt von 'variable' gleich werden mit dem Ergebnis von 'ausdruck'. Die Zuweisung ist also nicht zu verwechseln mit einer mathematischen Gleichung. In LET steckt der Hinweis darauf, daß die beiden Seiten **nicht gleichzeitig gleich** sind, sondern erst gleich **werden.**

Beispiel: A = A + 1

Als mathematische Gleichung wäre dies ein Widerspruch. Die Zuweisung aber berechnet **zuerst** das Ergebnis des Ausdrucks auf der rechten Seite und speichert **dann** das Ergebnis in die Variable auf der linken Seite!

Um Mißverständnisse auszuschließen, ist es üblich, die Zuweisung im Beispiel folgendermaßen zu lesen:

Der Wert von A **ergibt sich aus** A+1.

Alle Probleme, die sich bei der Zuweisung ergeben können, sind in den Abschnitten 'Daten', 'Variable', 'Umwandlungen', 'Ausdrücke' und 'Stack' ausführlich beschrieben. Wir wollen hier nur nochmal auf die wesentlichen Fehlermöglichkeiten aufmerksam machen.

Die Zuweisung muß im Typ übereinstimmen:

A = A$ bzw. A$ = A	TYPE MISMATCH ERROR (bei der Zuweisung)
A = B + A$	TYPE MISMATCH ERROR (im Ausdruck)

Umwandlung von Integer in Gleitkomma ist immer möglich, umgekehrt nur im Wertebereich von Integer und mit Änderung des Wertes durch Abrunden.

Beispiele:

A = B%	Integer in Gleitkomma ist immer möglich
A = -3.5 B% = A ? B% -4	Wertänderung durch Abschneiden
A = 32768 B% = A ILLEGAL QUANTITY ERROR	kein Integer-Wert

Der Ausdruck darf nicht umfangreicher werden, als der Stack zuläßt, sonst wird OUT OF MEMORY ERROR gemeldet (s. Stack).

4.3 Zuweisung von Konstanten in DATA durch READ

4.3.1 Einführung

In DATA-Anweisungen können Zahlen- oder Stringkonstante abgelegt sein, die durch READ an Variable zugewiesen werden können.

Normalerweise verwendet man DATA und READ in erster Linie, um Variablenfelder mit Werten zu versorgen. Diese Werte können dann entweder der endgültige Inhalt dieses Feldes sein, oder nur sogenannte Anfangswerte.

Eine weitere Anwendung von DATA ist der Ersatz einer Band- oder Floppydatei durch eine DATA-Datei während der Entwicklung eines Programmes bzw. zu Testzwecken.

In der Regel ist es nicht sinnvoll, einfache Variable aus DATAs mit Anfangswerten zu versorgen, da dies mit einer normalen Zuweisung durch '=' genauso effektiv und übersichtlicher erreicht werden kann.

Die Anweisungen werden jetzt in der Reihenfolge READ, DATA, RESTORE vorgestellt. Anschließend wird ihre Zusammenarbeit beschrieben.

4.3.2 READ (Lesen aus DATA)

Zweck

READ übergibt Konstante aus DATA in Variable.

Format

READ variable1 , variable2 , variable3 .... : weitere Anweisungen

Beispiel

READ A1,B%,C$,D(I),C$(J)

Hinter READ kann eine oder mehrere Variable stehen. Jeder Variablentyp ist zugelassen, die Anzahl ist nur durch die Zeilenlänge begrenzt. Die Variablen werden durch **Komma** getrennt. Durch Doppelpunkt getrennt können hinter READ weitere Anweisungen stehen.

Die Anordnung der Variablen ist identisch mit der bei INPUT. Dadurch kann zu Testzwecken INPUT sehr leicht durch READ ersetzt werden!

4.3.3 DATA (Datenfeld)

Zweck

In DATA Anweisungen werden Konstante innerhalb des BASIC-Programmes gespeichert, die mit READ in Variable übertragen werden.

Format

DATA konstante1 , konstante2 , .. : weitere Anweisungen

'konstante' kann eine Zahlen- oder Stringkonstante sein.

Hinter DATA können mehrere Konstanten stehen. Sie müssen durch **Komma** getrennt werden.

Hinter einer DATA Anweisung können durch Doppelpunkt getrennt weitere Anweisungen in der gleichen Zeile folgen. Wenn keine weitere Anweisung in der gleichen Zeile folgt, muß kein Doppelpunkt stehen.

Stringkonstante

Grundsätzlich müssen Stringkonstante in Anführungszeichen stehen. Bei DATA gibt es jedoch eine Ausnahme: Wenn keine SHIFT-Buchstaben oder Steuerzeichen oder Komma oder Doppelpunkt enthalten sind, können die Anführungsstriche entfallen.

Wenn zwei Kommas unmittelbar aufeinander folgen, wird dies als leerer String bzw. die Zahl 0 interpretiert!

Wichtig ist, daß eine DATA Anweisung nicht versehentlich durch ein Komma abgeschlossen wird. Dies würde nämlich ein zusätzliches leeres Element bedeuten!

Blanks

Innerhalb von Anführungszeichen werden Blanks berücksichtigt, sie werden also durch READ in die Stringvariable mit übernommen.

Dagegen werden **führende** Blanks auch bei Strings ignoriert, wenn **keine** Anführungszeichen verwendet werden. **Nachfolgende** Blanks werden aber mitgelesen.

Bei Zahlen haben Blanks keinerlei Bedeutung.

Sonderzeichen

Sonderzeichen, wie z.B. die Cursorbewegungen oder RVS ON/OFF müssen innerhalb von Anführungszeichen stehen.

Wird versucht, z.B. RVS ON ('R' revers) ohne Anführungszeichen einzugeben, so zeigt 'LIST' ein normales 'R', das dann natürlich nicht mehr die Bedeutung von RVS ON hat!

SHIFT-Zeichen

Wenn SHIFT-Buchstaben (allgemein Zeichen mit ASC-Code größer als 128) ohne Anführungszeichen eingegeben werden, zeigt 'LIST', daß sie nicht übernommen wurden!

4.3.4 RESTORE (DATA-Zeiger zurücksetzen)

Zweck

Der 'DATA-Zeiger' merkt sich, welche Konstante als letzte durch READ gelesen wurde. Er sorgt dadurch dafür, daß die Inhalte der einzelnen DATAs in ihrer Reihenfolge gelesen werden können.

RESTORE setzt den DATA-Zeiger auf die erste Konstante im ersten DATA des Programms. (RUN bewirkt automatisch RESTORE.)

Dadurch kann eine DATA-Datei öfter gelesen werden.

Format

RESTORE

Anmerkung

Es ist nicht möglich, auf das DATA in einer bestimmten Programmzeile zurückzusetzen.

4.3.5 Eigenschaften einer DATA-Datei

Die Daten hinter allen DATAs bilden eine serielle Datei. Die Eigenschaft dieser Datei ist vergleichbar mit Band- oder Floppydateien. Allerdings mit zwei Unterschieden.

Erstens kann man nur eine einzige serielle Datei in DATAs anlegen. Das kommt daher, daß das erste READ das erste Datenelement im ersten DATA liest und die weiteren READs das jeweils nächste Element. Wenn ein DATA erschöpft ist, wird beim nächsten DATA weitergelesen, sofern es vorhanden ist.

Man kann durch READ also nur alle Datenelemente vom ersten bis zum letzten einlesen, nicht aber z.B. bestimmte DATA-Zeilen anwählen. Durch den Befehl RESTORE hat man lediglich die Möglichkeit, wieder beim allerersten DATA mit dem Lesen zu beginnen.

Zweitens handelt es sich um eine 'Nur-Lese-Datei', weil es vom Programm aus keine Möglichkeit gibt, Daten z.B. vom Bildschirm aus in DATAs zu übernehmen.

Reihenfolge der DATAs

Die Reihenfolge der Daten wird nur durch die Reihenfolge der Zeilennummern der einzelnen DATAs bestimmt. Dagegen ist es unwichtig, ob das erste DATA vor dem ersten READ steht oder umgekehrt. Der Zeiger auf das nächste DATA-Element ist nämlich völlig unabhängig vom normalen Programmzeiger.

Beispiel

```
10 DATA 3 :REM ANZAHL DER STAEDTE
20 DATA 8000,"Muenchen"
22 DATA 1000,"Berlin"
24 DATA 6000,"Frankfurt"
30 READ AZ: DIM PZ%(AZ),SN$(AZ)
40 FOR I = 1 TO AZ: READ PZ%(I),SN$(I): NEXT
```

Die erste Konstante der DATA-Datei gibt die Zahl der einzulesenden Elemente an (Beispiel!). Aufgrund dieser Information werden die beiden Felder für die Postleitzahlen und die Städtenamen dimensioniert (30). Die Schleife in 40 liest dann alle DATA-Elemente in die beiden Felder ein.

4.3.6 Fehlermeldungen

Wenn mehr READs vorhanden sind, als DATA-Elemente, wird abgebrochen mit OUT OF DATA ERROR IN .....

Diese Meldung würde auftauchen, wenn im Beispiel in Zeile 10 die Zahl 4 anstatt 3 stehen würde.

Diese Fehlermeldung kann nur durch READ verursacht werden, im folgenden häufigen Fall allerdings sehr versteckt:

Auf dem Bildschirm steht READY., der Computer befindet sich im sogenannten Direktmodus. Sie gehen mit dem Cursor in die Zeile mit dem READY. und drücken die RETURN-Taste. Der Computer bringt dann die OUT OF DATA Meldung. Er hat READY. als READ Y interpretiert, aber kein DATA gefunden!

Dies zeigt nebenbei, daß READ auch im Direktmodus zugelassen ist, wofür es aber kaum Anwendungen geben dürfte.

SYNTAX ERROR IN .....

Diese Meldung ist sehr bemerkenswert, weil sie erstens falsch ist und zweitens eine Zeilennummer meldet, die man nicht ohne weiters als Quelle des Fehlers erkennen kann: (Aber auch ein Computer ist manchmal nur ein Mensch.)

Wenn versucht wird, mit READ in eine Zahlenvariable ein Stringelement aus DATA zu lesen, meldet der Rechner SYNTAX ERROR. Die Zeilennummer, die dahinter angegeben wird, ist die der DATA-Anweisung, nicht die der READ-Anweisung!

Die richtige Fehlermeldung müßte lauten: TYPE MISMATCH ERROR!

Beispiel

```
10 DATA "123"
20 DATA ABCD
30 READ A
40 RESTORE: READ A$
50 READ A
```

Nach RUN wird SYNTAX ERROR IN 10 gemeldet, nach RUN 40 dagegen SYNTAX ERROR IN 20!

5. Ein- / Ausgabe - Anweisungen

5.1 Einführung

Ihr Computer ist in einer sehr leistungsfähigen und übersichtlichen Weise mit seiner Peripherie verbunden. Hardwaremäßig wurde dies durch Verwendung des IEC-Bus erreicht, durch den maximal 12 Peripheriegeräte an den Rechner angeschlossen werden können. Auf der Seite der Betriebssoftware wurde das Konzept des IEC-Bus konsequent zur Verwendung von logischen Dateien ausgebaut. Da die logische Datei zentrale Bedeutung im E/A-Konzept hat, soll zuerst dieser Begriff erläutert werden.

Die logische Datei

Zum Verkehr mit der Aussenwelt werden eigentlich nur zwei verschiedene Befehle benötigt: Einer, der Daten vom Rechner zur Peripherie überträgt und ein anderer, der Daten von der Peripherie zum Rechner bringt. Dem Rechner ist es dabei gleichgültig, an wen er Daten schickt, bzw. wer ihm Daten übergibt, da die Norm des IEC-Bus dafür sorgt, daß die Datenübertragung für alle angeschlossenen Geräte einheitlich erfolgt.

U.a. sieht diese Norm vor, daß Byte für Byte nacheinander übertragen werden muß. Vom Rechnerbetriebssystem aus gesehen haben also alle 'Dateien' die Eigenschaft, daß sie Bytes nacheinander aufnehmen, bzw. abgeben können. Man kann also nicht ohne weiteres ein ganz bestimmtes Byte verlangen, sondern muß alle Bytes vom Anfang der Datei bis zu diesem Byte lesen, ganz gleich welches Gerät dahinter steht.

Konsequenterweise sprechen die E/A-Befehle (INPUT, GET, PRINT) nicht unmittelbar bestimmte Geräte an, sondern über die sogenannte **logische Dateinummer** eine symbolische Datei, die wir als **logische Datei** bezeichnen.

Natürlich muß jeder logischen Datei ein Gerät zugeordnet werden. Dies wird von OPEN erledigt. OPEN verknüpft eine logische Dateinummer mit einer Gerätenummer (Primäradresse) und einer Sekundäradresse. Was es mit diesen beiden Ausdrücken auf sich hat, soll im folgenden kurz erklärt werden:

Gerätenummer und Sekundäradresse

Jedes Gerät am IEC-Bus wird durch eine Gerätenummer angesprochen, die zwischen 0 und 15 liegen kann. Nun gibt es aber Geräte, die selbst gewissermaßen mehrere Geräte verkörpern, etwa verschiedene Betriebszustände (z.B. Drucker) oder verschiedene Datenkanäle (z.B. Floppy). Solche Geräte benötigen deshalb eine sogenannte Sekundäradresse, durch die die Betriebsart bzw. der Datenkanal festgelegt wird. Der Begriff Sekundäradresse kommt daher, daß die Gerätenummer auch Primäradresse heißt. Die Sekundäradresse kann zwischen 0 und 31 liegen.

Weil Ihr Computer selbst vier Geräte ist, sind die ersten vier Gerätenummern für ihn selbst reserviert. Im einzelnen sind dies:

Nummer	Gerät
0	Tastatur
1,2	Rekorder 1,2
3	Bildschirm

Diese Geräte hängen nicht wirklich am IEC-Bus, werden aber vom Betriebssystem für BASIC so dargestellt, als ob.

Da der Computer es Ihnen mit sich selbst besonders einfach machen will, gibt es für Tastatur, Rekorder 1 und Bildschirm abgekürzte E/A-Befehle, nämlich INPUT und PRINT für den Bildschirm, GET für die Tastatur und SAVE/LOAD/VERIFY für Rekorder 1.

Nun wollen wir noch kurz auf die weiteren Anweisungen eingehen, die Sie in diesem Abschnitt erwarten.

Da ist einmal das sehr wichtige CLOSE, mit dem Dateien geschlossen, also abgemeldet werden. CLOSE ist das Gegenstück zu OPEN.

Beim Verkehr mit Peripheriegeräten muß oft der Zustand des Gerätes durch den Status ST abgefragt werden. Insbesondere meldet der Status beim Lesen, wann eine Datei zuende ist.

Die Funktionen DS und DS$ stehen zur Verfügung, um den Zustand des Floppy zu melden.

Das Kommando CMD sorgt u.a. dafür, daß Sie Ihr BASIC-Listing auch auf den Drucker ausgeben können.

Schließlich sind noch drei Funktionen vorhanden, um Ihnen die Formatierung von Bildschirmausgaben zu erleichtern (TAB, SPC, POS).

5.2 OPEN (Datei öffnen)

Zweck

Durch OPEN wird die Verbindung zwischen den Ein- / Ausgabebefehlen PRINT§ , INPUT§ und GET§ und einem Gerät am IEC - Bus hergestellt (s. logische Datei). Das gleiche gilt für DOPEN (Disk-OPEN). Die Syntax von DOPEN unterscheidet sich aber stark von der OPEN-Syntax. DOPEN ist im Floppy-Handbuch beschrieben, deshalb wird hier darauf verzichtet.

Format

OPEN la , gn , sa , dn

la:	Logische-Adresse	(1 - 127 / 128 - 255)
gn:	Geräte-Nummer	(0 - 15)
sa:	Sekundär-Adresse	(0 - 31)
dn:	Datei-Name	(Stringausdruck)

la, gn, sa sind jeweils **Byte**ausdrücke, **dn** ist ein **String**ausdruck.

la muß angegeben werden, die restlichen Parameter können entfallen. Wird aber irgendein Parameter angegeben, müssen alle vor ihm auch angegeben werden.

Nicht angegebene Parameter erhalten automatisch folgende Werte zugewiesen:

gn	1 (Rekorder 1)
sa	0
dn	leerer String

Die Bedeutungen der Parameter werden in den folgenden Punkten ausführlich erklärt.

5.2.1 Logische Adresse (la)

Die logische Adresse oder logische Dateinummer muß entspechend bei PRINT§la, INPUT§la und GET§la angegeben werden. Durch die logische Adresse **la** wird bei jedem dieser Befehle das Gerät mit der Gerätenummer **gn** und der Sekundäradresse **sa** angesprochen.

Gleichzeitig kann **nur eine** Datei mit einer **bestimmten la** geöffnet sein. Wird versucht, zum zweiten Mal ein OPEN mit der gleichen la durchzuführen, ist die Folge ein FILE OPEN ERROR (IN ...) (Datei ist bereits geöffnet).

Vom CBM Betriebssystem sind maximal **10 Dateien gleichzeitig** zugelassen. Wird diese Anzahl überschritten, erfolgt die Fehlermeldung TOO MANY FILES ERROR (IN...) (zu viele Dateien).

Bei beiden Fehlermeldungen werden gleichzeitig vom Betriebssystem **alle** anderen **Dateien geschlossen!**

Im Wertebereich für la sind zwei Bereiche zu unterscheiden:

Bei la von 1 - 127 wird hinter einem PRINT§ , das nicht mit ';' abgeschlossen ist, **nur CR**, bei la von 128 - 255 wird CR + LF gesendet (s. PRINT§).

5.2.2 Gerätenummer / Primäradresse (gn)

Die Gerätenummer wird auch physikalische Adresse, Primäradresse oder Geräteadresse genannt. Folgende Adressen sind vom Betriebssystem fest belegt:

gn	Gerät	
0	Tastatur	(für Sonderanwendungen)
1	Rekorder 1	
2	Rekorder 2	
3	Bildschirm	(für Sonderanwendungen)

Für das Floppy ist standardmäßig 8 und für den Drucker 4 vorgesehen.

Ist kein Gerät angeschlossen, so wird bei OPEN - sofern dn angegeben wurde - bzw. bei PRINT§, INPUT§ oder GET§ gemeldet: DEVICE NOT PRESENT ERROR (Gerät ist nicht vorhanden / nicht eingeschaltet)

Werte größer als 31 sollten Sie unbedingt vermeiden, da das Betriebssystem dadurch 'abstürzen' kann oder sonst irgendwelche sonderbaren Reaktionen bringt.

5.2.3 Sekundäradresse (sa)

Die Sekundäradresse stellt bestimmte Betriebsarten ein (Drucker) oder wählt Datenkanäle aus (Floppy). Die entsprechenden sa mit ihren Wirkungen sind in den jeweiligen Gerätehandbüchern beschrieben. Wegen der Bedeutung bei Banddateien schlagen Sie bitte bei 'Rekorder' nach. Bei der Tastatur (gn=0) und dem Bildschirm (gn=3) sind alle sa-Werte erlaubt, es ist aber sinnlos, eine sa anzugeben, da sie keinerlei Wirkung hat.

5.2.4 Dateiname / Kommandostring (dn)

Der vierte Parameter des OPEN ist ein String. Die Bedeutung für das Floppy ist im Floppy-Handbuch beschrieben. Für die Rekorder enthält er den Dateinamen (s. Rekorder).

Bitte beachten Sie folgenden Hinweis zum Format: Falls Sie den Dateinamen bei Band oder Floppy nicht als 1 Konstante bzw. Variable angeben, müssen die einzelnen Teile durch '+' verbunden werden, und nicht etwa durch ';', wie bei PRINT, es handelt sich nämlich um einen Stringausdruck!

5.2.5 OPEN-Tabelle (Für Fortgeschrittene)

Die Zuordnung zwischen den einzelnen ln, gn und sa wird vom Betriebssystem in einer Tabelle verwaltet. Die Anzahl der offenen Dateien steht in 174. Falls nun eine der beiden Fehlermeldungen FILE (NOT) OPEN ERROR auftritt, durch die ja alle Dateien geschlossen werden, kann man den Zeiger in 174 auf die Anzahl der offenen Dateien setzen und dadurch diese Dateien wieder öffnen. Dies ist deshalb möglich, weil der Inhalt der eigentlichen Tabelle durch diese Fehlermeldungen nicht verändert wird, sondern nur durch CLOSE.

Beispiel:

Sie hatten zwei Dateien offen, ehe die Fehlermeldungen kamen, dann können Sie durch

```
POKE 174 , 2
```

die Dateien wieder öffnen. Dieser Tip ist natürlich nur für das Arbeiten im Direktmodus sinnvoll oder für das Austesten von Programmen.

5.3 CLOSE (Datei schließen)

Zweck

CLOSE schließt die logische Datei la.

Format

CLOSE la

la ist die logische Adresse, also ein Byteausdruck. Pro CLOSE muß genau eine la angegeben werden.

Anwendung

'CLOSE' meldet die Datei ab, die durch 'OPEN' der logischen Adresse la zugeordnet wurde. Man sagt auch, die Datei wird **geschlossen.** Jedem 'OPEN' muß im Programm ein 'CLOSE' zugeordnet werden, sobald die betreffende Datei nicht mehr gebraucht wird.

'CLOSE' wirkt auf das Rechner-Betriebssystem, kann aber auch auf das Peripheriegerät wirken. Im Rechner wird dadurch die betreffende logische Dateinummer wieder freigegeben. Im Floppy wird z.B. der entsprechende Kanal freigegeben. Beim Band wird eine Schreibdatei durch EOF (End Of File) abgeschlossen, bzw. bei sa=2 zusätzlich mit EOT.

Wichtiger Hinweis

Durch **CLR, LOAD, RUN** wird die Zuordnungstabelle von logischen Adressen zu Geräten und Sekundäradressen gelöscht. Für den **Rechner** sind dadurch **alle Dateien geschlossen.** Dies trifft aber **nicht für das Peripheriegerät** zu. Sollte dadurch versehentlich eine Floppy- oder Band-Schreibdatei nicht mehr geschlossen werden können, kann **POKE 174 , anzahl offene Dateien** (s. OPEN) weiterhelfen.

Wenn **LOAD im BASIC Programm** ausgeführt wird (Overlay), bleiben **alle Dateien geöffnet.**

Fehlermöglichkeiten

CLOSE **ohne la** dahinter erzeugt SYNTAX ERROR. Dagegen kann man hinter CLOSE mehrere la durch Komma getrennt angeben, von denen aber nur die erste berücksichtigt wird. Trotzdem wird kein Fehler gemeldet. Deshalb sei es nochmal ausdrücklich gesagt:

Hinter CLOSE muß **genau eine la** stehen, nicht weniger und nicht mehr!

Durch CLOSE einer la, auf die keine Datei geöffnet wurde, wird kein Schaden angerichtet, dieser Befehl wird ignoriert.

DCLOSE

Disk-CLOSE hat die gleiche Wirkung und den gleichen Zweck wie CLOSE, kann sich aber nur auf Floppy-Dateien beziehen. Der Vorteil von DCLOSE ist, daß damit alle Floppy-Dateien mit einem Befehl geschlossen werden können. Näheres entnehmen Sie bitte dem Floppy-Handbuch.

5.4 CMD (Bildschirm-Ausgabe umlenken)

Zweck

CMD adressiert das la zugeordnete Gerät als Listener (Datenempfänger) und meldet dieses Gerät erst dann wieder ab, wenn PRINT§ auf eine beliebige offene Datei ausgeführt wird, jedoch nicht bei PRINT (Bildschirm).

Format

CMD la , string

la	Logische Adresse (s. OPEN)
string	auszugebender String (Format wie bei PRINT, kann auch entfallen

Erklärungen

Alle Daten, die auf den Bildschirm geschrieben werden, werden durch CMD auf ein beliebiges Peripheriegerät umgelenkt.

Dies gilt für alle Ausgaben, ganz gleich ob sie vom BASIC-Programm durch PRINT oder durch Meldungen des Betriebssystems oder z.B. durch LIST erzeugt werden. Dagegen werden Eingaben von der Tastatur während INPUT oder im Direktmodus weiterhin nur auf den Bildschirm geschrieben und können dort auch geändert werden.

Folgende Befehlssequenz gibt das Listing an den Drucker aus:

Ausführliche Form:

```
OPEN 4 , 4
CMD 4
LIST
PRINT § 4
```

Kurzform:

```
Op4,4:Cm4:Li
Pr4
```

Wir haben die Kurzform mit Abkürzungen zusätzlich mit aufgeführt, weil diese Sequenz oft im Direktmodus gebraucht wird.

Statt der Datei auf den Drucker könnte auch eine auf das Floppy geöffnet werden. Dadurch könnte der Bildschirmtext statt auf den Drucker auf Floppy ausgegeben werden.

Statt des Listings kann auch sonstiger Text ausgegeben werden. Ersetzen Sie z.B. Li durch DIrD1, so wird das Inhaltsverzeichnis des Laufwerks 1 an den Drucker ausgegeben.

Beendung des CMD-Zustandes

Solange nur Ausgaben auf den Bildschirm erfolgen, bleibt der CMD-Zustand eingeschaltet. Sie erkennen ihn im Direktmodus daran, daß nach Ausführung eines Befehls nicht wie gewöhnlich READY auf dem Bildschirm steht, sondern nur der Cursor. (Die Meldung READY wurde an das Peripheriegerät ausgegeben.)

Durch ein PRINT§la auf eine beliebige offene Datei wird CMD abgeschaltet! Deshalb steht in dem obigen Beispiel in der nächsten Zeile PRINT§4.

Eine etwas unfeinere Art ist, einen SYNTAX ERROR zu erzeugen, indem man Unsinn auf den Bildschirm schreibt und RETURN drückt. Diese Merthode ist natürlich nur im Direktmodus sinnvoll.

Feinheiten

Falls Sie CMD innerhalb eines Programms anwenden wollen (zu Testzwecken), beachten Sie bitte, daß jedes PRINT§ den CMD-Zustand beendet. Der Inhalt dieses PRINT§ wird allerdings zusätzlich auf das CMD-Gerät ausgegeben, d.h. für diese eine Datenübertragung sind zwei Geräte gleichzeitig Empfänger.

Sie können also nicht eine Ausgabe auf Floppy mit CMD auf den Drucker umlenken, jedenfalls nicht mehr als ein einziges PRINT§. Sollten Sie eine solche Umlenkung wollen, brauchen Sie nur im OPEN anstatt des Floppy (z.B. 8) den Drucker (z.B. 4) angeben.

Fehlermöglichkeiten

Da CMD eine logische Adresse als Parameter hat, ist natürlich Voraussetzung für CMD, daß vorher durch OPEN die entsprechende Datei geöffnet wurde, anderenfalls wird FILE NOT OPEN ERROR gemeldet.

Versuchen Sie nicht, zwei CMD-Kanäle zur gleichen Zeit zu öffnen. Das Betriebssystem reagiert darauf mit verkehrten Fehlermeldungen und der Cursor kann nach RETURN nicht in die nächste Zeile, sondern in die dritte Spalte der gleichen Zeile gehen. Dieses Zustand können Sie durch einen SYNTAX ERROR beenden (s.o.).

5.5 PRINT (drucken)

Zweck

PRINT § la, gibt beliebige Daten an Peripheriegeräte oder speziell den Bildschirm aus.

Format

PRINT § la , ausgabedaten
oder
PRINT ausgabedaten
bzw.
? ausgabedaten

Die erste Form bewirkt die Ausgabe von Daten auf eine logische Datei. Dabei ist vorausgesetzt, daß eine Datei la mit OPEN geöffnet wurde.

Die zweite Form (ohne la) bewirkt die Ausgabe auf den Bildschirm. Die dritte Form zeigt lediglich die Abkürzung von PRINT (auf den Bildschirm).

Bitte verwechseln Sie nicht '?' und 'Pr', bzw. schreiben Sie **nicht '?§'**, ein SYNTAX ERROR wäre die Folge.

'la' ist die logische Adresse (s. OPEN)

'ausgabedaten' sind nach spezieller PRINT-Syntax aneinandergereihte numerische Ausdrücke oder Stringausdrücke. Diese Syntax ist für beide PRINT-Formen identisch und tritt sonst nur noch bei CMD auf. Das Besondere an der PRINT Syntax sind die Trennzeichen und Trennfunktionen zwischen den Ausdrücken.

Die Trennzeichen zwischen den Ausdrücken haben sehr unterschiedliche Wirkung auf das Druckformat. Zusätzlich muß für die Wirkung des gleichen Trennzeichens unterschieden werden zwischen PRINT§la und PRINT. Um das ganze nicht zu kompliziert werden zu lassen, wollen wir zuerst das Trennzeichen beschreiben, das für beide PRINTs gleiche Wirkung hat.

Strichpunkt

Der Strichpunkt trennt die einzelnen Ausdrücke hinter PRINT voneinander.

Beispiel: PRINT A+3 ; B ; C$; "abcd"

Dieser Befehl gibt das Ergebnis von A+3, den Inhalt von B und C$, sowie die Zeichen 'abcd' **unmittelbar hintereinander** auf den Bildschirm aus.

Der Strichpunkt dient also nur für den Interpreter dem Zweck, die einzelnen Ausdrücke auseinanderhalten zu können, fügt aber keine zusätzlichen Leerräume oder dergleichen beim Drucken ein.

Eine besondere Bedeutung kommt dem Strichpunkt am Ende des PRINT-Befehls zu:

Beispiel

```
PRINT A$
PRINT A$;
```

Im ersten Fall wird nach dem Drucken des Inhalts von A$ sofort eine neue Zeile begonnen (auf dem Bildschirm), während im zweiten Fall der (gedachte) Cursor hinter dem letzten Zeichen von A$ stehen bleibt. Ein nächstes PRINT beginnt also im ersten Fall am Anfang der nächsten Zeile zu drucken, im zweiten Fall unmittelbar hinter der letzten Druckposition.

Bei PRINT§ muß diese Wirkung allgemeiner formuliert werden:

Wird PRINT§ **nicht mit Strichpunkt** (oder Komma) abgeschlossen, so wird automatisch nach dem letzten Zeichen ein **CR** (Carriage Return = CHR$(13)) gesendet, sofern die logische Adresse **kleiner als 128** ist. Bei la **größer als 127** wird nach dem CR **auch noch LF** (Line Feed = CHR$(10)) gesendet. Bitte beachten Sie, daß bei **Floppy und Rekorder** deshalb nur **la kleiner als 128** verwendet werden sollten (s. INPUT).

Zahlenformat

Allen Zahlen gemeinsam ist, daß sie an der ersten Stelle ein Blank (anstatt +) oder ein Minuszeichen haben. Nach der Zahl steht entweder 'Cursor nach rechts' (PRINT) oder Blank (PRINT§).

Gleitkommazahlen werden je nach Zahlenwert als normale **Gleitkommazahl** (ohne Exponent) oder als **Mantisse mit Exponent** (Exponentialdarstellung) ausgegeben. Die Regeln sind folgende:

Ist der **Betrag** der Zahl größer als 1, so wird in Exponentialdarstellung umgeschaltet sobald die Anzahl der Stellen 9 überschreitet:

```
 100 000 000  ergibt     100000000
1000 000 000  ergibt     1E+09
```

Ist der **Betrag** kleiner als 1, so wird ab 1E-03 in Exponentialdarstellung umgeschaltet:

```
.0111          ergibt     .0111
.00111         ergibt     1.111E-03
```

Der **Exponent** wird in jedem Fall **zweistellig** und mit **explizitem Vorzeichen** ausgegeben, d.h. '+' wird nicht als Blank, sondern als '+' dargestellt.

Die Anzahl der **Nachkommastellen** bei **Exponentialdarstellung** kann zwischen 0 und 8 liegen. Im Fall von 0 Nachkommastellen entfällt auch der Punkt. Bei der Exponentialdarstellung steht immer genau eine Ziffer vor dem Punkt.

Bei **Gleitkommadarstellung werden maximal 9 Stellen** ausgegeben. Die Verteilung dieser 9 Stellen auf Vor- und Nachkommastellen kann von 1 Vorkomma- und 9 Nachkommastellen bis zu 9 Vorkomma- und 0 Nachkommastellen alle Stufen annehmen: Bei 0 Nachkommastellen entfällt auch der Punkt.

Der folgende Ausdruck zeigt alle möglichen Darstellungsformen:

```
1.23123123E-03
.0123123123
.123123123
1.23123123
12.3123123
123.123123
1231.23123
12312.3123
123123.123
1231231.23
12312312.3
123123123
1.23123123E+09
```

Es gibt keinen BASIC-Befehl, der Zahlen in ein bestimmtes Format bringt.

Stringformat

Strings werden ohne irgendwelche zusätzlichen Zeichen aneinandergereiht, solange nicht die speziellen Formatierungsbefehle z.B. Blanks dazwischenschreiben.

Komma

Trennt man Ausdrücke anstatt durch Strichpunkt durch Komma, so wird dadurch auf dem **Bildschirm** (PRINT) jeweils ab der nächsten erreichbaren **10-er-Spalte** gedruckt, also ab Spalte 10, 20, 30 usw.

Die Spalten sind innerhalb einer Zeile von 0 bis 79 durchnummeriert.

Der Zwischenraum bis zur nächsten Druckposition wird nicht verändert, alle Zeichen, die vorher im Zwischenraum standen, stehen nach der Tabulierung durch Komma immer noch dort. Die Tabulierung auf dem Bildschirm wird also nicht durch Blanks, sondern durch 'Cursor nach rechts' erreicht. Dies gilt auch für TAB und SPC!

Bei **PRINT§1a** (nicht auf den Bildschirm) sollte Komma als Trennzeichen zwischen Ausdrücken **nicht verwendet** werden, da die Tabulierung dort nicht funktioniert, sondern bei jedem Komma 10 Blanks (CHR$(32)) gesendet werden, was ja **keine Tabulierung** bewirkt.

TAB( )

Mit TAB kann jede Spalte einer Bildschirmzeile direkt erreicht werden.

In den Klammern hinter TAB kann ein Byte-Ausdruck stehen, dessen Ergebnis zwischen 0 und 79 liegen sollte, aber theoretisch bis zu 255 groß sein darf.

TAB arbeitet nur von links nach rechts. Wenn der Cursor z.B. schon in Spalte 30 steht und im PRINT TAB(17) steht, so wird die TAB-Anweisung **ignoriert.** Der Cursor kann durch TAB also **nicht nach links** bewegt werden.

Innerhalb einer Bildschirmzeile können (in aufsteigender Reihenfolge) mehrere TABs verwendet werden.

Der Tabulator ist nur für die Verwendung innerhalb **einer** Bildschirmzeile (40 bzw. 80 Spalten) gemacht. Wollen Sie mehrere Zeilen durch einen TAB-Sprung überbrücken, so müssen Sie folgendes Verhalten berücksichtigen: Bei TAB-Werten größer als 39 bzw. 79 wird der Sprung **nur beim ersten Mal richtig** ausgeführt, folgende TABs werden aber immer ab **Beginn** der Zeile berechnet, in der der Cursor gerade steht!

Bei **PRINT§la** (nicht auf den Bildschirm) hat **TAB identische Wirkung wie SPC** , tabuliert also nicht (s. Komma), sondern sendet soviele Blanks, wie der Ausdruck in den Klammern angibt!

SPC( )

SPC( ) überspringt bei PRINT (Bildschirm) soviele Spalten, wie der Byte-Ausdruck in Klammern angibt. Wie bei TAB werden dabei Zeichen im übersprungenen Bereich nicht gelöscht (Cursor nach rechts).

Bei PRINT§ werden soviele Blanks gesendet, wie in Klammern angegeben ist.

SPC( ) darf ebenso wie TAB( ) nur hinter PRINT auftreten. Man kann also z.B. nicht durch SPC( ) einer Stringvariablen Blanks zuweisen!

Hochschieben des Bildschirminhaltes

Wenn durch PRINT die letzte Bildschirmzeile überschritten werden soll, wird der gesamte Bildschirminhalt um eine Zeile nach oben geschoben, wobei die oberste Zeile verlorengeht.

Durch 'RVS' entsteht nach jedem Hochschieben eine Pause, was ein Verlangsamen der Bildschirmausgabe bewirkt.

Anmerkungen für Fortgeschrittene

1) Strichpunkt kann entfallen:

Hinter folgenden Zeichen kann der Strichpunkt als Trennzeichen entfallen:

)	Klammer zu
"	Anführungszeichen
%	Integervariable
$	Stringvariable

2) **Zahlen werden als String ausgegeben:**

Zahlen werden durch PRINT grundsätzlich als String ausgegeben. Dies gilt auch für die Ausgabe auf andere Geräte als den Bildschirm. Anders ausgedrückt, werden Zahlen **nicht** komprimiert als 5 Bytes (Gleitkomma) oder 2 Bytes (Integer) ausgegeben, sondern immer als String mit maximal 15 Zeichen.

3) **PRINT §la auf Bildschirm:**

Falls Sie über OPEN 3,3 und PRINT§3 mit einem PRINT§ auf den Bildschirm ausgeben, druckt Komma, TAB und SPC SPACE's auf den Bildschirm, d.h. Zeichen in den Zwischenräumen werden gelöscht! Komma und Tabulator tabulieren aber richtig.

Sie können diese Wirkung ausnützen, falls Sie die Zwischenräume wirklich löschen wollen.

4) **PRINT (auf Bildschirm):**

Das 'Überspringen' auf dem Bildschirm bei Komma, TAB und SPC wird durch 'Drukken' von 'Cursor nach rechts' bewirkt (s. Bildschirmverwaltung). Sie können dies sichtbar machen, indem Sie am Anfang des PRINT CHR$(34) drucken lassen. Dadurch wird die Bildschirmverwaltung in den Controlmodus geschaltet.

In diesem Zusammenhang sei auf eine Konsequenz des Zahlenformats auf dem Bildschirm hingewiesen. Jede Zahl wird von 'Cursor nach rechts' gefolgt. Dies macht sich manchmal negativ bemerkbar, wenn Zahlen revers gedruckt werden sollen, da zwar das führende Blank weiß gedruckt wird, aber nicht die Stelle nach der Zahl, da sie ja nicht als Blank gedruckt wird.

Durch folgendes Anhängsel können Sie diesen 'negativen (Ein)Druck' verhindern: Hinter der Zahl bzw. dem numerischen Ausdruck öffnen Sie Anführungszeichen, drücken die Taste 'Cursor nach links' und SPACE und schließen die Anführungszeichen. Dadurch wird rechts neben der Zahl ein Blank gedruckt. Sofern Sie vorher auf Revers-Druck geschaltet hatten, wird die Zahl mit nachfolgendem hellem Blank gedruckt.

5.6 POS( )

Zweck

POS meldet die Position (Spalte) des Cursors in der Bildschirmzeile, ist also nur in Zusammenhang mit PRINT sinnvoll. Das Ergebnis von POS ist ein ganzzahliger Wert zwischen 0 und 79.

Format

A = **POS(0)**

POS (0) ist eine Funktion mit Byte-Ergebnis. Der Parameter in Klammern hat keine Bedeutung, muß aber angegeben werden, um die Syntax-Prüfung zufriedenzustellen.

Das Ergebnis kann jeder Gleitkomma- oder Integervariablen zugewiesen werden.

5.7 INPUT (Eingabe mehrerer Zeichen)

Zweck

INPUT bringt Zahlenwerte oder Strings von einem Peripheriegerät oder vom Bildschirm in eine oder mehrere Variablen.

Format

INPUT "hinweis"; **v1** , v2 , ...

INPUT§ la , v1 , v2 , ...

hinweis	Stringkonstante
la	logische Adresse (s. OPEN)
v1, v2	beliebige Variablen

Vor INPUT§la muß mit OPEN eine Datei auf la geöffnet worden sein.

INPUT darf **nur im Programm** verwendet werden, nicht aber im Direktmodus.

5.7.1 Gemeinsame Eigenschaften

INPUT vom Bildschirm (INPUT) und von Peripheriegeräten (INPUT§,la) unterscheiden sich in einigen Details. Beachten Sie aber folgende Gemeinsamkeiten, um den Überblick nicht zu verlieren.

INPUT kann **maximal 80 Zeichen** (81 einschließlich Carriage Return) verarbeiten.

Der Grund liegt darin, daß alle Daten (gleichgültig ob vom Bildschirm, vom Rekorder oder vom IEC-Bus) in den BEP (BASIC-Eingabe-Puffer) gelegt werden. Erst von dort werden sie in die einzelnen Variablen übertragen. Dieser Puffer ist genau 81 Bytes lang. Die Übertragung in diesen Puffer wird durch **CR** (CHR$(13)) unterbrochen.

Zusätzlich kennt INPUT noch die Trennzeichen **Komma** (CHR$(44)) und **Doppelpunkt** (CHR$(58)), sofern nicht seit dem letzten CR ein Anführungszeichen (CHR$(34)) gesendet wurde. Komma und Doppelpunkt wirken aber nur innerhalb des BEP, sind also nicht mit CR vergleichbar, weil die Übertragung in den BEP durch sie nicht unterbrochen wird. Komma und Doppelpunkt trennen lediglich einzelne Variableninhalte voneinander ab.

INPUT kann in gemischter Reihenfolge Zahlendaten und Stringdaten lesen. In Stringvariable kann jeder String gelesen werden, insbesondere auch ein String, der eine Zahl darstellt. Umgekehrt können aber in Zahlenvariable nur Zeichenfolgen gelesen werden, die eine Zahl darstellen

Falls Sie Zahlen und Strings mischen, müssen Sie also darauf achten, daß hinter INPUT Zahlen- und Stringvariable in der richtigen Reihenfolge kommen.

Zahlenformat:

Folgende Zeichen sind zugelassen: Ziffern, Blank, '+', '-', 'E' (Exponent) (E muß **ohne SHIFT** eingegeben werden!)

Blanks können überall (auch zwischen Ziffern) auftreten, sie werden ignoriert.

Die Mantisse kann mehr als 9 Stellen haben, sie wird automatisch gerundet. Führende Nullen vor dem Komma werden ignoriert, nachfolgende Nullen nach dem Komma ebenfalls.

5.7.2 Besonderheiten des Bildschirm-INPUT

(1) **Der Hinweisstring:**

Um dem Benutzer mitzuteilen, welche Eingabe von ihm gefordert wird, kann unmittelbar hinter INPUT ein Hinweisstring angegeben werden:

Beispiel: INPUT "Vorname";VN$

Der Hinweisstring kann nur eine String**konstante** sein. Die erste Variable muß durch **Strichpunkt** vom Hinweisstring abgetrennt werden.

Die Befehlsfolge: PRINT H$;:INPUT E$ ist in der Wirkung identisch mit der Angabe des Hinweisstrings in INPUT, hat aber den Vorteil, daß hinter PRINT auch Variable zugelassen sind.

Wird Bildschirm-INPUT im Programm angetroffen, so erscheint auf dem Bildschirm in der nächsten Druckposition ein Fragezeichen und zwei Spalten dahinter blinkt der Cursor. Falls ein Hinweisstring angegeben wurde, wird dieser vor dem Fragezeichen ausgegeben. Der Anwender kann nun die von ihm verlangte Eingabe tippen und beliebig verbessern. Sobald er die RETURN-Taste drückt, wird seine Eingabe an die Variable(n) übergeben, die hinter INPUT aufgeführt sind.

Weil der Anwender völlige Freiheit auf dem Bildschirm hat, können Sie vom BASIC-Programm aus nicht kontrollieren, was er tut. Falls Sie einen übersichtlichen und abgesicherten Dialogaufbau auf dem Bildschirm anstreben, sollten Sie sich mit folgenden Reaktionen des INPUT auseinandersetzen:

(2) **Leere Eingabe:**

Drückt der Anwender RETURN, ohne in der Zeile etwas geschrieben zu haben, so wird der Programmlauf beendet. Das Betriebssystem meldet sich dann mit READY. Diese Reaktion auf die leere Eingabe soll die STOP-Taste ersetzen, da diese während INPUT ignoriert wird.

Mit CONT könnte der Anwender das Programm bei dem INPUT wieder starten, wo er es verlassen hat. Da für einen unbedarften Anwender derlei Krücken aber nicht akzeptabel sind, sollte man diesen Zustand verhindern.

Abhilfe: Sie können entweder INPUT§ verwenden (s.u.) oder mit Hilfe des Hinweisstrings hinter das Fragezeichen Zeichen schreiben, die keine Blanks sind:

INPUT"EINGABE:rr*lll";E$

r = 'Cursor nach rechts' oder SPACE
* = beliebiges Zeichen, auch SHIFT/SPACE
l = 'Cursor nach links'

Wenn Sie für * SHIFT/SPACE einsetzen, sieht die Eingabe auf dem Bildschirm genauso aus, wie das normale INPUT.

Wenn der Anwender keine Eingabe macht, wird das Zeichen für * in E$ übergeben.

* können auch mehrere Zeichen sein, allerdings müssen dann entsprechend viele l angegeben werden.

Die 'leere Eingabe' müssen Sie dann durch eine Abfrage auf * gesondert behandeln, da sie ja kein vom Benutzer eingegebenes Zeichen ist.

(3) **Zahlenvariable:**

Haben Sie hinter INPUT eine Zahlenvariable angegeben, aber der Benutzer gibt eine nichtnumerische Zeichenfolge ein, so wird REDO FROM START gemeldet und das INPUT-Fragezeichen erscheint in der nächsten Zeile nochmal. Dies soll dem Anwender signalisieren, daß er die ganze Eingabe wiederholen soll.

Abhilfe: Generell nur Stringvariable verwenden und hinterher Strings in Zahlen umwandeln, falls Zahlen benötigt werden (VAL). Dadurch fällt aber ihrem Programm die Aufgabe zu, die Eingabe zu überprüfen und entsprechend zu reagieren.

(4) **Mehr Eingaben als Variablen:**

Tippt der Anwender Komma oder Doppelpunkt und steht hinter INPUT nicht eine Variable mehr, als der Anwender Kommas und Doppelpunkt geschrieben hat, so wird EXTRA IGNORED gemeldet. Das soll dem Anwender andeuten, daß alles nach einem bestimmten Komma oder Doppelpunkt ignoriert wurde. Dem Anwender nützt diese Meldung aber nichts, da er es nicht ändern kann und das BASIC-Programm merkt nichts davon, so daß das Programm nicht auf derlei Eingabefehler reagieren kann.

Abhilfe: Entweder den Anwender ausdrücklich darauf hinweisen, daß er ',' und ':' nicht verwenden darf, oder (nur bei Stringvariablen) von ihm verlangen, daß er als erstes Zeichen ein Anführungszeichen schreibt.

(5) **Weniger Eingaben als Variablen:**

Stehen hinter INPUT mehrere Variable und gibt der Anwender nicht soviele Kommas oder Doppelpunkte ein, um entsprechend viele Datenfelder zu trennen, so werden in die nächste Zeile zwei Fragezeichen geschrieben und dahinter blinkt der Cursor. Dies soll dem Anwender signalisieren, daß mindestens noch eine Eingabe fehlt.

Abhilfe: Nicht mehr als eine Variable hinter INPUT verwenden. Das hat auch den Vorteil, daß Sie auf einzelne Falscheingaben gezielter reagieren können.

(6) **Falsche Zeile:**

Der Anwender kann aus Versehen mit Hilfe der vertikalen Cursorbewegungstasten die Eingabezeile verlassen. Ganz gleich, ob er dann wieder in die Eingabezeile zurückkehrt oder nicht, gilt die folgende Eigenschaft des INPUT:

Ist (oder war!) der Cursor in einer anderen Zeile als das INPUT-Fragezeichen, so wird die **ganze Zeile** eingelesen, in der der Cursor steht, sobald RETURN gedrückt wird.

Ist diese Zeile leer, wird allerdings das Programm nicht abgebrochen, sondern eine Stringvariable hinter INPUT erhält den leeren String zugewiesen, bzw. eine Zahlenvariable den Wert 0.

Abhilfe: Da in diesem Fall in der Stringvariablen mindestens das Fragezeichen enthalten ist, kann man durch eine Abfrage dies feststellen und z.B. durch eine nochmalige Eingabeaufforderung darauf reagieren. Angenommen, das INPUT--Fragezeichen steht ganz am Anfang der Zeile, dann können Sie durch folgende Abfrage den Eingabefehler korrigieren:

```
10 INPUT A$
20 IF LEFT$(A$,1)="?" THEN A$ = MID$(A$,3)
30 ...
```

5.7.3 Besonderheiten des Datei-INPUT§

Im Betrieb mit Commodore-Peripheriegeräten kommt INPUT nur in Verbindung mit Floppy oder Rekorder vor. Da von diesen Geräten in der Regel nur solche Daten gelesen werden können, die vorher mit PRINT§ in eine Datei geschrieben wurden, werden die Eigenschaften des INPUT§ mit PRINT§ verglichen, um die Zusammenhänge zu verdeutlichen. Mit diesem Verständnis können Fehler leichter vermieden werden.

Die übliche Anwendung von PRINT§ und INPUT§ bei Floppy oder Rekorder sieht vor, daß Variableninhalte mit PRINT§ in eine Datei geschrieben werden und mit INPUT§ zu einem späteren Zeitpunkt die Daten aus der Datei wieder in Variable gebracht werden müssen.

Folgende Hinweise sind zu beachten, um einen fehlerlosen Datenverkehr zu erhalten:

(1) **Reihenfolge der Daten:**

Sie können die Inhalte von Zahlen- und Stringvariablen in beliebiger Reihenfolge mit PRINT§ in eine Datei schreiben. Beim Lesen mit INPUT§ müssen Sie dann aber beachten, daß die Daten wieder in der gleichen Reihenfolge kommen, also in einander entsprechende Variablen gelesen werden müssen. (Meistens werden die Variablen hinter PRINT§ und INPUT§ sogar identisch sein, sie müssen es aber nicht.)

Versuchen Sie, einen nichtnumerischen String in eine Zahlenvariable zu lesen, erhalten Sie die Fehlermeldung FILE DATA ERROR. Meistens ist die Ursache ein überflüssiges Trennzeichen in der Datei oder eben eine falsche Reihenfolge der Variablen.

(2) **Länge der Daten**

Wie oben erwähnt, kann INPUT nur 80 Zeichen verarbeiten. Da CR die Übertragung in den Puffer beendet, müssen Sie darauf achten, bei PRINT§ nie mehr als 80 Zeichen ohne CR zu senden, da sonst STRING TOO LONG ERROR gemeldet wird.

Diese Einschränkung ist speziell bei Strings zu beachten, da der Inhalt von Stringvariablen ja 255 Zeichen lang sein kann.

Wie bei PRINT§ erklärt, werden Zahlen genauso wie Strings ausgegeben. Exakter ausgedrückt, sind Zahlen nur spezielle Strings. Wenn Sie sich nun noch daran erinnern, daß PRINT§ die Möglichkeit bietet, beliebig viele Zeichen ohne CR zu senden, indem einzelne Ausdrücke durch ';' getrennt werden, bzw. die PRINT-Anweisung durch ';' abgeschlossen wird, sehen Sie, daß auch ohne Stringvariable die 80 Zeichen-Grenze Beachtung verdient.

Für den Normalfall empfehlen wir, pro PRINT nur eine Variable (einen Ausdruck) zu verwenden, bzw. die Ausdrücke durch CR zu trennen. Zwei Beispiele sollen Ihnen dies verdeutlichen:

```
(10 OPEN 1,1,1,"NAME")
20 PRINT§1,A$:PRINT§1,B*3
```

Durch den fehlenden Strichpunkt am Ende jedes PRINTs sendet das Betriebssystem automatisch CR nach dem Variableninhalt zur Datei. Beachten Sie hierbei, daß die logische Adresse kleiner als 128 sein muß! (s. OPEN)

Völlig gleiche Wirkung erzielen Sie durch folgende Anordnung:

```
15 CR$=CHR$(13)
20 PRINT§1,A$CR$B*3
```

Der Variablen CR$ wird in Zeile 15 der Code für CR (Carriage Return) zugewiesen. In Zeile 20 wird dann nach A$ CR gesendet, dann das Ergebnis von B*3 und dann wieder CR (wegen des fehlenden Strichpunkts am Ende).

Bei beiden Beispielen muß sichergestellt sein, daß A$ nicht mehr als 80 Zeichen enthält. Dagegen kann B*3 (Gleitkommaausdruck) nie länger als knappe 20 Zeichen werden.

(3) **Andere Trennzeichen:**

Wie oben erwähnt, könnte auch Komma und Doppelpunkt verschiedene Variableninhalte voneinander trennen. Da dies aber das Pufferproblem nicht löst, empfehlen wir, nur CR als Trennzeichen zu verwenden!

Wollen Sie trotzdem Komma und Doppelpunkt verwenden, müssen Sie darauf achten, daß sich Ihre Daten nicht über mehr als 80 Zeichen erstrecken. Beachten Sie dabei, daß 6 Gleitkommazahlen schon länger als 80 Zeichen sein können.

(4) **Sonder - Codes**

PRINT§ kann in Zusammenhang mit Stringvariablen bzw. der Funktion CHR$() jeden Code zwischen 0 und 255 als ein Byte (Zeichen) in eine Datei schreiben. INPUT§ kann aber nicht jeden dieser Codes lesen. Folgende Codes können nicht gelesen werden oder nur bedingt:

Code	Darstellung	Bedeutung
0	CHR$(0)	'Null'
13	CHR$(13)	CR = Carriage Return
32	"SPACE"	Blank
34	CHR$(34)	Anführungszeichen
44	","	Komma
58	":"	Doppelpunkt

Die Codes 0 und 13 können von INPUT überhaupt nicht gelesen werden, d.h. ihr Wert wird nicht in eine Stringvariable übergeben.

Die letzten beiden werden nur dann übernommen, wenn seit dem letzten CR eine ungerade Anzahl von Anführungszeichen gelesen wurde, da sie sonst als Trennzeichen wirken, selbst aber nicht übernommen werden.

Blank wird im folgenden behandelt:

(5) **Blanks (SPACE, Leerraum)**

Bei Zahlen werden Blanks völlig ignoriert (s.o.). Bei Strings sind folgende Fälle zu unterscheiden:

Führende Blanks werden nur an die Stringvariable übergeben, wenn vorher CHR$(34) gelesen wurde. Ist dies nicht der Fall, werden sie ignoriert, d.h. der String wird ab dem ersten Code, der von 32 verschieden ist, übernommen.

Blanks zwischen anderen Zeichen werden immer übernommen, ebenso folgende Blanks.

Beachten Sie den Unterschied zwischen INPUT§ aus Dateien und INPUT vom Bildschirm bezüglich nachfolgender Blanks: Beim Bildschirm werden nachfolgende Blanks genauso wie führende ignoriert, falls nicht Anführungszeichen oder ein Trennzeichen die Blanks abschließt.

(6) **Status:**

Die Behandlung der Statusvariable ST ist bei Status beschrieben. Kurz zusammengefaßt ergeben sich für INPUT§ folgende Hinweise:

Solange ST = 0 ist, enthält die Datei weitere Daten.

Sobald ST 64 enthält ist der letzte Variableninhalt übertragen worden. Danach ist es sinnlos, weiterzulesen.

Versucht man es dennoch, enthält ST den Wert 2. In diesem Fall wird kein Wert an die INPUT-Variable übergeben, sie enthält also nach INPUT den Wert bzw. den String, den sie vorher enthielt und nicht etwa unbedingt 0 oder den leeren String!

(7) **INPUT§ von der Tastatur**

Das Problem der leeren Eingabe bei INPUT kann man umgehen, indem auf die Tastatur eine Datei geöffnet wird (OPEN 1,0) und dann durch INPUT§1 das normale INPUT ersetzt wird.

INPUT§ druckt kein Fragezeichen auf den Bildschirm, sondern setzt einfach den Cursor bei der nächsten Druckposition auf.

Die leere Eingabe wird folgendermaßen verarbeitet:

(1) Wenn der Cursor nicht am Anfang einer Zeile aufgesetzt wird und auch vom Anwender nicht aus dieser Zeile gefahren wird, so ignoriert INPUT§ die leere Eingabe. D.h., der Cursor blinkt einfach weiter, INPUT§ gibt die Kontrolle in diesem Fall also nicht an BASIC zurück.

(2) Wenn der Cursor am Zeilenanfang aufgesetzt wird, oder der Anwender die Zeile mit dem Cursor verläßt, so übergibt INPUT§ den leeren String an die INPUT§-Variable, das Programm läuft aber weiter. Dadurch haben Sie im Programm die Möglichkeit, auf die leere Eingabe gezielt zu reagieren.

(8) **INPUT§ vom Bildschirm**

OPEN 3,3 .... INPUT§3 liest den Rest einer Bildschirm-Zeile ab der aktuellen Cursorposition bis maximal zur Spalte 78, also maximal 79 Zeichen. Die letzte Spalte wird in keinem Fall gelesen.

Falls in der Zeile Doppelpunkt oder Komma enthalten sind, wird wie beim normalen INPUT ab einschließlich des Trennzeichens nicht weitergelesen. Im Unterschied zum normalen INPUT werden folgende Blanks mit übernommen, führende dagegen auch nicht. Durch Anführungszeichen können auch führende Blanks mit übernommen werden.

Der Hauptunterschied dieses INPUT§ zu allen anderen INPUT-Versionen liegt darin, daß eine Bildschirmzeile ohne Zutun des Anwenders gelesen werden kann.

5.8 GET (Hole ein Zeichen)

Zweck

GET holt genau ein Zeichen (Byte) aus einer Datei (GET§) oder aus dem Tastaturpuffer (GET) in eine Variable.

Format

GET §la, **vl** , v2 ,...

Im Gegensatz zu PRINT und INPUT hat GET§ kein eigenes Kurzzeichen, die Abkürzung Ge kann also mit oder ohne § verwendet werden.

GET§la setzt OPEN voraus.

vl ist eine beliebige Variable.

v2 ... sind weitere Variable, die jeweils durch Komma getrennt werden.

GET darf nur im Programm verwendet werden, nicht aber im Direktmodus.

5.8.1 GET aus dem Tastaturpuffer

GET bietet wie INPUT die Möglichkeit, mit dem Anwender in Dialog zu treten, allerdings auf ganz andere Art. Bei Tastaturpuffer ist beschrieben, daß jede Taste, die während eines Programmlaufes gedrückt wird, automatisch in den Tastaturpuffer übernommen wird. Maximal 10 Tasten können dort gespeichert werden. GET holt nun das zuerst im Speicher abgelegte Zeichen bzw. den ASC-Code der zuerst gedrückten Taste in die Stringvariable. Dabei bleibt aber weder das Programm stehen, noch wird dieses Zeichen automatisch auf den Bildschirm geschrieben, wie bei INPUT.

Die Hauptanwendung von GET liegt darin, einen sehr sicheren Dialog aufzubauen, da GET im Gegensatz zu INPUT keine Fehlermeldungen bringen kann. Die einzige Möglichkeit für den Anwender trotz GET das Programm zu stoppen ist die STOP-Taste.

GET kann jeden Code holen, der durch die Tastatur erzeugt werden kann, also auch die Codes der Funktionstasten. Ist der Puffer leer, wird an die Stringvariable der leere String übergeben.

Falls man auf eine Eingabe des Anwenders warten will, kann dies demnach durch folgende Abfrage geschehen:

```
10 GET G$ : IF G$ = "" GOTO 10
20 REM VERARBEITUNG VON G$
```

Das Programm läuft dann solange in 10, bis eine Taste gedrückt wurde.

Auch der umgekehrte Fall kann eintreten, daß man nämlich wissen will, ob irgendwann seit dem letzten GET eine Taste gedrückt wurde, um z.B. das Programm anzuhalten:

```
10 REM BELIEBIGE OPERATION
90 GET G$ : IF G$ = "" GOTO 10 : REM OPERATION WEITERMACHEN
95 REM ABBRUCHTASTE(N) BEHANDELN
```

In jedem Fall können Sie durch Abfragen jeden beliebigen Code herausfiltern, um ihn dann z.B. auf den Bildschirm zu drucken oder aber, um gerade diesen Code nicht zu drucken.

Zahlenvariable

Grundsätzlich kann hinter GET auch eine Zahlenvariable angegeben werden. Da aber dann bei den meisten Tasten eigenartige Fehlermeldungen erscheinen, empfehlen wir, GET nur mit Stringvariablen zu verwenden. Die Reaktionen bei GET (aus Tastaturpuffer) und Zahlenvariable sind im einzelnen:

Erlaubt sind alle Ziffern, '+', '-' und '.', Blank und 'E' (ohne SHIFT).

Bei der leeren Eingabe (kein Zeichen im Puffer), sowie bei '.', '+', '-', Blank und 'E' wird 0 an die Variable übergeben, bei den anderen 9 Ziffern der entsprechende Zahlenwert.

Bei Komma und Doppelpunkt wird EXTRA IGNORED gemeldet und bei allen anderen nichterlaubten Tasten SYNTAX ERROR. Diese Meldungen erfolgen ohne Angabe der Zeilennummer!

Durch dieses Verhalten ist der große Vorteil des GET, nämlich seine Kontrollierbarkeit durch das BASIC-Programm weg.

5.8.2 GET§ aus Dateien

GET§ holt Byte für Byte aus einer Datei. Gegenüber INPUT§ bietet GET§ den Vorteil, keinerlei Trennzeichen zu benötigen. Deshalb kann GET jeden der 256 Codes lesen, den PRINT§ erzeugen kann. Der Code 0 kann allerdings nur über einen Umweg gelesen werden:

CHR$(0)

Anstatt des Codes 0 übergibt GET§ an die Stringvariable den leeren String. Da aber in einer Datei das leere Zeichen nicht vorkommen kann, kann durch folgende Abfrage der Code 0 eindeutig rekonstruiert werden:

```
10 GET§1,G$ : IF G$ = "" THEN G$ = CHR$(0)
```

Status

Wie bei Status beschrieben ist ST 0 solange noch weitere Zeichen in der Datei stehen, geht mit dem letzten Zeichen auf 64 und dann auf 2, falls noch weitergelesen wird.

Sobald ST 2 enthält, wird an die Stringvariable bei jedem GET§ CHR$(13), also CR übergeben!

5.8.3 GET§ aus dem Bildschirm

Durch OPEN 3,3 und GET §3,G$ kann ab einer bestimmten Stelle Zeichen für Zeichen aus dem Bildschirm gelesen werden.

Revers-Zeichen, die nicht durch Anführungszeichen eingeschlossen sind, werden dabei in die entsprechenden Normal-Zeichen gewandelt. Ansonsten wird der Bildschirminhalt völlig identisch in die Stringvariable abgebildet.

Gelesen wird ab der nächsten Druckposition. Durch Cursorbewegungen kann also bestimmt werden, ab wo gelesen werden soll.

Da GET Spalte für Spalte überträgt, ohne sich um den Inhalt zu kümmern, wird auch SPACE übertragen.

5.8.4 GET§ von der Tastatur

Der Vollständigkeit halber sei erwähnt, daß GET§ auf das Gerät Nr. 0 (Tastatur) identische Wirkung zeigt mit GET.

5.9 Status (Zustand von Peripheriegeräten / Dateien)

Die Variable ST (Status) enthält Information über den Zustand der Datei. Er wird bei jedem E/A-Befehl neu in ST übertragen. ST stellt daher eigentlich eine Funktion in Gestalt einer Variable dar.

Die Bedeutung der 8 Bits des Status, sowie ihr Dezimalwert sind in der folgenden Tabelle zusammengefaßt:

Bit	Dezimal	IEC-Bus	Rekorder
0	1	Zeitüberschr. b. Schreiben	-
1	2	Zeitüberschr. b. Lesen	-
2	4	-	zu kurzer Block
3	8	-	zu langer Block
4	16	-	nicht korrigierbarer Lesefehler
5	32	-	Prüfsummenfehler
6	64	Dateiende (EOI)	Dateiende (EOF)
7	-128	DEVICE NOT PRESENT	Bandende (EOT)

Die wichtigste Statusmeldung ist 'Dateiende' (EOI/EOF). Ehe sie aber ausführlich erklärt wird, sollen noch kurz die anderen Meldungen vorgestellt werden.

Beim Band bedeuten die Bits 2,3,4,5 einfach Lesefehler, die durch schlechtes Band oder schlechte Kopfjustierung zustande kommen.

Wenn Bit 7 gesetzt ist, wird entweder FILE NOT FOUND (Band) oder DEVICE NOT PRESENT ERROR (IEC-Bus) gemeldet.

FILE NOT FOUND = Datei wurde nicht gefunden
DEVICE NOT PRESENT = Gerät ist nicht vorhanden

5.9.1 Dateiende

In manchen Fällen wissen Sie nicht, wieviele Daten eine Datei enthält, wie oft Sie also mit INPUT oder GET lesen müssen, um alle Daten zu holen. Sie müssen dann mit Hilfe des Status feststellen, wann die Datei zu Ende ist.

Merken Sie sich dazu die Regel, daß der Status dann den Wert 64 enthält, wenn das letzte Byte der Datei gelesen wurde. Versuchen Sie dann noch weiterzulesen, wird Bit 1 (Wert 2) des Status gesetzt und vom Betriebssystem CR (Carriage Return) geschickt.

Sehen Sie sich dazu folgendes Beispiel an:

```
10 OPEN 2,8,2,"0:BSP1,W"
20 PRINT§2,1
25 PRINT§2,2
30 CLOSE 2
110 OPEN2,8,2,"0:BSP1,R"
120 GET§2,G$:EN=ST
130 PRINT ST,ASC(G$),G$
140 IF EN=0 GOTO 120
150 CLOSE 2
210 OPEN2,8,2,"0:BSP1,R"
220 INPUT§2,G$:EN=ST
230 PRINT ST,G$
240 IF EN=0 GOTO 220
250 CLOSE 2
```

In Zeile 10 wird eine Floppy-Datei zum Schreiben geöffnet. Die Zeilen 20 und 25 schreiben die Zahlen 1 und 2 in die Datei, die dann in 30 geschlossen wird.

In den Zeilen 110-150 wird die Datei mit GET gelesen und für jedes Zeichen der Status, der ASC-Code des Zeichens und das Zeichen selbst auf den Bildschirm gedruckt.

In den Zeilen 210-250 wird die Datei mit INPUT gelesen und für jede Zahl der Status und die Zahl auf den Bildschirm gedruckt.

Der Ausdruck sieht folgendermaßen aus (wir haben zusätzlich die Zeilen numeriert):

Zeile	Status	ASC	Zeichen
1	0	32	Blank
2	0	49	1
3	0	32	Blank
4	0	13	CR
5	0	32	Blank
6	0	50	2
7	0	32	Blank
8	64	13	CR
9	0		1
10	64		2

In Zeile 8 sehen Sie, daß der Status gleichzeitig mit dem letzten Byte der Datei auf 64 gesetzt wird. Zeile 10 zeigt, daß bei INPUT der Status mit der letzten Zahl auf 64 geht.

Daraus läßt sich folgendes 'Kochrezept' ableiten:

(1) Unmittelbar nach dem Lesen eines Datums weisen Sie den Wert von ST an eine andere Variable zu. Dies ist immer dann erforderlich, wenn vor der Abfrage dieses Wertes eine E/A-Operation kommt, also z.B. ein PRINT auf den Bildschirm, da der Status durch jede E/A-Operation neu gesetzt wird!

(2) Danach verarbeiten Sie das eben geholte Zeichen (GET), bzw. die Zahl oder den String (INPUT).

(3) Ehe Sie das nächstemal GET oder INPUT rufen, fragen Sie den gemerkten Status-Wert ab. Wenn er 0 ist, können Sie weitermachen, ansonsten ist die Datei zu Ende.

5.9.2 Zeitüberschreitung

Um zu verhindern, daß der Rechner sich an einem langsamen IEC-Bus-Teilnehmer 'aufhängt', bricht das Betriebssystem nach 65 ms den E/A-Versuch ab. Der Abbruch wird dann im Status vermerkt, indem Bit 0 (PRINT) oder 1 (INPUT, GET) gesetzt wird.

Wünschen Sie nicht, daß das Betriebssystem abbricht, können Sie durch

```
POKE 1020, 128
```

erreichen, daß solange gewartet wird, bis sich das Peripheriegerät meldet. Meldet sich das Gerät allerdings nicht, kann nur von Hand durch die STOP-Taste unterbrochen werden, vom Programm aus besteht keine Möglichkeit dazu!

Bei Commodore-Geräten kann die Zeitüberschreitung nur im Fehlerfall auftreten. Wir wollen zwei typische Fehlersituationen schildern:

5.9.3 Gerät nicht angeschlossen

Das angesprochene Gerät ist nicht eingeschaltet oder gar nicht angeschlossen. Falls ein anderes Gerät angeschlossen und eingeschaltet ist, wird nicht DEVICE NOT PRESENT ERROR gemeldet.

Vielmehr treten bei INPUT und PRINT zwei verschiedene Zustände im Status auf: INPUT bzw. GET bewirkt die Meldung 'Zeitüberschreitung beim Lesen' (Status enthält den Wert 2) und das Betriebssystem schickt bei jedem INPUT bzw. GET ein CR (s. o.)

Bei PRINT wird dagegen Bit 7 im Status (Wert -128) gesetzt, was tatsächlich DEVICE NOT PRESENT bedeutet.

5.9.4 Floppy-Datei geschlossen

Durch Programmier- oder Bedienungsfehler, die im Floppy-Handbuch beschrieben sind, kann es vorkommen, daß eine Floppy-Datei geschlossen ist, obwohl die zugeordnete logische Datei des Rechners offen ist.

In diesem Fall meldet der Status bei INPUT und GET wie oben den Wert 2. Bei PRINT dagegen wird der Wert -127 gemeldet, was bedeutet, daß Bit 0 und Bit 7 gesetzt sind. Der Status meldet also 'Zeitüberschreitung beim Schreiben', was richtig ist und gleichzeitig DEVICE NOT PRESENT, was falsch ist.

5.9.5 Abfrage des Status-Wertes

Sie haben gesehen, daß der Status-Wert nicht als eine Zahl, sondern als 8 Bits zu interpretieren ist. Wollen Sie prüfen, ob der Status den Wert 64 enthält, ob also Bit 6 gesetzt ist, ist das nicht gleichbedeutend damit, daß Sie fragen, ob der Status gleich 64 ist:

```
IF 64 AND ST THEN          richtig
IF ST = 64 THEN            falsch
```

Am Beispiel -127 können Sie sehen, daß die Abfrage mit '=' schief geht:

```
IF -128 AND ST OR 1 AND ST THEN ...
```

Diese Abfrage prüft, ob Bit 0 oder Bit 7 gesetzt ist!

Wenn es Ihnen jetzt zu kompliziert geworden ist, lesen Sie vielleicht noch folgendes Rezept:

Von Spezialfällen und Spezialgeräten abgesehen, kann man die Statusabfrage folgendermaßen aufbauen:

Unmittelbar nach INPUT oder GET fragen Sie ab, ob der Status ungleich 0 ist und wenn ja, springen Sie zur Statusbehandlung. Ist er nicht 0, kommen Sie in die normale Datenbehandlung.

In der Statusbehandlung stellen Sie fest, ob 'Dateiende' aufgetreten ist und setzen in diesem Fall den Endeschalter EN auf ungleich 0.

Wenn nicht, laufen Sie in die Fehlerbehandlung, die hier nur angedeutet ist.

```
10 INPUT§LA,IN$ : IF ST GOTO 100
20 REM DATENBEHANDLUNG
.
.
.
50 IF EN = 0 GOTO 10
60 CLOSE LA : REM ENDE DER DATEIBEHANDLUNG
.
.
100 IF ST AND 64 THEN EN = 64 : GOTO 20
110 ?"FEHLER" : STOP
```

6. Sprunganweisungen

6.1 Einführung

Das BASIC-Programm wird normalerweise Zeile für Zeile abgearbeitet. Sehr oft braucht man aber Anweisungen, die den Programmablauf ändern, indem zu anderen Zeilen als zur nächsten 'gesprungen' wird. Wir unterscheiden solche Sprünge, die in jedem Fall ausgeführt werden (unbedingte Sprünge) und solche, die abhängig von Bedingungen ausgeführt werden (bedingte Sprünge). Eine andere Unterscheidung kann danach gemacht werden, ob nach einem Sprung das Programm ein 'Unterprogramm' ausführt und danach an der Stelle weitermacht, von wo aus in das Unterprogramm gesprungen wurde, oder ob einfach an eine andere Stelle gesprungen wird, ohne daß das Programm wieder an die alte Stelle zurückkehrt.

Der einfache unbedingte Sprung wird von GOTO ausgeführt, der Aufruf eines Unterprogramms von GOSUB.

Beide Befehle gibt es auch in bedingter Form als 'Sprungverteiler' (ON ..).

Schließlich existiert natürlich die wichtigste Form der Entscheidung, nämlich die 'Wenn, dann .... ,sonst ...' (IF ... THEN ...).

Zwei Befehle, die ein Programm beenden bzw. unterbrechen werden auch unter dieser Überschrift behandelt, da sie einen Sprung aus dem BASIC-Programm zurück zum Direktmodus (Kommandomodus) darstellen (END, STOP).

Weil sie sonst ganz alleine dastehen würde, wurde die REM-Anweisung auch als Sprunganweisung angesehen, da sie eine Bemerkung überspringt.

Ehe Sie die folgenden Befehlsbeschreibungen lesen, erinnern Sie sich bitte daran, daß BASIC-Zeilennummern nur als Konstante existieren können und daß in einer BASIC-Zeile mehrere Anweisungen durch Doppelpunkt getrennt stehen können.

Hinweis für Fortgeschrittene

Wir wollen kurz erklären, wie intern ein Sprungbefehl behandelt wird, weil sich daraus Regeln für die Anordnung von Sprüngen ableiten lassen, um möglichst kurze Ausführungszeiten zu erhalten.

Die einzelnen BASIC-Zeilen sind durch sogenannte Vorwärtszeiger miteinander verknüpft. D.h. am Anfang jeder Zeile steht für den Interpreter in zwei Bytes die Information, wo die nächste BASIC-Zeile beginnt. Dadurch muß bei Sprüngen nicht die ganze Information aller übersprungenen Zeilen gelesen werden, sondern der Interpreter 'hangelt' sich bei Sprüngen über die Vorwärtszeiger von Zeilenanfang zu Zeilenanfang, bis er die richtige Zeile gefunden hat.

Nun 'weiß' der Interpreter aber, ob die Zielzeile eine kleinere oder größere Zeilennummer hat, als die Zeile, in der er gerade die Sprunganweisung gefunden hat. Wenn er zu einer Zeile springen muß, die weiter hinten steht (größere Zeilennummer), so sucht er diese Zeile ab seiner jetzigen Position. Muß er dagegen nach vorne springen, so sucht er sofort ab dem Programmanfang.

Zeilen werden also am schnellsten gefunden, wenn sie möglichst weit vorne stehen und zwar unabhängig von der Zeile, von der aus gesprungen wird - oder wenn sie möglichst knapp hinter der aufrufenden Zeile stehen.

6.2 GOTO (Sprung zu Zeile)

Zweck

Unbedingter Sprung zu der angegebenen Zeile.

Format

GOTO bz

bz BASIC-Zeilennummer

Anmerkungen

GOTO sollte die letzte Anweisung in einer BASIC-Zeile sein, da etwaige Anweisungen dahinter nie ausgeführt werden.

Existiert die hinter GOTO angegebene Zeilennummer nicht, so ist die Folge ein UNDEF'D STATEMENT ERROR.

GOTO kann auch - ähnlich wie RUN - zum Starten eines Programms verwendet werden. Lesen Sie dies bitte bei RUN nach.

Beispiel

```
10 A = A + 1
20 PRINT A
30 GOTO 10
```

In diesem Beispiel wird durch GOTO eine sogenannte 'Endlosschleife' aufgebaut, die erst nach sehr langer Zeit durch OVERFLOW ERROR zum Stillstand käme, falls man sie nicht vorher durch STOP unterbricht.

6.3 GOSUB/RETURN (Aufruf eines Unterprogrammes)

Zweck

Unbedingter Unterprogramm-Sprung zu der angegebenen Zeile. Beim ersten RETURN wird unmittelbar hinter dem GOSUB mit der Programmabarbeitung weitergemacht.

GOTO **SUB**ROUTINE = Springe zum Unterprogramm
RETURN (from Subroutine) = kehre (aus dem Unterprogramm) zurück

Format

GOSUB bz

bz BASIC-Zeilennummer

RETURN (nicht mit RETURN-Taste verwechseln)

Anmerkungen

Hinter GOSUB können in der gleichen Zeile weitere Anweisungen stehen, sie werden ausgeführt, sobald durch RETURN der Rücksprung aus dem Unterprogramm erfolgt ist.

Existiert die hinter GOSUB angegebene Zeilennummer nicht, so ist die Folge ein UNDEF'D STATEMENT ERROR.

Wird im Programm RETURN gefunden, ehe das zugehörige GOSUB ausgeführt wurde, wird RETURN WITHOUT GOSUB ERROR gemeldet.

Wenn der Stack nicht anderweitig benötigt wird, sind maximal 26 Unterprogrammebenen möglich, das heißt Sie können bis zu 26 mal aus einem Unterprogramm wieder ein Unterprogramm aufrufen. Bitte beachten Sie dazu die Hinweise zum Stack.

Anwendung

Unterprogramme verwendet man, um Platz zu sparen und um die Übersicht über das Programm zu erhöhen.

Kommt das gleiche Programmstück in einem Programm öfter vor, so legt man es ab einer beliebigen Zeilennummer einmal ab und schließt es durch RETURN ab. Dann kann es von beliebigen Stellen aus durch GOSUB aufgerufen werden. Beim ersten RETURN springt es automatisch an die Stelle zurück, vor der es aufgerufen wurde.

Wo sollen Unterprogramme stehen?

Prinzipiell läßt der Interpreter jede Zeilennummer als Beginn eines Unterprogrammes zu. Aus Gründen der Übersichtlichkeit sollte man sich jedoch angewöhnen, Unterprogramme nicht wahllos im Programm zu verstreuen. Zusätzlich sind u.U. Aufrufzeiten zu beachten. Wie vorher erwähnt, wird die Zeile, die hinter GOSUB angegeben ist, entweder ab Programmbeginn gesucht, wenn sie kleiner ist als die Zeilennummer, hinter der GOSUB steht, oder ab der GOSUB-Zeile, wenn die gesuchte Nummer größer ist.

Für sehr häufig aufgerufene Unterprogramme empfiehlt sich deshalb der Programmbeginn als Standort, weil sie dort unabhängig von der Aufrufstelle sehr schnell gefunden werden.

Große, aber selten gerufene Unterprogramme sollten am Programmende stehen.

Beachten Sie bei Geschwindigkeitsüberlegungen, daß die Zeit für den Rücksprung (RETURN) gegenüber dem Hinsprung (GOSUB) vernachlässigt werden kann.

Wenn Sie Unterprogramme am Programmanfang ablegen, denken Sie bitte daran, daß sie mit GOTO übersprungen werden müssen, falls das Programm nur mit RUN gestartet wird, was meistens der Fall ist. Würde im folgenden Beispiel nicht vor Zeile 20 die Anweisung GOTO 100 stehen, so wäre nach RUN ein RETURN WITHOUT GOSUB ERROR IN 29 die Folge.

```
10 GOTO 100
20 REM UNTERPROGRAMM 1
.
29 RETURN
30 REM UNTERPROGRAMM 2
.
.
100 REM BEGINN DES HAUPTPROGRAMMS
```

Wenn Sie Unterprogramme am Programmende ablegen, müssen Sie darauf achten, daß vor den Unterprogrammen das Hauptprogramm durch END beendet wird, da es sonst wieder zum RETURN WITHOUT GOSUB ERROR kommt.

Globale Variable

Für Programmierer, die andere höhere Programmiersprachen kennen, sei erwähnt, daß BASIC nur globale Variable kennt, also nicht für Unterprogramme neue Variablen mit gleichen Namen verwalten kann, die trotzdem verschiedene Speicherplätze kennzeichnen.

GOSUB im Direktmodus

Falls Sie zu Testzwecken ein Unterprogramm direkt aufrufen wollen, können Sie dies mit GOSUB anstatt GOTO oder RUN machen. Der Vorteil ist, daß Sie dann am Ende nicht RETURN WITHOUT GOSUB ERROR bekommen, wie bei RUN und GOTO. Da diese Fehlermeldung aber in diesem Fall keine Konsequenz hat, hat diese Möglichkeit des GOSUB nur akademischen Wert.

GOSUB / RETURN ist schneller als GOTO / GOTO

Da die Zeit für den Rücksprung (RETURN) vernachlässigt werden kann, ist ein GOSUB immer schneller als zwei GOTO's.

6.4 END / STOP (Programm-Ende / -Unterbrechung)

Zweck

END und STOP beenden den Programmlauf. Sie werden in jedem Fall ausgeführt, sind also unbedingte Sprünge zurück in den Direktmodus. STOP meldet die Programmzeile, in der der Abbruch erfolgte, END dagegen nicht. In beiden Fällen kann mit CONT die Programmausführung fortgesetzt werden.

Format

END
STOP

Anwendung

Wie erwähnt, haben beide Befehle die gleiche Wirkung, wenn man von der Meldung auf dem Bildschirm absieht. Prinzipiell sollte man aber wegen der Meldung folgende Anwendungen trennen:

END

END steht am logischen Ende des Programms. Das logische Ende muß keineswegs gleichzeitig die letzte Programmzeile sein. In einem Programm können ohne weiteres mehrere END vorkommen. Wenn die letzte Programmzeile gleichzeitig das (einzige) logische Ende des Programms ist, kann END entfallen.

Die Meldung nach END lautet einfach READY.

STOP

STOP setzt man üblicherweise während der Testphase ein. Die zweite Möglichkeit sind Fehlerausgänge im Programm, von denen man annimmt, daß sie aufgrund von vorhergehenden Prüfungen nie angesprungen werden, die aber dennoch logisch möglich sind. Diese STOPs beläßt man dann im Programm, um wenigstens den Fehler lokalisieren zu können, falls er wider Erwarten (oder nach Änderungen) auftreten sollte.

STOP bewirkt die Meldung: BREAK IN ... / READY.

Wird die STOP-Taste während des Programmlaufes gedrückt, ist die Wirkung identisch mit einem programmierten STOP-Befehl. Der Unterschied liegt nur darin, daß die Taste im Allgemeinen an unbekannter Programmstelle unterbricht.

Beide Anweisungen verändern nichts am Programm und seinen aktuellen Daten. Insbesondere bleiben auch Schleifen- und Unterprogrammstrukturen erhalten.

6.5 REM (Anmerkung)

Zweck

REM (Remark) erlaubt Anmerkungen ins BASIC-Programm zu schreiben, die vom Interpreter ignoriert werden.

Format

REM text

text: Alle Buchstaben ohne SHIFT und Sonderzeichen. Sollen auch SHIFT-Buchstaben eingegeben werden, muß der Text in Anführungszeichen geschrieben werden.

Anmerkungen

REM kann hinter anderen Anweisungen stehen.

Geben Sie SHIFT-Buchstaben ohne Anführungszeichen ein, so werden sie bei LIST als Befehle oder Fehlermeldungen dekodiert.

REM verbraucht Speicherplatz (1 Byte pro Textzeichen) und senkt die Ausführungszeit. In zeit- und platzkritischen Programmen sollten deshalb keine REMs vorkommen.

6.6 IF...THEN... (Wenn...dann...)

Zweck

Dieser sehr wichtige Befehl erlaubt die Verzweigung in zwei verschiedene Programmteile abhängig von einer Bedingung.

Format

IF logischer Ausdruck THEN beliebige Anweisungen

IF logischer Ausdruck THEN Zeilennummer

IF logischer Ausdruck GOTO Zeilennummer

logischer Ausdruck (s. logische Ausdrücke und Operatoren)

Wirkung

Wenn der logische Ausdruck 'wahr' ergibt, werden die Anweisungen hinter THEN bzw. der Sprung zur Zeilennummer ausgeführt.

Im Fall von 'falsch' wird immer die nächste BASIC-Zeile ausgeführt.

Bitte beachten Sie, daß 'falsch' einfach durch den Zahlenwert 0 repräsentiert wird und 'wahr' durch jeden anderen Wert.

Anmerkungen

IF ... **THEN** 10 und IF ... **GOTO** 10 haben identische Wirkung, aber auch IF ... **THEN GOTO** 10 ist nicht falsch, aber überflüssig.

Ausgehend von allgemeinen Programmstrukturen unterscheidet man **einseitige** und **zweiseitige** Sprünge. In anderen Sprachen werden zweiseitige Sprünge durch IF ... THEN ... ELSE unterstützt. In den folgenden Beispielen wird gezeigt, daß jede denkbare Struktur auch mit den Ihnen zur Verfügung stehenden Mitteln gebildet werden kann, also auch ohne ELSE.

Alle vier Lösungen gehen von der Beispiels-Problemstellung aus, daß der Nenner einer Division aufgrund der Eingabe (Vorgeschichte) 0 sein kann, aber bei der Division nicht 0 sein darf, weil sonst DIVISION BY ZERO ERROR gemeldet wird.

(1) einseitig

```
10 INPUT"ZAEHLER";Z
20 INPUT"NENNER ";N
30 IF N=0 THEN N=1E-20        ja: Anweisung
40 Q=Z/N                      gemeinsam
50 ?"Quotient="Q
```

(2) zweiseitig, aber 'ja'-Zweig hinter THEN

```
10 INPUT"ZAEHLER";Z
20 INPUT"NENNER ";N
30 IFN=0THENQ=1E20:GOTO50 ja: Anweisung(en) und Sprung
40 Q=Z/N                  nein
50 ?"Quotient="Q          gemeinsam
```

(3) zweiseitig mit völlig getrennten Zweigen

```
10 INPUT"ZAEHLER";Z
20 INPUT"NENNER ";N
30 IF N=0 GOTO 45         Sprung zu ja
40 Q=Z/N : GOTO 50        nein: Sprung zu gemeinsam
45 Q=1E20                 ja
50 ?"Quotient="Q          gemeinsam
```

(4) Rückwärtssprung (Schleife)

```
10 INPUT"ZAEHLER";Z
20 INPUT"NENNER ";N
30 IF N=0 GOTO 20         ja: Rückwärtssprung (Schleife)
40 Q=Z/N                  nein
50 ?"Quotient="Q
```

Im ersten Fall wird im ja-Fall eine Anweisung ausgeführt, die im nein-Fall fehlt. Der nein-Fall ist also gleichzeitig der gemeinsame Teil der beiden Fälle. Dies ist kennzeichnend für die einseitige Struktur.

Im zweiten Fall wird nach der Anweisung des ja-Falles die Anweisung des nein-Falles übersprungen, so daß zwei getrennte Zweige entstehen. Damit hat man im Prinzip eine zweiseitige Struktur realisiert. Allerdings ist für den ja-Fall nicht beliebig viel Platz, da ja die Zeile hinter THEN nach einigen Anweisungen voll ist.

Dies wird im dritten Fall dadurch umgangen, daß im ja-Fall sofort auf einen eigenen ja-Teil gesprungen wird. Nach der Abfrage kommt wie immer der nein-Teil, der durch einen Sprung über den ja-Teil zum gemeinsamen Teil abgeschlossen wird. Dadurch ist eine echte und unbegrenzte zweiseitige Struktur realisiert, da beide Fälle jetzt bliebig viele Anweisungen umfassen können.

Im vierten Fall wird im ja-Fall ein Rückwärtssprung ausgeführt, der in diesem Fall zu einer Schleife führt, weil das Programm solange 20 und 30 durchläuft, bis ein von 0 verschiedener Wert eingegeben wird.

Rundungsfehler

Bei Zahlenvergleichen kann es vorkommen, daß das Ergebnis 'falsch' ist, obwohl es nach den Werten, wenn man sie auf den Bildschirm drucken läßt, 'richtig' lauten müßte. Wie bei Rundungsfehler beschrieben, liegt dies an der angefangenen 10-ten Stelle, die intern dargestellt werden kann, aber durch PRINT nicht ausgegeben wird.

Allgemein kann gesagt werden, daß es immer sicherer ist, eine Abfrage mit 'kleiner' bzw. 'größer als' durchzuführen, als mit 'gleich' bzw. 'ungleich'. Dies gilt natürlich nicht für Abfragen an Strings.

Dazu soll noch ein Beispiel angeführt werden. Angenommen, Sie erwarten nach einer umfangreichen Berechnung einen Wert E, der gleich sein soll dem Wert in der Variablen A, um eine Abfrage 'wahr' werden zu lassen. Wenn Ihnen nun z.B. aufgrund der Problemstellung bekannt ist, daß eine Abweichung von +/- 1E-5 statthaft ist, sollten Sie die Abfrage wie folgt formulieren:

IF E größer als A-1E-5 AND E kleiner als A+1E-5 THEN ...

Sie haben dadurch alle Werte zugelassen, die in der angegebenen (.00001) 'Umgebung' vom exakt erwarteten Wert liegen.

Abkürzungen

Gerade bei Zeilen mit Abfragen kann es sinnvoll sein, die BASIC-Anweisungen in abgekürzter Form einzugeben, um gerade noch eine Struktur nach (1) oder (2) zu erreichen. Beim Ändern einer solchen überlangen Zeile müssen Sie dann nochmal alle Abkürzungen tippen und die Reste der Anweisungen mit DEL löschen.

Zeitüberlegungen für Fortgeschrittene

Grundsätzlich können in einer Zeile mehrere Abfragen stehen. Da die Anweisung hinter der letzten Abfrage nur dann ausgeführt wird, wenn alle Abfragen dieser Zeile mit 'wahr' beantwortet wurden, handelt es sich um eine Und-Verknüpfung, ist also logisch äquivalent durch AND zu erreichen:

```
IF A=B THEN IF C=D THEN ...
ist logisch gleichwertig mit
IF A=B AND C=D THEN
```

Folgendes Testprogramm liefert die nachstehend aufgeführten Zeitwerte (in 1/60 s bei 1000 Durchläufen):

```
10 T=TI
20 FOR I = 1 TO 1000
30 IF 1=2 AND 3=4 THEN NEXT : GOTO50
40 NEXT
50 ? TI-T
```

Durch Variation von 'wahr' (1=1 / 3=3) und 'falsch' (1=2 / 3=4) mit AND und THEN IF ergeben sich folgende Werte:

			AND	THEN IF
wahr	...	wahr	575	449
wahr	...	falsch	526	453
falsch	...	wahr	525	269
falsch	...	falsch	470	269

Die wichtigste Folgerung aus dieser Tabelle ist, daß THEN IF immer schneller ist, als AND!

Falls die Abfrage schon beim ersten Vergleich 'falsch' ergibt, ist die Überlegenheit von THEN IF gravierend (Zeile 3 und 4).

IF THEN ist zwischen fast 50% (Fall 3) und ca. 15% (Fall 2) schneller als AND.

6.7 ON...GOTO / GOSUB ..... (Sprungverteiler)

Zweck

Abhängig von einem Byte-Ausdruck wird zu einer von mehreren Sprungadressen verzweigt, bzw. eines von mehreren Unterprogrammen aufgerufen.

Format

ON byteausdruck GOTO sprungliste

ON byteausdruck GOSUB unterprogrammliste

byteausdruck	(s. Byte-Ausdruck)
sprungliste	durch Komma getrennte Zeilennummern
unterprogrammliste	wie Sprungliste

Wirkung

Hinter GOTO / GOSUB stehen n Zeilennummern, die (gedacht) von 1 bis n durchnummeriert sind. Wenn byteausdruck einen Wert zwischen 1 und n ergibt, wird zu der entsprechenden Zeilennummer gesprungen. Wenn byteausdruck 0 ergibt oder einen Wert größer als n, wird die gesamte ON-Anweisung ignoriert und die dahinterliegende Anweisung ausgeführt.

Bei ON..GOSUB ist die Rücksprungstelle unmittelbar hinter der ON..GOSUB-Anweisung, also hinter dem Doppelpunkt, der hinter der Sprungliste steht, bzw. in der nächsten Zeile, falls keine weitere Anweisung hinter der Sprungliste folgt. Das bedeutet, daß mehrere ON..GOSUB-Verteiler in einer Zeile stehen können.

Beispiele

```
10 INPUT A
20 ON A GOTO 40,50,60
30 PRINT"IGNORIERT" : GOTO 10
40 PRINT"40" : GOTO 10
50 PRINT"50" : GOTO 10
60 PRINT"60" : GOTO 10
```

```
10 INPUT A
20 ON A GOSUB 40 , 50 , 60 : GOTO 10
40 PRINT"40" : RETURN
50 PRINT"50" : RETURN
60 PRINT"60" : RETURN
```

In der Regel wird es nicht sinnvoll sein, Werte größer als die Anzahl der Sprungadressen bzw. den Wert 0 für 'byteausdruck' zuzulassen. Hinter ON GOTO kann man durch ein STOP leicht kontrollieren, ob fehlerhafte Sprungzeiger berechnet wurden. Bei ON GOSUB ist dieser Weg nicht möglich, weil immer hinter dem Verteiler weitergemacht wird. Hat man Bedenken wegen der Sprungzeiger muß man durch eine IF-Abfrage vor dem ON GOSUB falsche Werte ausschließen bzw. melden.

Beispiel: Simulation eines Unterprogramms

In Stack wurde erklärt, daß die Anzahl der geschachtelten Unterprogramme begrenzt ist. Hier soll nun gezeigt werden, wie der mehrmalige Aufruf eines Unterprogramms durch GOTO ersetzt werden kann, wenn der Rücksprung über ON...GOTO erfolgt.

Annahme: Sie möchten das Unterprogramm in der Zeile 1000 von den Zeilen 10, 20 und 30 aus aufrufen:

```
.
10 ... :GOSUB 1000
11 ...
.
20 ... :GOSUB 1000
21
.
30 ... :GOSUB 1000
31 ...
.
.
1000 REM UNTERPROGRAMM
.
1099 RETURN: REM ENDE VON UNTERPROGRAMM
```

Identisch wäre dann folgende Konstruktion:

```
.
10 ... :RA=1:GOTO 1000
11 ...
.
20 ... :RA=2:GOTO 1000
21
.
30 ... :RA=3:GOTO 1000
31 ...
.
.
1000 REM UNTERPROGRAMM
.
1099 ON RA GOTO 11,21,31: REM ENDE VON UNTERPROGRAMM
```

Bis auf den höheren Aufwand in BASIC und die längere Rücksprungzeit ist diese Lösung identisch mit GOSUB. Sie veranschaulicht nebenbei ganz gut die Struktur von Unterprogrammen.

Verteiler mit vielen Sprungsadressen

In eine BASIC-Zeile passen maximal ca. 20 Sprungadressen hinter einen ON-Verteiler. Falls Sie mehr Adressen benötigen, können Sie eine der nachfolgend beschriebenen Methoden anwenden.

Aufeinanderfolgende ON..GOTO-Verteiler

Bei ON GOTO können Sie sich die Eigenschaft zunutze machen, daß die Anweisung ignoriert wird, falls 'bytewert' die Anzahl der Adressen überschreitet. Wie erwähnt, funktioniert diese Methode nicht bei ON GOSUB.

Beispiel

```
10 INPUT A
100 ON A    GOTO 1000,1100,1200,1300,1400,1500,1600,1700,1800,1900
110 ON A-10 GOTO 2000,2100,2200,2300,2400,2500,2600,2700,2800,2900
120 ON A-20 GOTO 3000,....
130 STOP
```

In diesem Beispiel greift für Werte von A zwischen 1 und 10 die Sprungliste in Zeile 100. Für größere Werte läuft das Programm weiter in die Zeile 110. In dieser Zeile wird der Wert von (A-10) als Zeiger in die Sprungliste verwendet. Zeile 110 ist also zuständig für Werte zwischen 11 (1 = 11 - 10) und 20 (10 = 20 - 10). Für die nächsten 10 Werte steht die Zeile 120 usw.

Das Programm läuft auf das STOP in Zeile 130, wenn A einen zu großen Wert hat oder 0 ist.

Hierarchische Verteiler

Für Sprungadressen, die mit ON..GOSUB angesprungen werden, kann die obige Struktur nicht verwendet werden. In diesem Fall muß ein Überverteiler schon einmal eine Vorauswahl treffen und auf verschiedene Unterverteiler verweisen.

Selbstverständlich ist diese Methode auch bei ON GOTO sinnvoll. Vor allem bei großen Verteilern mit einigen zehn Adressen bietet die hierarchische Lösung einen Zeitvorteil gegenüber der oben beschriebenen.

Beispiel

```
10 INPUT A
100 ON INT((A-1)/5)+1 GOTO 110,120,130:STOP
110 ON A    GOSUB 1000,1100,1200,1300,1400:GOTO10
120 ON A-5  GOSUB 2000,2100,2200,2300,2400:GOTO10
130 ON A-10 GOSUB 3000,3100,3200,3300,3400:GOTO10
140 STOP
```

In diesem Beispiel werden pro Sprungverteiler fünf Sprungadressen eingetragen. In Zeile 100 wird vorverteilt:

- für Werte von 1 bis 5 auf Zeile 110
- für Werte von 6 bis 10 auf Zeile 120
- für Werte von 11 bis 15 auf Zeile 130

Die Berechnung der Abfragekriterien läßt sich **allgemein** formulieren:

In jede Verteilerzeile können SA Sprungadressen eingetragen werden. Die Werte von A beginnen bei 1.

```
100 ON INT((A-1)/SA)+1 GOTO 110,120,130,...
110 ON A      GOSUB 1000,1100,....
120 ON A-1*SA GOSUB 2000,2100,....
130 ON A-2*SA GOSUB 3000,3100,....
```

7. Schleifenbefehle

7.1 Einführung

Programm-Schleifen kommen immer dann vor, wenn dieselben Anweisungen fortlaufend an gleichen oder gleichartigen Daten durchgeführt werden müssen. Solche Schleifen laufen meistens, bis eine bestimmte (vorher bekannte) Anzahl von Durchläufen ausgeführt wurde. Sehr oft kommen solche Schleifen in Zusammenhang mit indizierten Variablen (Feldern) vor.

Sie können jede beliebige Art von Schleifen durch IF..GOTO realisieren. Für den speziellen Fall der Zählschleife gibt es aber spezielle Anweisungen.

Eine Zählschleife braucht immer eine Variable, die als Zähler benützt wird, sie wird als Laufvariable, Schleifenvariable oder Index bezeichnet. Beim Aufbau der Schleife muß bekannt sein, wo zu Zählen begonnen werden soll, wann das Ende erreicht ist und in welchen Schritten gezählt werden soll. Diese drei Werte nennen wir 'Anfangswert', 'Endwert' und 'Schrittweite'.

Damit sind alle Parameter des sogenannten 'Schleifenkopfes' erklärt, er hat folgendes Aussehen:

FOR laufvariable = anfangswert **TO** endwert **STEP** schrittweite

Nach dem Kopf folgt der 'Schleifenkörper', er enthält den eigentlichen Algorithmus, der für jeden Wert der Laufvariable ausgeführt werden soll.

Am (logischen) Ende des Schleifenkörpers steht die Anweisung, die die Wiederholung des Schleifenkörpers, bzw. den Aussprung aus der Schleife bewirkt. Diese Anweisung heißt **NEXT** laufvariable.

Anhand eines einfachen Beispiels sollen die Schleifenbefehle nochmal kurz erklärt werden:

```
10 FOR I = 1 TO 10 STEP 2
20 PRINT I;
30 NEXT
```

In Zeile 10 wird der Variablen I der Anfangswert 1 zugewiesen und die Schleifenverwaltung speichert den Endwert 10 und die Schrittweite 2. Dann wird zum ersten Mal der Schleifenkörper ausgeführt, der hier nur den Wert der Laufvariablen ausdruckt. NEXT erhöht den Inhalt der Laufvariablen um 2 und prüft ob der neue Inhalt größer als der Endwert ist. Da 3 noch nicht größer als 10 ist, wird der Schleifenkörper nochmal ausgeführt. Folgender Ausdruck entsteht auf dem Bildschirm:

```
1  3  5  7  9
```

Zuletzt wurde zu 9 nochmal 2 addiert. Dadurch wurde aber der Endwert überschritten und deshalb der Schleifenkörper nicht nochmal ausgeführt, sondern die nächste Anweisung nach NEXT.

Nach der detaillierten Beschreibung der einzelnen Schleifenparameter wird noch die Schachtelung von Schleifen, sowie der vorzeitige Aussprung besprochen.

7.2 Die Parameter

Die Form der Schleife (auch Laufanweisung genannt) ist:

FOR laufvariable = anfangswert TO endwert STEP schrittweite

schleifenkörper (Algorithmus)

NEXT laufvariable1 , laufvariable2

7.2.1 Laufvariable

Die Laufvariable muß eine einfache Gleitkommavariable sein. Sowohl Integervariable als auch indizierte Gleitkommavariable werden mit SYNTAX ERROR abgelehnt.

Der Wert der Laufvariablen darf im Schleifenkörper verändert werden. Nach dem Aussprung aus der Schleife bleibt er erhalten. Wenn die Schleife regulär über NEXT verlassen wurde, so enthält die Laufvariable den Endwert + Schrittweite.

Bei Schrittweiten, die Nachkommastellen enthalten oder wenn Laufwerte vorkommen, die mehr als 9 Stellen haben, treten Rundungsfehler auf. In diesem Zusammenhang verweisen wir auf Rundungsfehler und INT. Da nicht nur geringfügig zu große (1.26000001), sondern auch zu kleine Werte (1.25999999) vorkommen können, ist echte Rundung erforderlich, falls Sie glatte Zahlen benötigen. Ein Beispiel soll zeigen, wie Sie die Laufvariable auf z.B. 2 Stellen nach dem Komma runden können:

```
10 FOR I = 100 TO 999 STEP 0.24 : I=INT(I*100+0.5)/100
20 REM SCHLEIFENKOERPER
```

Durch rechtzeitige Rundung wie im Beispiel verhindern Sie, daß sich der Fehler in gefährliche Größenordnungen aufsummieren kann.

7.2.2 Anfangswert

Der Anfangswert ist ein beliebiger Gleitkommaausdruck. Die Zuweisung des Anfangswertes an die Laufvariable erfolgt vor der Berechnung des Endwertes und der Schrittweite.

Der Ausdruck für den Anfangswert darf als Sonderfall auch die Laufvariable selbst enthalten. Der Teil 'laufvariable = anfangswert' ist bis auf die Einschränkung bei der Laufvariablen eine ganz normale Zuweisung, d.h. der alte Wert der Laufvariablen lebt solange, bis die Zuweisung erfolgt ist. Das folgende Beispiel soll dies verdeutlichen:

```
10 I = 10
20 FOR I = I+1 TO 100
```

Die Schleifenverwaltung setzt in diesem Fall den Anfangswert auf 11. Diese Form der Anfangswertzuweisung in Abhängigkeit von der Laufvariablen kann u. U. hilfreich sein, wenn eine Schleife stückweise abgearbeitet wird und zwischendurch verlassen werden muß.

7.2.3 Endwert

Der Endwert ist ein beliebiger Gleitkommaausdruck. Er wird nur beim Eintritt in die Schleife berechnet und gespeichert. Der einmal berechnete Endwert kann nicht mehr verändert werden.

Sollte der Endwert-Ausdruck die Laufvariable enthalten, beachten Sie bitte, daß bereits der Anfangswert in der Laufvariablen steht, sobald der Endwert berechnet wird. Beispiel:

```
FOR I = I+1 TO I+100
```

bewirkt, daß die Schleife immer 101 mal durchlaufen wird, unabhängig von I und +1 in I+1. Beachten Sie dazu das vorhergehende Beispiel.

7.2.4 Schrittweite

Die Schrittweite ist ein beliebiger Gleitkommaausdruck. Sie wird nur beim Eintritt in die Schleife berechnet und gespeichert. Die einmal berechnete Schrittweite kann nicht mehr verändert werden.

Wird keine Schrittweite angegeben, so wird automatisch +1 angenommen.

Die Schrittweite kann auch negativ sein. Die Schleife zählt dann 'rückwärts'. Bei negativen Schrittweiten muß der Anfangswert größer als der Endwert sein!

Beispiel:

```
10 FOR I=100 TO 50 STEP -10 : PRINT I; : NEXT
```

Der Ausdruck dieses Programms auf dem Bildschirm lautet:

```
100  90  80  70  60  50
```

Obwohl die Schrittweite 0 sinnlos ist, weil sie eine Endlosschleife erzeugt, wird sie nicht durch eine Fehlermeldung verhindert!

7.3 NEXT

NEXT addiert die Schrittweite zum Inhalt der Laufvariablen und prüft, ob der Endwert überschritten ist.

Wenn nicht, wird die Programmausführung unmittelbar hinter dem entsprechenden Schleifenkopf fortgesetzt.

Wenn der Endwert überschritten ist, wird die Programmausführung hinter dem NEXT fortgesetzt.

Die Prüfung lautet bei positiver Schrittweite auf Laufvariable größer als Endwert, bei negativem Schritt auf kleiner! Für 'gleich' wird also jeweils gerade noch ein Schleifendurchlauf gemacht.

Die Prüfung wird nur bei NEXT ausgeführt. Daraus folgt, daß jede Schleife mindestens 1-mal durchlaufen wird, auch wenn beim Eintritt in die Schleife die Abbruchbedingung bereits erfüllt ist:

```
10 FOR I = 1 TO 0 : PRINT I : NEXT
```

Diese Schleife wird einmal durchlaufen.

Sollten Sie eine sogenannte 0-Schleife benötigen, müssen Sie durch eine Abfrage vor dem Eintritt in die Schleife diesen Sonderfall abfangen:

```
10 IF A g E GOTO 50
20 FOR I = A TO E

40 NEXT
50 REM PROGRAMMTEIL NACH DER SCHLEIFE
```

Angabe der Laufvariable hinter NEXT

Hinter NEXT kann eine (oder mehrere bei geschachtelten Schleifen) Laufvariable angegeben werden. Sofern man aber die Schleifen nicht geöffnet zurückläßt, braucht man die Laufvariablen nicht anzugeben. Dies hat den Vorteil, daß der Interpreter nicht prüfen muß, ob die angegebene Laufvariable wirklich zur aktuellen Schleife gehört und dadurch ca. 10% Zeit einspart!

Mehrere NEXT pro Schleife

Grundsätzlich sind mehrere NEXT pro Schleife zugelassen, da zu einem Zeitpunkt immer nur ein NEXT gefunden wird und nur das ist für den Interpreter interessant. Aus Strukturüberlegungen heraus sollte man aber nicht mehrere NEXT pro Schleife nehmen.

Für Interessierte seien trotzdem die zwei Möglichkeiten dargestellt.

```
10 FOR I = A TO E
20 .....:IF...THEN....:GOTO 50
30 .....:IF...THEN....:GOTO 50
40 ....
50 NEXT
60 REM PROGRAMM NACH SCHLEIFE
```

```
10 FOR I = A TO E
20 .....:IF...THEN....:NEXT:GOTO 60
30 .....:IF...THEN....:NEXT:GOTO 60
40 ....
50 NEXT
60 REM PROGRAMM NACH SCHLEIFE
```

Die beiden Lösungen sind identisch in der Wirkung, aber die zweite ist schneller, da innerhalb der Schleife der Sprung mit GOTO entfällt. Strukturmäßig sauberer ist allerdings die erste Lösung!

Die zweite Lösung bietet den zusätzlichen Vorteil, daß anstelle von GOTO 60 nach NEXT auch für jeden NEXT-Aussprung eine andere Zeile angesprungen werden kann. Wenn abhängig von verschiedenen Abbruchkriterien einer Schleife auch verschiedene Nachbehandlungen erforderlich sind, kann durch diese Lösung Zeit gespart werden, da eine zusätzliche Abfrage nicht mehr erforderlich ist.

7.4 Aussprung aus Schleifen

Beim Durchsuchen von Feldern nach bestimmten Werten benützt man die Zählschleife nur zum Zählen, das eigentliche Abbruchkriterium wird dagegen in einer IF-Abfrage im Schleifenkörper abgefragt. Wenn der gewünschte Wert gefunden wurde, möchte man sofort aus der Schleife springen. Es hätte ja keinen Sinn, die restlichen Werte auch noch durchzuarbeiten, nur um die einmal angefangene Schleife zum Ende zu bringen.

Ganz ohne weiteres sollten Sie aber Schleifen nicht verlassen. Ehe wir erklären, warum das so ist, wollen wir die 'saubere' Lösung vorstellen, die sicher immer funktioniert.

Wenn ein (assoziatives) Abbruchkriterium erfüllt ist, setzen Sie die Laufvariable auf den Endwert oder darüber und lassen dann ein NEXT ausführen. Dadurch wird die Schleife ordnungsgemäß geschlossen. Falls Sie den Wert der Laufvariablen benötigen, bei dem abgebrochen wurde, müssen Sie ihn retten:

```
10 FOR I = A TO E
.
20 IF A(I) g M THEN I1=I: I=E: NEXT: GOTO 50
30 NEXT
40 PRINT"NICHTS GEFUNDEN"
50 PRINT"NAECHSTER WERT IST "A(I1)
.
.
```

Im Beispiel wird das Feld A() nach einem Wert durchsucht, der größer als M ist. Trifft diese Bedingung zu, soll die Schleife verlassen werden. Der aktuelle Index wird in I1 gerettet, dann wird der Index auf den Endwert gesetzt und die Schleife durch NEXT beendet. In 50 wird der gefundene Wert weiterbehandelt.

An diesem Beispiel läßt sich auch zeigen, daß es von Vorteil sein kann, wenn mehrere Aussprünge aus der Schleife vorhanden sind. Falls nämlich kein Wert gefunden wird, der der Bedingung genügt, läuft das Programm hinter dem NEXT in Zeile 30 weiter. Zeile 40 deutet dann die Behandlung dieses Sonderfalles an.

7.5 Schleife und Stack

Wie in Stack beschrieben, belegt jede Schleife Platz im Stack. Dieser Platz wird sofort freigegeben, wenn eine Schleife beendet wird, indem bei NEXT die Überschreitung des Endwertes festgestellt wird.

Verlassen Sie dagegen die Schleife durch IF...GOTO, weiß die Schleifenverwaltung nichts davon und die Schleifendaten bleiben im Stack. Falls später eine Schleife mit der gleichen Laufvariable eingerichtet wird, werden die alten Daten dieser Schleife entfernt.

Richten Sie aber im Lauf des Programms immer wieder Schleifen mit anderen Laufvariablen ein, werden Sie sehr bald OUT OF MEMORY ERROR erhalten.

Deshalb nochmal der Hinweis auf den vorhergehenden Abschnitt. Wenn Sie sich die dort beschriebene Lösung unbedingt 'ersparen' wollen, müssen Sie die Mechanismen des Stack genau verstanden haben und auch Ihr Programm gut überblicken.

7.6 Schachtelung

Z. B. bei der Behandlung von mehrdimensionalen Feldern kommen geschachtelte Schleifen vor. Bei der Schachtelung von Schleifen muß nur beachtet werden, daß zum zuletzt durchgeführten FOR-Statement zuerst das entsprechende NEXT im Programm erscheinen muß. Wie der Ausdruck Schachtelung andeutet, müssen die Schleifen vollständig ineinander stehen, so wie die Schachtel in der Schachtel nur ganz enthalten sein kann, aber nicht teilweise.

Im Beispiel soll ein zweidimensionales Gleitkommafeld existieren, dessen Werte ausgedruckt werden sollen:

```
10 DIM A(LZ,LS)
.
100 FOR Z=0 TO LZ
120 FOR S=0 TO LS
130 PRINT "A("Z","S") = "A(Z,S)
140 NEXT:NEXT
```

Die äußere Schleife läuft über die Spalten, die innere über die Zeilen der Matrix. Die innere Schleife läuft 'schneller', d.h. nur jedesmal, wenn die innere Schleife 'erschöpft' ist, zählt die äußere die Zeile um eins weiter und startet dann wieder die innere Schleife.

Geschachtelte Schleifen können also als Zählwerk betrachtet werden, bei dem jede Stelle eine andere Anzahl von Werten annehmen kann.

Geschachtelte Schleifen benötigen verschiedene Laufvariablen. Sollten Sie aus Versehen in einer inneren Schleife nochmal die gleiche Variable verwenden, so wird beim Eintritt in die innere Schleife nichts gemeldet, die äußere wird aber aus dem Stack genommen. Bei dem NEXT, das eigentlich zur äußeren Schleife gehört, wird dann NEXT WITHOUT FOR ERROR gemeldet.

7.7 Ausführungszeit

Der Sprung vom NEXT zum Anfang des Schleifenkörpers erfolgt ungleich schneller als ein GOTO. Die Zeit ist vergleichbar mit der bei RETURN.

Versuchen Sie innerhalb von Schleifen möglichst nur Vorwärtssprünge zu haben, sofern Sie überhaupt GOTO benötigen. Rückwärtssprünge brauchen normalerweise wesentlich länger.

Gerade innerhalb von Schleifen ist es ratsam, auf möglichst viele Anweisungen pro Zeile zu achten, Blanks zu vermeiden und REMs wegzulassen, weil jede Verzögerung sich mit der Anzahl der Durchläufe multipliziert. Vor allem bei mehrfach geschachtelten Schleifen können Sie durch Zeitoptimierungen den Programmablauf wesentlich beschleunigen.

7.8 NEXT WITHOUT FOR ERROR

Sobald ein NEXT gefunden wird, dem kein FOR mehr zugeordnet ist, wird NEXT WITHOUT FOR ERROR gemeldet. Wenn Sie bei NEXT nicht die Laufvariable angegeben haben, ist manchmal nicht klar, welches FOR eigentlich nicht existiert. Falls Sie Zweifel haben, sollten Sie wenigstens während der Testphase doch die Laufvariable hinter NEXT schreiben und sei es nur deshalb, daß Sie sich selbst klar werden müssen, welches FOR zu dem NEXT gehört.

8. Dimensionierung von indizierten Variablen

8.1 Einführung in indizierte Variablen

Indizierte Variable ist der Überbegriff für eindimensionale Felder (Listen, Vektoren), zweidimensionale Felder (Tabellen, Matrizen) und mehrdimensionale Felder.

Wenn Ihnen diese Begriffe geläufig sind, können Sie bei 8.2 weiterlesen.

Eindimensionale Felder

Angenommen, Sie haben eine Liste mit den Namen Ihrer Kunden. Diese Liste wollen Sie mit dem Computer bearbeiten. Dann beantragen Sie beim Interpreter ein Feld mit sovielen Elementen, wie Sie Kunden haben. Danach lesen Sie in einer Schleife alle Namen vom Bildschirm ein und speichern die Namen in fortlaufenden Elementen ab. Das entsprechende Programm sieht folgendermaßen aus:

```
10 INPUT"ANZAHL KUNDEN";AK
20 DIM KN$(AK)
30 FOR K = 1 TO AK
40 PRINT "KUNDE NR"K;:INPUT KN$(K)
50 NEXT
```

Zeile 10 fragt nach der Anzahl der Kunden, damit das Feld in 20 entsprechend vereinbart werden kann. Die folgende Schleife verlangt nun einen Kundennamen nach dem anderen. Am Ende stehen alle Kundennamen in dem Feld KN$()

Sie sehen, daß man auf ein Feld zugreift, indem man den Index (die Nummer) des Elementes in Klammern dahinter angibt. Der Index in der Klammer kann wie im Beispiel in einer Schleife weitergezählt werden. Die große Leistungsfähigkeit Ihres Computers in der Datenverarbeitung zeigt sich vor allem bei der Abarbeitung von Feldern in Schleifen. An diesem kleinen Beispiel sehen Sie, daß es dem Programm gleich ist, ob Sie zehn oder tausend Namen eingeben wollen. Am Programm ändert sich dadurch nichts.

Zweidimensionale Felder

Bauen wir das Beispiel von vorhin noch etwas aus. Wenn Sie auch noch Vornamen und Adressen 'erfassen' wollen, vereinbaren Sie ein Feld mit 3 Spalten und soviel Zeilen, wie Kunden vorhanden sind.

```
10 INPUT"ANZAHL KUNDEN";AK
20 DIM KN$(AK,3)
30 FOR K = 1 TO AK
40 PRINT "KUNDE NR"K
50 PRINT "VORNAME ";:INPUT KN$(K,1)
60 PRINT "NACHNAME";:INPUT KN$(K,2)
70 PRINT "ADRESSE ";:INPUT KN$(K,3)
80 NEXT
```

In diesem Beispiel werden die drei Angaben pro Kunde im Programm durch getrennte Eingaben verlangt. Im nächsten Beispiel soll dies durch eine Doppelschleife allgemeiner formuliert werden. Da wir aber trotzdem auf die einzelnen Hinweise zu den Eingaben nicht verzichten wollen, müssen wir diese Hinweise auch schleifengerecht' machen. Dazu lesen wir sie aus DATA in ein eigenes Feld.

```
 5 GOSUB 1000
10 INPUT"ANZAHL KUNDEN";AK
20 DIM KN$(AK,AS)
30 FOR K = 1 TO AK
40 PRINT"KUNDE NR"K
50 FOR S=1 TO AS :PRINTH$(S);:INPUT KN$(K,S): NEXT
80 NEXT
90 END
1000 AS=3:DIM H$(AS)
1010 FOR I=1 TO AS: READ H$(I): NEXT
1050 RETURN
1090 DATA VORNAME,NACHNAME,ADRESSE
```

8.2 DIM (Felder dimensionieren)

Zweck

DIM vereinbart für die nachfolgenden Feldnamen die angegebene Anzahl Dimensionen und Elemente pro Dimension.

Format

DIM variablenname (anzahl elemente , anzahl elemente) , variablenname ...

variablenname	beliebiger Name, alle drei Typen sind möglich
anzahl elemente	Integer-Ausdruck, gibt für jede Dimension einzeln die Anzahl der Elemente an

Pro Feld können mehrere Dimensionen angegeben werden (durch Komma getrennt). Pro DIM können mehrere Felder dimensioniert werden (durch Komma getrennt).

Variablenname

Mit Ausnahme von FN, IF, ON, OR und TO sind alle Namen aller drei Typen zugelassen. Auch wenn ein Name schon eine einfache Variable definiert, kann er trotzdem unabhängig davon noch ein Feld kennzeichnen, allerdings nur ein Feld eines Typs.

Anzahl Dimensionen

Die Anzahl der Dimensionen wird entweder von der Zeilenlänge oder vom Speicherplatz begrenzt, meistens aber schon vorher vom Verständnis. Die theoretische Anzahl aufgrund der Feldverwaltung liegt bei 255.

Anzahl Elemente

Anzahl der Elemente ist im Prinzip ein positiver Integer-Ausdruck.

Die Anzahl der Elemente ist praktisch nur durch den Speicherplatz begrenzt, theoretisch könnten 32767 Elemente pro Dimension angelegt werden.

Für jede Dimension kann die Anzahl der Elemente unabhängig von den anderen Dimensionen festgelegt werden.

Reserviert werden immer die angegebene Anzahl + 1, da immer das Element mit der Nummer 0 mitvereinbart wird.

Dies ist bei mehrdimensionalen Feldern zu beachten. Angenommen, Sie verlangen ein Feld mit DIM A(10,10,10), dann werden in jeder Richtung 11 Elemente angelegt, was insgesamt 11*11*11 = 1331 Elemente ausmacht. Wenn Sie nun die 0-ten Elemente nicht nutzen, weil Ihr Algorithmus ab 1 zählt, haben Sie 331 Elemente zuviel vereinbart, was 331*5 = 1655 verschwendete Bytes bedeutet! Daraus folgt, daß es sich bei mehrdimensionalen Feldern lohnt, die Algorithmen anstatt von 1 bis n lieber von 0 bis n-1 zählen zu lassen und entsprechend bei DIM nur n-1 Elemente anzugeben.

Automatische Dimensionierung

Sprechen Sie ein Feldelement an, ohne vorher das entsprechende Feld mit DIM dimensioniert zu haben, so werden automatisch für jede Dimension 11 Elemente (von 0 bis 10) angelegt.

Diese Dimensionierung kann genausowenig wie die explizite mit DIM rückgängig gemacht werden. Vor allem bei mehrdimensionalen Feldern läuft die automatische Dimensionierung meistens auf eine gewaltige Platzverschwendung hinaus, da man selten pro Dimension genau 11 Elemente braucht. Wir empfehlen daher, Felder grundsätzlich mit DIM zu 'beantragen'.

Platzbedarf von Feldern

Der überschlägige Platzbedarf ergibt sich aus der Anzahl der Elemente multipliziert mit

2 für Integerfelder
3 für Stringfelder (zuzüglich Inhalt + 2)
5 für Gleitkommafelder

Der genaue Platzbedarf berechnet sich aus:

5 + ad*2 + ae*f

ad = Anzahl der Dimensionen
ae = Anzahl Elemente (insgesamt)
f = Faktor (s.o.)

Löschen von Feldern

Einmal dimensionierte Felder können nicht mehr gelöscht werden, d.h. der eben beschriebene Platzbedarf bleibt für die ganze Laufzeit des Programms erhalten.

Allerdings kann bei Stringfeldern wenigstens der Inhalt gelöscht werden, was sehr wichtig sein kann, da der Inhalt von Stringfeldern im Allgemeinen wesentlich mehr Platz benötigt, als die Verwaltung des Inhalts mit 5 Bytes pro Element, da ja ein Element maximal 255 Bytes Inhalt speichern kann.

Da man Felder nicht mehr löschen kann, sollte man sich gut überlegen, ob man ein schon dimensioniertes Feld nicht später im Programm nochmal verwenden kann, ehe man ein neues dimensioniert und das alte ungenutzt liegen läßt.

Fehler

Wenn Feldelemente angesprochen werden, die noch nicht dimensioniert wurden, so wird BAD DIMENSION ERROR gemeldet.

Versucht man, ein schon dimensioniertes Feld nochmal zu dimensionieren, so wird REDIM'D ARRAY ERROR gemeldet. Dies kann besonders leicht passieren, wenn aus Versehen schon ein Feldelement angesprochen wurde, ehe das Feld durch DIM dimensioniert wurde. Wie erwähnt, werden noch nicht dimensionierte Felder automatisch dimensioniert, sobald ein Feldelement angesprochen wird.

9. Stringfunktionen

9.1 Einführung

Was ein String ist und wie er auftreten kann, ist bei Datenarten und Darstellungsweisen erklärt. Dort ist auch beschrieben, daß Zahlen und Strings grundverschiedene Datenarten sind, die ohne weiteres nicht zusammengewürfelt werden können.

Dennoch gibt es Möglichkeiten, diese beiden Datenarten ineinander überzuführen, bzw. umzuwandeln, soweit dies sinnvoll ist. Zwei Gruppen von Umwandlungen sind zu unterscheiden:

Einerseits werden ganze (Gleitkomma-)Zahlen umgewandelt, also Strings mit mehreren Zeichen in Zahlen und umgekehrt. Dies sind die Funktionen VAL bzw. STR$.

Andererseits kann ein einziges Byte in eine (Byte-)Zahl und umgekehrt umgewandelt werden. Diese Funktionen werden von ASC bzw. CHR$ ausgeführt.

Nichts mit Umwandlungen zu tun haben die Funktionen LEFT$, MID$ und RIGHT$, sondern sie dienen dazu, Teilstrings aus Strings auszusondern.

Schließlich gibt es noch die Funktion LEN, die die Länge eines Strings übergibt.

Für Fortgeschrittene gibt es am Ende noch Hinweise zu einer ganz ausgeklügelten Nutzung von Stringvariablen.

9.2 LEN (Länge des Strings)

Zweck

LEN übergibt die Länge des angegebenen Strings.

Format

LEN (stringausdruck)

stringausdruck beliebiger Stringausdruck

Die Funktion LEN ist ihrerseits ein Byte-Ausdruck.

Anmerkungen

Obwohl als Argument von LEN ein beliebiger Stringausdruck zugelassen ist, der auch Stringkonstante enthalten kann, wird meistens das Argument nur aus Stringvariablen bestehen, da die Länge von Stringkonstanten ja bekannt ist.

Der Wertebereich von LEN geht von 0 bis 255. Der Wert 0 kennzeichnet den leeren String, also einen Stringausdruck, der kein Zeichen enthält.

Beispiele

```
A = LEN (A$)
IF LEN (A$+B$) g 15 THEN
C = LEN (A$(3))+7
```

9.3 Gleitkommaumwandlungen

9.3.1 Einführung

Bei INPUT wurde besprochen, daß die Fehlermeldung REDO FROM START umgangen werden kann, wenn man auch Zahleneingaben in Stringvariable liest. Zur Weiterverarbeitung muß dann aber eine Möglichkeit gegeben sein, die Zahl im String an eine Zahlenvariable zuzuweisen. Diese Möglichkeit bietet VAL.

Bei PRINT wurde erwähnt, daß das Zahlenausgabeformat nicht direkt beeinflußt werden kann. Indem man aber eine Zahl in einen String verwandelt, öffnen sich beliebige Bearbeitungsmöglichkeiten, indem man den String in Teilstrings zerlegt (z.B. Vor- und Nachkommateil), diese beliebig weiterbearbeitet und zum Schluß wieder zusammensetzt. Die Umwandlung von Zahlen in Strings wird von STR$ durchgeführt.

9.3.2 VAL (Zahlenwertwert eines Strings)

Zweck

VAL wandelt den Stringausdruck im Argument in einen Gleitkomma-Zahlenwert um.

Format

VAL (stringausdruck)

stringausdruck = beliebiger Stringausdruck

VAL stellt einen Gleitkomma-Ausdruck dar.

Anmerkungen

Bei der Umwandlung wird der String von links nach rechts nach einem nichtnumerischen Zeichen durchsucht. Bis vor dieses Zeichen wird der String in einen Zahlenwert umgewandelt.

Ist bereits das erste Zeichen ein nichtnumerisches, wird der Wert 0 übergeben. 0 wird auch übergeben, wenn der String leer ist, oder wenn er keine der Ziffern von 1 bis 9 erhält. Folgende Strings ergeben den Wert 0 (hier wird nur der Stringinhalt in Anführungszeichen dargestellt):

""	leerer String
" - E -20 "	Mantisse fehlt
" "	Blanks
" 0000000.0 "	Nullen
"Z123456"	Z = nichtnumerisch
" 123456"	SHIFT-Blank am Anfang

Der letzte Fall kann einigen Ärger verursachen, da der Anwender möglicherweise aus Versehen SHIFT-Blank gedrückt hat, den Unterschied zum normalen Blank auf dem Bildschirm nicht sehen kann und trotzdem von Ihrem Programm die Meldung bekommt, er dürfe nicht 0 eingeben. Wollen Sie dies verhindern, müssen Sie etwaige SHIFT-Blanks vor der Umwandlung mit VAL feststellen und abschneiden.

Der Wert 0 läßt also keinen eindeutigen Schluß auf den Inhalt des Strings zu.

Für die Beschaffenheit des Zahlenstrings gelten dieselben Regeln wie für INPUT (s. INPUT / Zahlenformat).

Der weitere Inhalt des Strings hinter dem ersten nichtnumerischen Zeichen ist für VAL belanglos. Sie haben aber keine Möglichkeit, festzustellen, bis wohin VAL den String ausgewertet hat.

9.3.3 STR$ (Stringdarstellung einer Zahl)

Zweck

STR$ wandelt einen Gleitkommaausdruck in einen String um.

Format

STR$ (gleitkommaausdruck)

STR$ übergibt einen String von maximal 15 Zeichen Länge.

Anmerkungen

Mit der gleich erklärten Ausnahme ist die Darstellung der Zahl als String identisch mit dem Format, das bei PRINT erklärt ist.

Im Gegensatz zum PRINT-Ausgabeformat folgt aber nach der Zahl kein weiteres Zeichen (Blank oder 'Cursor nach rechts')

Beispiel

```
A$ = STR$(A)
B$ = STR$(C+D)+"/"+STR$(E-F)
```

9.4 Byteumwandlungen

9.4.1 Einführung

Wie bei (s. Datenarten) beschrieben, ist ein Byte die kleinste Dateneinheit für BASIC und kann 256 verschiedene Zustände darstellen, in BASIC ausgedrückt durch die Dezimalwerte von 0 bis 255.

Eine Stringvariable kann 255 Bytes aufnehmen. Welche Bedeutung diese einzelnen Bytes für irgendein Peripheriegerät, eine Funktion oder einen Algorithmus haben, ist völlig gleichgültig.

Welche Anwendungen sich für solch ein 'Bytefeld' ergeben können, ist im Anschluß an die Stringbefehle erläutert.

Voraussetzung für die meisten Anwendungen, die nicht nur Text in Strings speichern, sind die zwei Funktionen, die ein Byte in die entsprechende Dezimal-Zahl umwandeln können und umgekehrt.

Die Funktion ASC wandelt von String in Zahlendarstellung um. Ihr Name ist vom ASCII-Code abgeleitet (s. Codes).

Die Funktion CHR$ wandelt eine Byte-Zahl in ein Byte um. CHR ist die Abkürzung von CHaracteR, also von Zeichen. Sie dürfen diesen Namen nur nicht zu eng sehen, weil nicht jedes Byte sichtbar dargestellt werden kann, wenn es als ASC-Code interpretiert wird (s. ASC, BSC).

9.4.2 ASC( ) (Code des ersten Stringzeichens)

Zweck

ASC( ) wandelt das erste Byte eines Strings in sein Dezimaläquivalent um.

Format

ASC (string)

string beliebiger, nichtleerer String
ASC( ) stellt einen Byte-Ausdruck dar

Anmerkungen

Da es zum leeren String keine Entsprechung im ASC-Code gibt, meldet der Rechner ILLEGAL QUANTITY ERROR, falls der ASC-Wert des leeren Strings gebildet werden soll.

Beispiel:

```
?ASC("")          ergibt:   ILLEGAL QUANTITY ERROR
```

Der ASC-Wert 0, der bei INPUT und GET Probleme bereitet, kann durch ASC gebildet werden und ist nicht mit dem leeren String zu verwechseln.

Beispiel:

```
A$ = CHR$(0)
?ASC(A$)
0
```

Übungsbeispiel (setzt MID$ voraus)

Für alle Zeichen eines Strings soll der ASC-Code dargestellt werden:

```
10 INPUT"STRING";A$              String eingeben
20 FOR I = 1 TO LEN (A$)         Schleife über alle Zeichen des Strings
30 M$=MID$(A$,I,1) :             I-tes Zeichen isolieren
   A=ASC(M$):                    in ASC-Wert umwandeln
   PRINT I,M$,A                  laufende Nummer, Zeichen und ASC-Code
                                 drucken
40 NEXT                          nächstes Zeichen
```

Zeile 30 könnte mit gleicher Wirkung auch so aussehen:

```
30 PRINT I , MID$(A$,I,1) , ASC(MID$(A$,I))
```

Beachten Sie, daß beim zweiten MID$ der Längenparameter entfallen kann, weil ASC( ) sowieso nur das erste Byte des Strings umwandelt!

9.4.3 CHR$ (ASC-Code in String-Zeichen umwandeln)

Zweck

CHR$( ) wandelt eine Byte-Zahl in ein Zeichen-Byte eines Strings um.

Format

CHR$ (byte-ausdruck)

byte-ausdruck siehe Byte-Zahl
CHR$( ) stellt einen String-Ausdruck dar

Anmerkungen

Die Anwendung von CHR$( ) ist problemlos. Lediglich bei PRINT ist Vorsicht geboten, da durch CHR$( ) auch Steuercodes erzeugt werden können. Z.B. löscht PRINT CHR$(147) den Bildschirm (s. Codes).

Beispiele

CL$ = CHR$(13)+CHR$(10)	schreibt die Codes für Carriage-Return (CR) und Line-Feed (LF) in CL$
AN$ = CHR$(34)	einzige Möglichkeit, ein Anführungszeichen in eine Stringvariable zu schreiben.

9.5 Teilstrings bilden (LEFT$, MID$, RIGHT$)

9.5.1 Einführung und Zweck

Die drei Funktionen, die im folgenden besprochen werden, bringen aus einem String einen linken Teil, einen Teil aus der Mitte oder einen rechten Teil, so sagen es jedenfalls die Bezeichnungen LEFT, MID und RIGHT aus. Eine Skizze soll für alle drei Funktionen veranschaulichen, was sie tun. Wir legen jedesmal denselben Ausgangsstring zugrunde und nummerieren seine Zeichen durch. Der Tabelle ist die Zuordnung Z$ = funktion (Q$) zugrunde gelegt.

1234567890	1234567890	1234567890	Inhalt von Q$
LEFT$(Q$,7)	MID$(Q$,3,7)	RIGHT$(Q$,7)	Funktion
1234567	3456789	4567890	Inhalt von Z$

Alle drei Teilstringfunktionen haben gemeinsam, daß sie als Parameter einen String (Quellstring) und die Anzahl der Zeichen benötigen, die übernommen werden sollen.

Bei LEFT$ wird die gewünschte Zeichenzahl ab dem linken Rand (Stringanfang) übernommen, bei RIGHT$ ab dem rechten Rand. Bei MID$ ist ein weiterer Parameter erforderlich, der angibt, ab wo die gewünschte Anzahl Zeichen genommen werden soll.

Da die drei Funktionen in der Wirkung und im Format sehr ähnlich sind, sollen sie in einem Abschnitt zusammen vorgestellt werden:

9.5.2 Format

LEFT$ (qs , az)
RIGHT$ (qs , az)
MID$ (qs , sp , az)

qs Quellstring (beliebiger Stringausdruck)
az Anzahl Zeichen, die übergeben werden sollen
sp Spalte, ab der übergeben wird

az und sp sind Byte-Ausdrücke!

Gemeinsame Parameter von LEFT$, MID$, RIGHT$

az darf alle Werte zwischen 0 und 255 annehmen. Sind weniger Zeichen als az im Quellstring, so werden soviele gebracht, wie vorhanden sind.

az = 0 hat zur Folge, daß der leere String übergeben wird.

Spalten-Parameter von MID$

sp darf alle Werte von 1 bis 255 annehmen.

sp = 0 erzeugt ILLEGAL QUANTITIY ERROR!

Wenn sp größer ist als die Länge von qs, wird der leere String übergeben.

Wenn az fehlt, wird ab sp bis zum Ende von qs übergeben.

Quellstring

Der Quellstring ist ein normaler Stringausdruck, dessen Ergebnis erst gebildet werden muß, ehe die Teilstringfunktion wirken kann. Falls Sie also z.B. MID$ (A$+B$,3,2) bilden wollen, darf A$ und B$ zusammen nicht länger sein als 255 Zeichen, da sonst STRING TOO LONG ERROR gemeldet wird. Es hilft also nichts, daß das Endergebnis nur 2 Zeichen lang gewesen wäre, da die Kettung A$+B$ nichts mit MID$ zu tun hat!

Beispiel

```
PRINT...;RIGHT$("          "+STR$(A),10);...
```

... steht stellvertretend für beliebige Ausgaben

Dieses Beispiel druckt den Wert von A rechtsbündig in ein 10 Zeichen breites Feld. Die Anzahl der Blanks muß mindestens Feldbreite-2 betragen.

Beachten Sie, daß bei Exponentialdarstellung die Feldbreite von 10 u.U. nicht ausreicht.

9.6 Strings als eindimensionale Byte-Felder

Eine Stringvariable ist im Grunde ein eindimensionales Bytefeld mit variabler Länge zwischen 0 und 255 Elementen.

Für den fortgeschrittenen Programmierer eröffnen sich eine Reihe von Möglichkeiten, deren Beherrschung u.U. darüber entscheiden kann, ob ein Problem auf diesem Computer gelöst werden kann oder nicht.

Gleitkomma, Integer und String sind die drei Datentypen, die im Commodore-BASIC durch entsprechende Variable und Variablenfelder unterstützt werden. Nun kommt es aber oft vor, daß man Daten hat, die durch 1 Byte oder 3 Bytes oder vielleicht 7 Bytes dargestellt werden können. Mit etwas Geschick kann man mit Hilfe von einfachen oder indizierten Stringvariablen maßgeschneiderte Felder für solche Daten aufbauen.

Damit wäre der Vorteil von Stringvariablen angesprochen, daß sie Informationen aufnehmen können, die auf eine beliebige Anzahl Bytes komprimiert sind.

Der zweite Vorteil, der mindestens genauso große Auswirkung haben kann, ist die variable Anzahl der Elemente pro Variable.

An einer einfachen Rechnung sei gezeigt, daß letztere Eigenschaft über Sein oder Nicht-Sein (eines Programmes) entscheiden kann:

Sie haben 400 Elemente (Kunden) und für jedes Element können maximal 200 Ereignisse (Einkäufe) auftreten, von denen jedes in 1 Byte dargestellt werden kann (Zeiger auf Datensatz auf der Diskette).

Wenn Sie aufgrund von Zeitanforderungen eine Verweisliste mit diesen Informationen ständig im Arbeits-Speicher halten müssen, benötigen Sie folgenden Speicherplatz, wenn eine mittlere Ereigniszahl von 50 vorausgesetzt wird:

400 Elemente * 50 Ereignisse/Element * 1 Byte/Ereignis

Für die Information benötigen Sie also 20 K.

Wenn Sie nun ein Stringfeld mit DIM A$(400) vereinbaren und im Mittel mit 50 Bytes füllen, benötigen Sie an zusätzlicher Verwaltung ca. 400 * 5 = 2000 Bytes. Falls Ihr Programm nicht größer als ca. 8 K ist, könnte diese Datenmenge gerade noch verwaltet werden.

Würden Sie aber ein zweidimensionales Integerfeld mit 400*200 Elementen (200 wegen der Maximalanforderung von 200 Ereignisse/Element) vereinbaren wollen, bräuchten Sie nur dafür einen Arbeitsspeicher von 400*200*2 = 160 KByte!

Wir haben also mit Hilfe eines zweidimensionalen Bytefeldes, dessen Länge nur in einer Richtung festgelegt, in der anderen aber variabel ist, den Speicherplatzbedarf auf 1/8 senken können.

Nun aber noch einige Hinweise zur internen Verwaltung eines Bytefeldes.

Im einfachsten Fall einer Datenlänge von einem Byte kann man einfach über die Funktion MID$ das gewünschte Byte herausgreifen.

Ist die Datenlänge größer, aber konstant, kann über einen einfachen Algorithmus zugegriffen werden.

Hat man aber Daten unterschiedlicher Länge in einem String, so ist eine von zwei klassischen Methoden zu wählen:

(1) Trennzeichen

Einer der Byte-Werte von 0 bis 255 wird als Trennwert (Delimiter) definiert, darf also in den Daten nicht auftreten. Das n-te Datum wird dann gefunden, indem das n te n+1 te Trennzeichen gesucht werden und mit MID$ der entsprechende Teilstring entnommen wird.

Der Nachteil dieser Lösung ist, daß ein Byte-Wert für das Trennzeichen geopfert werden muß, was oft nicht möglich ist. Außerdem dauert die Suche nach den Trennzeichen (in BASIC) u.U. zu lange. Der Vorteil ist einfaches Einfügen oder Löschen von Elementen.

(2) Zeigerverwaltung

Die ersten Bytes jedes Strings sind Zeigerbytes mit folgenden Aufgaben:

Das erste Byte gibt die Anzahl der Datenelemente in diesem String an und zeigt gleichzeitig auf den Beginn des ersten Datenelementes.

Das zweite Byte zeigt auf den Anfang des zweiten Datenelementes, kennzeichnet also gleichzeitig das Ende des ersten Datenelementes.

usw.

Der Platzbedarf für die Verwaltung ist pro String um 1 Byte höher als bei Lösung (1). Der Zugriff erfolgt schneller, da keine Suchschleife laufen muß, sondern nur ein drei- oder viermaliger Zugriff mit MID$. Der Nachteil gegenüber (1) liegt in der aufwendigen Zeigerverwaltung. Wenn ein neues Element hinzukommt oder wegfällt oder sich die Länge eines Elements ändert, müssen alle beteiligten Zeiger geändert werden, wodurch das Eintragen von neuen Informationen wesentlich länger dauern kann, als im Fall (1).

10. Mathematische Funktionen

10.1 Einführung

Die folgenden Funktionen bieten die Möglichkeiten, Zahlen umzuwandeln, zu zerlegen und grundlegende mathematische Funktionen zu bilden.

Hier noch einige Bemerkungen zur Rechengenauigkeit:

Wenn Sie auf eine Zahl eine mathematische Funktion anwenden und auf das Ergebnis wieder die Umkehrfunktion, dann müßte nach mathematischen Regeln das Ergebnis wieder die ursprüngliche Zahl sein. Voraussetzung ist natürlich, daß die Umkehrfunktion überhaupt existiert und eindeutig ist. Wiederholt man dieses Rechenexempel immer mit dem Ergebnis, dann müßte sich auch immer wieder das ursprüngliche Ergebnis wiederholen.

Beispiel:

Man bildet die Wurzel aus einer Zahl I und quadriert das Ergebnis wieder. Damit hat man wieder die Zahl I. Nun wiederholt man diese Rechnung mehrmals und kommt immer wieder auf dasselbe Ergebnis.

Anders bei der Anwendung der mathematischen Funktionen in BASIC. Führt man hier diese Berechnungen wie oben gezeigt durch, dann kann es im ungünstigen Fall schon bei der zweiten Wiederholung passieren, daß statt der Zahl 3 die Zahl 3.00000001 herauskommt.

Dieses Problem wurde schon bei Rundungsfehler und Intergerzahlen behandelt. Hier soll nur noch einmal darauf hingewiesen werden, daß man bei Berechnungen immer dafür sorgen muß, daß sich Rundungsfehler nicht fortpflanzen.

Argumente

Die Klammerausdrücke, die den mathematischen Funktionen folgen, können beliebige Gleitkomma- (oder Integer-) ausdrücke sein.

10.2 Zahlen zerlegen

Diese Funktionen sind einfache mathematische Funktionen, die auch der Programmierer, der keine eigentlichen mathematischen Probleme zu bearbeiten hat, benötigen wird. Sie werden sogar schon bei einfachsten kaufmännischen Aufgaben gebraucht.

Sie ermöglichen, den Betrag einer Zahl, den ganzzahligen Teil einer Zahl und das Vorzeichen einer Zahl zu bilden.

10.2.1 ABS (Absolut-Betrag)

Zweck

ABS gibt den Betrag oder **ABS**olutwert einer Zahl aus. Aus einem negativen Vorzeichen wird also ein positives gemacht.

Format

ABS (gleitkommaausdruck)

Beispiel

```
?ABS(-1.2);ABS(1.2)
1.2   1.2
```

10.2.2 INT (Ganzahliger Wert)

Zweck

INT bildet die nächstkleinere ganze Zahl der Zahl im Argument.

Format

INT (gleitkommaausdruck)

(Keine Abkürzung möglich)

Anmerkungen

Diese Funktion darf nicht mit der Zuweisung einer Gleitkommazahl an eine Integervariable verwechselt werden. Die Zuweisung macht nämlich eine Umwandlung vom Gleitkommaformat ins Integerformat der Zahlendarstellung.

Die Funktion INT gibt als Ergebnis wieder einer Gleitkommazahl aus. Die Zahl muß also nicht im Integer-Bereich liegen.

Das Ergebnis entspricht aber der Umwandlung Gleitkomma nach Integer:

Wenn der (positive) Gleitkommawert Nachkommastellen enthält, werden diese abgeschnitten, es wird also nicht gerundet!

Beispiel:

```
?INT(123.258)
123
```

Exakter ausgedrückt, wird die **nächstkleinere ganze Zahl** gebildet. Dies sieht bei negativen gebrochenen Zahlen so aus:

```
?INT(-123.587)
-124
```

Anwendung

Die hauptsächliche Anwendung der Funktion INT ist das Runden von Gleitkommazahlen an einer bestimmten Nachkommastelle (s. Rundungsfehler).

Beispiel

Die Variable A soll auf drei Nachkommastellen genau gerundet werden.

```
10 B=INT(A*1000+.5)/1000
```

Damit fallen Rechenungenauigkeiten hinter der dritten Dezimalstelle heraus.

10.2.3 SGN (Vorzeichen)

Zweck

SGN bildet das Vorzeichen einer Zahl wieder als Zahl ab. Dabei entsprechen sich:

Vorzeichen und Zahlen

+	+1
0	0
-	-1

Format

SGN (gleitkommaausdruck)

Beispiele

```
? SGN(500.84)
1

? SGN(-800)
-1
```

10.3 Logarithmische Funktionen

Die Funktionen für Wurzelberechnung, Exponentialfunktion und Logaritmus sind unter der Gruppe 'Logarithmische Funktionen' zusammengefaßt, da sie logarithmisch berechnet werden.

10.3.1 SQR (Quadratwurzel)

Zweck

SQR gibt die Quadratwurzel einer positiven Zahl aus.

Format

SQR (gleitkommaausdruck)

Der Gleitkommaausdruck muß einen Wert größer oder gleich Null haben, sonst ergibt sich die Fehlermeldung ILLEGAL QUANTITY ERROR.

Beispiel

```
?SQR(2)
1.41421356

?SQR(-2)
ILLEGAL QUANTITY ERROR
```

10.3.2 EXP (Exponentialfunktion)

Zweck

EXP gibt das Ergebnis der Berechnung e hoch gleitkommaausdruck aus. Die Zahl e (Eulersche Zahl) hat dabei den Wert 2.7182183. Diese Berechnung wird auch natürliche Exponentialfunktion genannt.

Format

EXP (gleitkommaausdruck)

Anmerkungen

Der Maximalwert des Gleitkommaausdrucks ist 88.02969189. Gibt man einen größeren Wert für die Berechnung der Exponentialfunktion an, ergibt sich OVERFLOW ERROR.

Den Wert der Zahl e bekommt man z.B. durch E=EXP(1).

Bitte verwechseln Sie das mathematische Kürzel 'e' nicht mit dem BASIC-Kürzel E bei der Exponentialdarstellung!

10.3.3 LOG (Natürlicher Logarithmus)

Zweck

LOG berechnet den Logarithmus einer Zahl zur Basis e (s. EXP). Dieser Logarithmus heißt auch 'Natürlicher Logarithmus'.

Format

LOG (gleitkommaausdruck)

Der Gleitkommaausdruck muß einen Wert größer als Null haben, sonst ergibt sich ILLEGAL QUANTITY ERROR.

Anmerkungen

Die Funktion LOG kann auch zur Berechnung von Logarithmen mit beliebiger Basis verwendet werden. Will man den Logaritmus zur Basis 10 berechnen, muß die Funktion so aussehen:

```
C = LOG(X)/LOG(10)
```

Bei beliebiger Basis B wird der Ausdruck verwendet:

```
C = LOG(X)/LOG(B)
```

10.4 Trigonometrische Funktionen

Für die Berechnung der trigonometrischen Funktionen gibt es im CBM-BASIC einen Satz von Grundfunktionen. Mit diesen lassen sich direkt die Funktionen Tangens, Sinus, Cosinus und Arcus Tangens berechnen.

Andere trig. Funktionen lassen sich mit Kombinationen dieser Grundfunktionen leicht berechnen (s. Umrechnungen).

Zur Genauigkeit

Bei Werten des Arguments, die größer als 6*PI sind, ergeben sich Ungenauigkeiten für die Berechnung von trig. Funktionen. Wir empfehlen daher, vor der Berechnung einer trig. Funktion das Argument auf den Bereich zwischen -2*PI und +2*PI zu beschränken.

Argument

Das Argument der trig. Funktionen muß im Bogenmaß angegeben werden. Für die Vereinfachung der Eingaben und Erhöhung der Genauigkeit existiert die feste Zuweisung der Zahl PI. Liegt das Argument im Gradmaß vor, muß es vor der Berechnung ins Bogenmaß umgerechnet werden. Diese Umrechnung kann mit folgender Berechnung erfolgen:

```
ARCUS = WINKEL * PI / 180
```

wobei ARCUS der Winkel im Bogenmaß
und WINKEL der Winkel im Gradmaß ist.

10.4.1 TAN, SIN, COS (Tangens, Sinus, Cosinus)

Zweck

Mit diesen Funktionen lassen sich die trigonometrischen Funktionen Tangens, Sinus und Cosinus berechnen.

Format

TAN (gleitkommaausdruck)
SIN (gleitkommaausdruck)
COS (gleitkommaausdruck)

10.4.2 ATN (Arcus-Tangens)

Zweck

ATN gibt den Arcus-Tangens einer Zahl aus.

Format

ATN (gleitkommaausdruck)

Anmerkungen

Das Ergebnis der ATN-Funktion ist ein Wert im Bogenmaß und liegt zwischen -PI/2 und +PI/2. Das Ergebnis ist also der Hauptwert des Arcus-Tangens.

Für die Umrechnung des Bogenmaß in Gradmaß kann folgende Formel helfen:

WINKEL = 180/PI * ARCUS

wobei WINKEL der Winkel im **Grad**maß
und ARCUS der Winkel im **Bogen**maß ist.

10.4.3 Die Zahl PI

Die Zahl PI ist im CBM-BASIC als Funktion implementiert und kann für Berechnungen verwendet werden. Sie hat den Wert 3.14159265.

10.4.4 Umrechnungen

Mit den folgenden Formeln können aus den bestehenden Funktionen andere berechnet werden:

COTAN (X)	= COS(X)/SIN(X)	für X ungleich 0
ARC SIN (X)	= ATN(X/SQR(1-X*X))	für positive X
ARC COS (X)	= ATN(SQR(1-X*X)/X)	für positive X
ARC COT (X)	= ATN(1/X)	für positive X

11. BIT-Funktionen

11.1 Einführung

Die BIT-Funktionen führen bitweise logische Verknüpfungen mit zwei 16-Bit-Zahlen durch. Dabei kommt es nicht darauf an, ob die Zahlen, die miteinander verknüpft werden, intern als 2-Byte-Zahl im Integerformat oder als 5-Byte-Zahl im Gleitkommaformat dargestellt wird.

Man kann beliebige Zahlenvariablen miteinander verknüpfen, sofern sie einzeln den Wert einer 16-Bit-Zahl nicht überschreiten, also nicht kleiner als -32767 und nicht größer als +32767 sind. Liegt ein Wert nicht in diesem Bereich, meldet der Rechner ILLEGAL QUANTITY ERROR.

Die Zahlen werden bei der Verknüpfung intern als 2-Byte-Zahl dargestellt und bitweise mit der angegebenen logischen Verknüpfung behandelt. Nachkommastellen werden also nicht berücksichtigt. Beachten Sie aber die Umwandlung von negativen gebrochenen Zahlen in ganze Zahlen!

11.2 Verknüpfungstabellen

Verknüpfung	Ergebnisbit
0 AND 0	0
0 AND 1	0
1 AND 0	0
1 AND 1	1
0 OR 0	0
0 OR 1	1
1 OR 0	1
1 OR 1	1
NOT 1	0
NOT 0	1

Beispiele von Zahlenverknüpfungen

Verknüpfung	Dezimalzahl	Binärdarstellung
	15	0000 0000 0000 1111
AND	8	0000 0000 0000 0100
Ergebnis	8	0000 0000 0000 0100
	32767	0111 1111 1111 1111
AND	-32767	1000 0000 0000 0001
Ergebnis	1	0000 0000 0000 0001
	32767	0111 1111 1111 1111
OR	-32767	1000 0000 0000 0001
Ergebnis	-1	1111 1111 1111 1111
	15	0000 0000 0000 1111
OR	256	0000 0000 0001 0000
Ergebnis	271	0000 0000 0001 1111
NOT	1	0000 0000 0000 0001
Ergebnis	-2	1111 1111 1111 1110
NOT	32767	0111 1111 1111 1111
Ergebnis	-32768	1000 0000 0000 0000

Die Zahl -32768 kann also als Ergebnis erscheinen, als Argument einer logischen Verknüpfung kann sie aber nicht verwendet werden (ergibt Fehlermeldung).

12. Sonstige Funktionen

12.1 Zufallszahl erzeugen

12.1.1 RND (Random = Zufall)

Zweck

RND erzeugt Zufallszahlen.

Format

RND (gleitkommaausdruck)

Ergebnis

Der Zufallszahlengenerator verwendet einen Algorithmus, der eine Zahl zwischen 0 und 1 liefert.

12.1.2 Argumente

Es gibt drei Möglichkeiten, eine Zufallszahl zu erzeugen. Welche man verwendet, hängt von der Problemstellung ab. Sie wird bestimmt durch das angegebene Argument, das positiv, Null oder negativ sein kann.

(1) **positives Argument**

Diese Funktion übergibt immer den nächsten Wert einer Zufallszahlenreihe, die durch einen numerischen Algorithmus in BASIC berechnet wird. Diese Reihe wird beim Einschalten initialisiert, indem ihr ein bestimmter Anfangswert zugewiesen wird. Die Reihe läuft unabhängig vom Wert des Arguments, man kann also der Einfachheit halber hier z.B. immer 1 angeben.

(2) **Negatives Argument**

Bei Angabe eines negativen Arguments wird die Zufallsreihe neu initialisiert, indem ihr ein neuer Anfangswert zugewiesen wird. Dieser Anfangswert ist abhängig vom Wert des Arguments.

Bei gleichem negativem Argument folgt immer dieselbe (Pseudo-) Zufallsreihe.

(3) **Argument = Null**

Bei Angabe eines Arguments Null wird eine ganz andere Möglichkeit der Zufallszahlenerzeugung verwendet. Das Programm liest verschiedene bezugslose Intervall-Zeitgeber und bildet durch einen Algorithmus eine Zahl (Zufallszahl).

Diese Möglichkeit gibt dann eine wirkliche Zufallszahl aus, wenn die Zufallszahlenaufrufe sehr viel langsamer als die Takte der Intervall-Zeitgeber sind. Das ist der Fall, wenn die Aufrufe z.B. nach Eingaben vom Benutzer erfolgen. In einem Programmablauf ergibt diese Möglichkeit dagegen keine wirkliche Zufallszahl, da das Programm relativ zu den Zeitgebern zu schnell läuft. In einem Programm ist also die einzige wirkliche Zufallszahl nur die erste aufgerufene.

12.1.3 Verbesserung der statistischen Verteilung der Zufallszahlen

Da die Zufallsreihe bei einem bestimmten Anfangswert immer gleich läuft, sollte in einem Programm vor jeder Zufallszahlenerzeugung die Reihe neu initialisiert werden. Der negative Wert für die Initialisierung kann sinnvollerweise aus der Zeitvariablen genommen werden.

```
D = RND(-TI)
D = RND(1)
```

12.1.4 Bereich der Zufallszahl einschränken

Die Zufallszahlenfunktion gibt nur Zahlen zwischen 0 und 1 aus. Die meisten Probleme benötigen aber Zufallszahlen, die in bestimmten Bereichen liegen müssen. Ganzzahlige Zufallszahlen innerhalb eines bestimmten Bereichs können mit folgender Umrechnung errecht werden:

Der Anfangswert des Bereichs sein AN, der Endwert sei EN.

Dann liegt die Zufallszahl, die nach der Formel

ZU = INT((EN-AN)*RND(1)+AN)

errechnet wird, innerhalb des gewünschten Bereiches.

12.2 Vom Programmierer definierbare Funktionen

12.2.1 Einführung

Mit Hilfe von DEF FN .. ( )= .. können eigene BASIC-Funktionen definiert werden, die mit ähnlicher Syntax wie die implementierten Funktionen aufgerufen werden können.

Durch FN .. ( ) können diese Funktionen dann beliebig oft mit einem jeweils anderen Parameter aufgerufen werden.

12.2.2 DEF FN ( ) (Funktion definieren)

Jede Funktion, die irgendwann im Programm aufgerufen wird, muß vorher definiert worden sein. Das Programm muß also einmal über die Funktionsdefinition gelaufen sein. Das bedeutet aber nicht, daß die Definition unbedingt hinter einer kleineren Zeilennummer stehen muß, als der Aufruf.

Format

DEF FN funktionsname (gleitkommavariable) = gleitkommaausdruck

(Hinter der Definition können weitere Anweisungen stehen, die mit der Definition nichts zu tun haben.)

Parameter

funktionsname

Jede Funktion muß einen eigenen Namen enthalten. Der Name wird nach den Regeln für Variablennamen gebildet. Von daher sind also über 900 verschiedenen Funktionen in einem Programm möglich.

Dieser Funktionsname hat aber nichts mit eventuell vorhandenen gleichnamigen Variablen zu tun. Für die Verwaltung stellen FN-Funktionsnamen einfach eine vierte Variablenart dar.

gleitkommavariable (in Klammern hinter funktionsname)

Jede Funktion muß ein Argument haben, das hinter dem Namen in Klammern angegeben wird. Dieses Argument wird bei der Definition durch eine beliebige Gleitkommavariable bezeichnet.

Diese Variable hat nur **Platzhalterfunktion.** Das bedeutet, daß diese Variable sowohl bei der Definition als auch beim Aufruf existieren kann, ohne daß ihr Wert verändert wird.

In der Regel wird diese Platzhaltervariable im Funktionsausdruck auf der rechten Seite vorkommen. Sie muß aber nicht vorkommen, wohl aber muß sie hinter dem Funktionsnamen in Klammern angegeben werden, um die SYNTAX-Prüfung zufriedenzustellen.

gleitkommaausdruck

Der gleitkommaausdruck bestimmt den Wert der Funktion beim Aufruf. Er kann ein beliebig komplexer Ausdruck sein und ist nur beschränkt durch die Stacktiefe und die Zeilenlänge.

Der Ausdruck kann die Platzhaltervariable und tatsächliche Variable enthalten. Der Wert der tatsächlichen Variablen ist beim Aufruf der Funktion der Wert, den diese Variable zu diesem Zeitpunkt gerade hat.

Durch Verschachtelung von Funktionen ist es möglich, Funktionen zu definieren, die mehrere BASIC-Zeilen lang sind.

Beispiele

```
10 DEF FN A(X)=1/COS(X)
20 DEF FN PR(V)=V/AV*100
30 DEF FN B(X)=FNA(X)+2
```

12.2.3 FN .. (Funktion aufrufen)

Format

FN funktionsname (gleitkommaausdruck)

Parameter

funktionsname (s. DEF FN)

gleitkommaausdruck

Der Wert dieses Ausdrucks wird bei der Berechnung der Funktion für die Platzhaltervariable eingesetzt, die bei der Definition ebenfalls in Klammern gestanden hat.

Die Funktion kann noch weitere Argumente haben, indem bei der Funktionsberechnung globale Variable verwendet werden, die vor dem Funktionsaufruf zugewiesen werden (s. DEF FN).

Anmerkungen

Ist in der Funktionsdefinition ein Fehler, der eine Systemfehlermeldung bewirkt, so wird die Fehlermeldung erst beim Aufruf der Funktion ausgegeben. Hier wird aber die Zeilennummer der aufrufenden Zeile ausgegeben, obwohl der Fehler in einer anderen Zeile steht.

Wenn Sie mit **Overlay** arbeiten, beachten Sie bitte folgende wichtige Eigenschaft: Ähnlich wie bei Strings wird bei DEF FN nur ein Zeiger auf die Programmstelle angelegt, an der die Funktion definiert ist. Wenn nun durch Overlay ein anderes Programm nachgeladen wird, zeigt der Zeiger auf eine sinnlose Stelle und es würde in der Regel SYNTAX ERROR gemeldet. Deshalb bleibt bei Overlay nur der Weg, in jedem neuen Programm die benötigten Funktionen wieder zu definieren.

Gleichzeitig können Sie daraus erkennen, daß die gleiche Funktion ohne weiteres mehrmals definiert werden kann. Gültig ist die jeweils letzte (zeitlich) Definition.

Beispiele

```
100 PRINT FN A(100)
200 PRINT FN PR(G)
```

12.3 Zeitvariablen

Es gibt in BASIC zwei Variablen, deren Wert mit der aktuellen Zeit versorgt wird. Dies sind einfache Möglichkeiten, in BASIC eine 'Echtzeituhr' zu betreiben.

Die Variablen werden mit einem Takt von 1/60 Sekunde aktualisiert.

12.3.1 TI$ (Zeit in Stunden, Minuten, Sekunden)

Die Variable TI$ enthält die aktuelle Tageszeit in sechs Zeichen im Format:

HHMMSS

wobei HH die Stunden
MM die Minuten
SS die Sekunden bedeuten.

Diese Variable kann vom Benutzer einen Wert zugewiesen bekommen. Die Zuweisung muß eine Stringzuweisung von genau sechs Zeichen sein, andernfalls ergibt sich ILLEGAL QUANTITY ERROR.

Die Tageszeit wird in einer 24-Stunden-Uhr angegeben, d.h. die Stunden werden von 00 bis 24 gezählt.

Beim Einschalten des Rechners wird die Zeit mit "000000" initialisiert. Will man die aktuelle Tageszeit während dem Rechnerbetrieb haben, muß diese nach dem Einschalten an TI$ zugewiesen werden.

Beispiel:

Die Zuweisung der Zeit 13 Uhr 59 und 10 Sekunden kann so aussehen:

```
TI$="135910"
```

Eine einfache Möglichkeit, die aktuelle Zeit auf dem Bildschirm zu verfolgen, ist mit folgendem kleinen Programm möglich:

```
10 ?"H"TI$ : GOTO10
```

Hier bedeutet **H** das Steuerzeichen für 'HOME'.

12.3.2 TI (Zeit als Zahlenwert)

Die Variable TI enthält den Wert der Stringvariablen TI$ als 1/60 Sekunden. Die allgemeine Formel ist

TI = ((HH*60+MM)*60+SS)*60, wenn HHMMSS der Inhalt von TI$ ist.

Wird TI$ "000000" zugewiesen, bekommt auch TI den Wert 0. Hat dagegen TI$ den Inhalt "000125", dann hat TI den Wert (1*60+25)*60 = 5100.

Wird versucht, an TI einen Wert zuzuweisen, meldet der Rechner SYNTAX ERROR, der Inhalt der 'Uhr' kann also nur über TI$ verändert werden.

Abfrage eines Zeitintervalls

Will man in einem BASIC-Programm eine Zeitschleife einbauen, die eine bestimmte Zeit wartet, kann das so aussehen:

```
10 TT=TI
20 IF (TI-TT) kleiner als 60 GOTO 20
```

Das Programm springt solange wieder auf die Zeile 20, bis eine Sekunde seit der Zuweisung der Variablen TT vergangen ist.

Exakter Zusammenhang zwischen TI und TI$

Obwohl bei einer Zuweisung 'TI$="....."' der Wert von TI aus TI$ errechnet wird, läuft der Inhalt von TI dann in 1/60 Sekunden-Schritten weiter, also 60 mal genauer als der Inhalt von TI$. Bitte beachten Sie diesen scheinbar unlogischen Zusammenhang:

Bei der Zuweisung einer neuen Zeit wird TI aus TI$ gebildet, bei der Abfrage von TI$ wird dagegen TI$ aus TI errechnet! Intern werden nicht zwei verschiedene Zeiten gespeichert, wie die beiden Variablen TI und TI$ vermuten lassen, sondern es wird nur eine Uhr geführt, die im 1/60 Sekunden-Takt hochgezählt wird. Aus derem Wert wird sowohl die Zahlen-, als auch die Stringdarstellung gebildet.

12.4 ST, DS, DS$ (Status-Variablen)

12.4.1 Einführung

Das CBM-BASIC verwaltet einige Statusvariablen, die den Zustand von Peripheriegeräten anzeigen. Diese Variablen dürfen keinen Wert vom Programm zugewiesen bekommen. Wird dies versucht, ergibt sich SYNTAX ERROR.

12.4.2 ST (Status)

Die Werte der Variablen ST und ihre Bedeutungen sind ausführlich im Kapitel **Status** behandelt.

12.4.3 DS, DS$ (Disk-Status)

Diese Variablen beinhalten den Status des bei der letzten Zugriffsoperation angesprochenen Floppy-Disk-Gerätes.

Der Rechner merkt sich jedesmal, wenn ein Disk-BASIC-Befehl ausgeführt worden ist, mit welchem Gerät dieser Peripherieverkehr durchgeführt wurde. Wird danach der Wert der Variablen DS oder DS$ abgefragt, dann wird vom Rechner die Statusmeldung des richtigen Floppys gelesen und auf DS und DS$ zugewiesen. Bei weiteren Abfragen von DS oder DS$ wird deren Wert solange nicht erneuert, solange der Rechner keine neue Operation mit einem Floppy feststellt.

Bei Verwendung von 'alten' Befehlen wie 'OPEN', 'CLOSE' und ähnlichen kann der Rechner keine Disketten-Operation feststellen und gibt somit den falschen Disk-Status aus. In solchen Fällen sollten Sie den Disk-Status mit INPUT§ über den Fehlerkanal des Floppys lesen.

Inhalt von DS$

DS$ enthält die Floppy-Statusmeldung im Klartext. Das Format und die Bedeutungen der Meldungen sind im Floppy-Handbuch beschrieben.

Inhalt von DS

DS enthält die Nummer der Floppy-Statusmeldung. Dies ist die Zahl, die in DS$ ganz vorne steht. Meldet das Floppy 'alles in Ordnung', so hat DS den Wert 0.

DS kann im BASIC-Programm verwendet werden, um je nach Floppy-Fehlermeldung in verschiedene Behandlungsteile zu verzweigen, während DS$ evtl. direkt auf den Bildschirm ausgegeben werden kann, um die Fehlermeldung im Klartext zu sehen.

13. Maschinenprogramm-Kopplung

13.1 Einführung

Das 'Herz' oder eigentlich mehr der 'Kopf' Ihres Computers ist einer der schnellsten 8-Bit-Microprozessoren auf dem Markt. Dieser Prozessor (6502) verwaltet mit Hilfe von ca. 18 KByte Maschinenprogramm den ganzen Computer. Insbesondere der BASIC-Interpreter ist seinerseits ein ca. 12 KByte großes Maschinenprogramm.

Durch die hohe Geschwindigkeit des Prozessors ist auch der BASIC-Interpreter einer der schnellsten BASIC-Interpreter dieses Umfangs. Das ändert aber nichts daran, daß der Komfort der BASIC-Interpreter-Sprache durch Zeiteinbußen erkauft werden muß.

Warum in Maschinensprache programmieren?

Vor allem bei technisch-/ wissenschaftlichen Anwendungen kann es deshalb unumgänglich sein, wenigstens zum Teil in Maschinensprache arbeiten zu müssen, um sogenannte 'Real-Time'-Überwachungen/Regelungen realisieren zu können. Aber auch im kommerziellen Bereich lohnt der Einsatz von Unterprogrammen in Maschinensprache, da dadurch mit relativ kleinem Aufwand oft ein Zeitgewinn von einigen zehn Prozent erreicht werden kann.

Grundsätzlich läuft ein Maschinenprogramm zwischen zehn und tausend mal schneller als ein BASIC-Programm gleicher Wirkung. In der Regel sind Maschinenprogramme auch wesentlich kürzer als entsprechende BASIC-Programme. Letzteres sollte aber nicht den Ausschlag geben, auf Maschinenebene 'hinabzusteigen', da durch Overlay bei BASIC Platzprobleme wesentlich unkomplizierter gelöst werden, als durch Maschinensprache.

Welche Hilfsmittel sind vorhanden?

BASIC bietet z.B. durch SYS die Möglichkeit, den Interpreter zu verlassen um ein eigenes Maschinenprogramm als Unterprogramm aufzurufen. Nach Abarbeitung dieses Unterprogramms kehrt SYS automatisch wieder in den Interpreter zurück, so daß die nächste Anweisung ausgeführt werden kann.

Auch für die Parameterübergabe stehen Befehle zur Verfügung und durch LOAD können auch Maschinenprogramme geladen werden.

BASIC bietet also alle Hilfsmittel, um Maschinenprogramme aufzurufen. Sofern Sie fertige Maschinenprogramme kaufen, um sie in Ihr BASIC-Programm einzubinden, haben Sie also alles beisammen.

Dagegen sieht es ganz anders aus, wenn Sie selbst Maschinenprogramme entwikkeln wollen. Da Ihr Computer vollkommen auf den eingebauten BASIC-Interpreter abgestimmt ist, bietet er fast keine (eingebauten) Hilfsmittel zur Assemblerprogrammierung. Der residente 'Monitor' stellt zwar auf unterer Ebene (Hexadezimal) eine Verbindung zum Prozessor her. Der Monitor dient aber im wesentlichen dazu, Maschinenprogramme abzuspeichern oder mal schnell ein Byte zu ändern. Notfalls kann man auch eine kleine Routine in Hex eintippen. Der Monitor ist aber nicht als Entwicklungssystem für Assemblerprogramme gedacht.

Welche Hilfsmittel brauchen Sie zusätzlich?

Absolut notwendig ist ein Assemblerprogramm, das Ihnen einige üble Arbeiten abnimmt, die das Maschinenprogrammieren zur Qual werden lassen. Mit einem guten Assembler und der entsprechenden Übung kann Maschinenprogrammieren eine Komfortebene erreichen, die durchaus mit BASIC vergleichbar ist.

Dies stimmt aber nur für die Erstellung und Übersetzung des Programms. Beim Testen von Maschinenprogrammen zeigt sich ein ganz wesentlicher Unterschied zum Programmieren in BASIC: Ein Maschinenprogramm verzeiht keinen Fehler. Es meldet nicht mit Klartext immer neue Fehler, sondern springt im Glücksfall in den Monitor und in den weitaus häufigeren Fällen 'in den Wald', der Computer ist 'tot'. Dann hilft nur noch Abschalten. Nach dem Einschalten ist aber leider alles weg und man muß mehr oder weniger mühsam den nächsten Test vorbereiten.

Zum Testen benötigt man also wieder Hilfsprogramme, die - je nach Güte - verhindern, daß der Computer 'abstürzt'. Dennoch ist das Austesten von Maschinenprogrammen immer schwieriger als von BASIC-Programmen.

Zum Teil kann man das aber durch sauberen und gewissenhaften Programmierstil ausgleichen, indem einfach manche Fehler gar nicht erst ins Programm eingebaut werden.

Das dritte Hilfsmittel sind Unterlagen über das Betriebssystem und den Interpreter. Die Betriebssoftware Ihres Computers kann sehr viel und Sie würden viel Zeit verlieren, wenn Sie dies alles selbst programmieren wollten.

Alle drei angesprochenen Hilfsmittel können Sie demnächst über Ihren Commodore-Fachhändler beziehen. Er gibt Ihnen auch Auskunft darüber, ob in Ihrer Nähe Assembler-Schulungskurse durchgeführt werden. Vor allem aber erkundigen Sie sich nach dem Assembler-Handbuch zu Ihrem Computer. Für die Einführung in die Befehle des Prozessors existieren bereits zwei Handbücher über Programmieren und Hardware.

Das Erlernen der Assembler-Programmierung

Die Sprache des Prozessors sieht zwar ganz anderes als BASIC und ist in der Datenbehandlung nicht so mächtig. Aber sie ist doch eine Programmiersprache mit Anweisungen für bedingte und unbedingte Sprünge, mit Unterprogrammaufrufen und mit Befehlen für Addieren und Subtrahieren. Die Logik ist also ähnlich.

Überschlägig kann man deshalb sagen, daß Sie Assembler etwa in der gleichen Zeit erlernen können, wie BASIC. Allerdings werden Sie eher länger brauchen, als weniger lang.

Über die Befehle

Die fünf Befehle zum Verkehr mit der Maschinenebene werden im folgenden nur nach Format und Parameter vorgestellt. Wir verzichten auf umfangreiche Beispiele, weil sie doch nur ein Tropfen auf den heißen Stein wären.

13.2 POKE, PEEK (Speicherzelle beschreiben/lesen)

Zweck

POKE schreibt in eine bestimmte Adresse einen Byte-Wert.

PEEK liest den Inhalt einer bestimmten Adresse.

Format

POKE ad , by

PEEK (ad)

ad Adresse
by Byte-Ausdruck

PEEK übergibt den Inhalt der Adresse als Byte-Wert.

Beispiele

POKE 32768,0	schreibt den 'Klammeraffen' links oben in den Bildschirm (s. BSC)
?PEEK(32768)	übergibt den BSC des ersten Zeichens im Bildschirm

13.3 SYS (Maschinenprogrammaufruf)

Zweck

SYS arbeitet ein Maschinenprogramm ab der angegebenen Adresse ab.

Format

SYS (adresse)	(die Klammern können entfallen)

Beispiel

SYS 1024	springt in den Monitor

13.4 USR( ) (Maschinenprogrammaufruf mit Parameterübergabe)

Zweck

USR übergibt den hinter USR angegebenen Wert an den FAC und springt zu der in den Zellen (1,2) angegebenen Adresse. Bei der Rückkehr wird als Ergebnis von USR der Wert des FAC übergeben.

Format

USR (gleitkommaausdruck)

Beispiel

A = USR (B) übergibt den Wert von B in den FAC, springt über den Zeiger in (1,2), arbeitet das entsprechende Maschinenprogramm ab und übergibt den Inhalt des FAC an A.

13.5 WAIT (Warte auf Bitmuster)

Zweck

Durch WAIT wird das BASIC-Programm blockiert, bis in einer bestimmten Speicherzelle ein bestimmtes Bitmuster vorgefunden wird.

Format

WAIT ad , eor , and

ad Adresse
eor EOR-Operand
and AND-Operand

eor und and sind Integer-Ausdrücke.
Für and wird 0 angenommen, wenn nichts angegeben wird.

Anmerkungen

Der Wert in 'ad' wird mit 'eor' durch 'EXCLUSIV-ODER' verknüpft und anschließend mit 'and' durch UND.

Wenn das Ergebnis 0 ist, wird der Wert in ad nochmal gelesen und behandelt. Wenn das Ergebnis von 0 verschieden ist, also mindestens ein Bit gesetzt ist, fährt BASIC mit der Abarbeitung der nächsten Anweisung fort.

Es gibt keine Möglichkeit, WAIT zu unterbrechen. Die verlangte Bedingung muß also eintreten, sonst muß der Rechner abgeschaltet werden.

14. Fehlermeldungen

14.1 Einführung

Mit knapp 30 Fehlermeldungen im Klartext versucht der Computer Ihnen klarzumachen, was Sie vermutlich oder sicher falsch gemacht haben, bzw. was er nicht versteht. Mit wenigen Ausnahmen sind die Fehlermeldungen so klar definiert, daß Sie den Fehler sofort finden können.

Allen Fehlermeldungen ist folgendes gemeinsam:

Wenn der Fehler im Direktmodus auftritt, ist das letzte Wort der Fehlermeldung ERROR. Wird der Fehler dagegen im Programm gefunden, wird ERROR gefolgt von der Zeilenangabe IN ..... (Ausnahme: REDO FROM START / EXTRA IGNORED)

Danach meldet der Computer durch READY, daß er bereit ist, neue Anweisungen zu empfangen.

Alle Variablenwerte bleiben erhalten, so daß Sie durch den Ausdruck der beteiligten Variablen den Fehler einkreisen können.

Der Stapel wird bei allen Fehlermeldungen (ohne REDO ... / EXTRA ...) geleert, so daß GOSUB- und FOR NEXT - Strukturen zerstört sind. Dies ist auch der Grund, warum CONT nach Fehlern nicht zugelassen ist.

14.2 Übersicht

Fehlermeldungen:

1) BAD DISK	Schlechte Diskette
2) BAD SUBSCRIPT	Falsche Dimensionierung
3) CAN'T CONTINUE	Kann nicht weitermachen
4) DEVICE NOT PRESENT	Kein Gerät am IEC-Bus
5) DIVISION BY ZERO	Division durch Null
6) FILE DATA	Datei-Daten-Fehler
7) FILE OPEN	Datei ist bereits offen
8) FILE NOT FOUND	Datei wurde nicht gefunden
9) FILE NOT OPEN	Datei ist nicht offen
10) FORMULA TOO COMPLEX	Zu komplizierter Ausdruck
11) ILLEGAL DIRECT	Unerlaubtes Direktkommando
12) ILLEGAL QUANTITY	Unerlaubter Wert
13) NEXT WITHOUT FOR	NEXT ohne FOR
14) NOT INPUT FILE	Keine Eingabedatei
15) NOT OUTPUT FILE	Keine Ausgabedatei
16) OUT OF DATA	Zuwenig Daten in DATAs
17) OUT OF MEMORY	Zuwenig Speicher
18) OVERFLOW	Überlauf des Zahlenbereichs
19) RETURN WITHOUT GOSUB	RETURN ohne GOSUB
20) REDIM'D ARRAY	Feld ist bereits dimensioniert
21) STRING TOO LONG	String ist zu lang
22) SYNTAX	Fehlerhafter Befehl
23) TOO MANY FILES	Zu viele Dateien
24) TYPE MISMATCH	Typen Durcheinander
25) UNDEF'D STATEMENT	Sprungziel nicht vorhanden
26) UNDEF'D FUNCTION	Funktion wurde nicht definiert

Hinweise/Fragen:

27) REDO FROM START	Alles nochmal eingeben
28) EXTRA IGNORED	Zusätzliches Datenelement wurde ignoriert
29) ARE YOU SHURE?	Sind Sie sicher?

14.3 Beschreibung der Fehlermeldungen

1) **BAD DISK** (schlechte Diskette)

Kann nur nach dem Befehl 'HEADER' im Direktmodus auftreten.

Bei jeder Floppy-Fehlermeldung, die während dem Formatieren auftritt, wird diese Meldung gebracht. Sie kann also z.B. auch auftreten, wenn gar keine Diskette im Laufwerk ist.

2) **BAD SUBSCRIPT** (falsche Dimensionierung)

(s. DIM)

Eine indizierte Variable wurde verwendet, die in dieser Größe nicht dimensioniert wurde, bzw. überhaupt nicht.

Durch einen Schreibfehler bei Funktionen, die einen Zahlenparameter in Klammern benötigen, kann ebenfalls diese Meldung auftreten: Man wollte ABS(A) schreiben und hat AB(A) geschrieben. Bei A größer als 10 wird BAD SUBSCRIPT gemeldet.

3) CAN'T CONTINUE (kann nicht weitermachen)

Kann nur bei CONT auftreten.

Nach einer Fehlermeldung oder einer Programmänderung kann nicht durch CONT weitergemacht werden.

4) DEVICE NOT PRESENT (Kein Gerät am IEC-Bus)

Kann bei OPEN, CLOSE, INPUT§, GET§, PRINT§, CMD, LOAD, SAVE und VERIFY auftreten.

Diese Meldung bedeutet lediglich, daß entweder kein Gerät am Bus angeschlossen ist, oder daß keines eingeschaltet ist. Sie bezieht sich nicht nur auf das Gerät, bei dessen Datei der Fehler gemeldet wird.

Wenn also mindestens ein Gerät am Bus eingeschaltet ist, wird nicht DEVICE NOT PRESENT gemeldet, auch wenn das betreffende Gerät tatsächlich nicht vorhanden ist (s. Status).

5) DIVISION BY ZERO (Division durch Null)

Kann nur durch '/' ausgelöst werden.

Durch 0 darf nicht dividiert werden.

6) FILE DATA (Datei-Daten-Fehler)

Diese Meldung kann nur durch INPUT§ oder GET§ ausgelöst werden.

In eine numerische Variable (Gleitkomma oder Integer) sollten nichtnumerische Zeichen aus einer Datei gelesen werden.

Schwer zu finden ist der Fehler, wenn durch fehlende oder überzählige Trennzeichen (Delimiter) die Anzahl der Elemente in der Datei anders ist, als man denkt.

Eine weitere Ursache wäre einfach, daß die falsche Datei gelesen wurde.

7) FILE OPEN (Datei ist bereits offen)

Kann nur durch 'OPEN' ausgelöst werden.

Durch OPEN sollte eine Datei geöffnet werden, die schon geöffnet ist.

Entweder wurde die Datei noch nicht geschlossen, oder man hat aus Versehen eine schon benützte logische Dateinummer nochmal verwendet.

Wichtiger Hinweis:

Durch diese Fehlermeldung werden **alle Dateien geschlossen,** allerdings nicht wirklich, wie bei CLOSE, sondern nur der Zeiger in die OPEN-Tabelle wird auf 0 gesetzt.

Dadurch, daß die betreffenden Peripheriegeräte kein CLOSE übermittelt bekommen, können sich beim Testen Probleme ergeben. Eine Lösung (beim Testen) ist, den Zeiger in die Tabelle wieder auf die Anzahl der geöffneten Dateien zu setzen: Nach 'POKE 174,anzahl der dateien' sind alle Dateien wieder 'offen'.

8) FILE NOT FOUND (Datei wurde nicht gefunden)

Kann auftreten bei LOAD und VERIFY von Floppy und Rekorder, sowie beim OPEN zum Lesen beim Rekorder.

Beim Rekorder bedeutet diese Meldung, daß ein EOT-Block gelesen wurde.

Beim Floppy wird FILE NOT FOUND bei jedem Floppy-Fehler gemeldet, der den Zugriff auf das Programm verhindert hat.

9) FILE NOT OPEN (Datei ist nicht offen)

Kann nur durch INPUT , GET oder PRINT ausgelöst werden.

Durch einen dieser Befehle sollte eine logische Dateinummer angesprochen werden, der noch nicht durch OPEN ein Gerät zugeordnet wurde.

Möglich ist natürlich auch, daß die Zuordnung zwar erfolgt ist, die Datei aber inzwischen durch CLOSE geschlossen wurde.

Im einfachsten Fall hat man sich nur bei der Angabe der logischen Dateinummer geirrt.

Wichtiger Hinweis:

Durch diese Fehlermeldung werden **alle Dateien geschlossen** (s. FILE OPEN ERROR).

10) FORMULA TOO COMPLEX (Zu komplizierter String-Ausdruck)

Kann nur durch Stringausdrücke ausgelöst werden, an denen zu viele Teilstrings beteiligt sind.

Wenn diese Meldung auftritt, muß man den Ausdruck, der sie hervorgerufen hat, in zwei oder mehrere Ausdrücke aufteilen, wobei man die Zwischenergebnisse in Variablen zwischenspeichern muß.

11) ILLEGAL DIRECT (unerlaubtes Direktkommando)

Diese Meldung kann nur durch GET, INPUT oder DEF FN( ) ausgelöst werden und das nur im Direktmodus.

Dies sind die einzigen drei Anweisungen, die im Direktmodus nicht erlaubt sind. Der Grund liegt darin, daß sie den BASIC-Eingabe-Puffer benützen, der aber auch zur Auswertung der Direkt-Befehle verwendet wird.

12) ILLEGAL QUANTITY (unerlaubter Wert)

Diese Meldung kann viele Ursachen haben. Grundsätzlich wurde der Zahlenbereich einer Anweisung bzw. einer Variablen überschritten. Man kann vier Haupt-Bereiche unterscheiden. Bitte lesen Sie die Definitionsbereiche bei Datenarten nach. Unter den einzelnen Bereichen sind im folgenden stichwortmäßig die Zusammenhänge angeführt, in denen die Bereiche benötigt werden.

Byte-Bereich:	ON, WAIT, POKE, OPEN, TAB, SPC, ASC, CHR$, LEFT$, MID$, RIGHT$
Integer-Bereich:	Integer-Variablen, NOT, AND, OR
Adressbereich:	WAIT, POKE, PEEK, SYS
Mathemat. Bereich:	Bei SQR, LOG und ATN sind die mathematischen Einschränkungen zu beachten.

Die Disk-Befehle haben eigene Grenzen für die Parameter, z.B. tritt ILLEGAL QUANTITY bei DOPEN für Gerätenummern größer als 31 auf.

13) NEXT WITHOUT FOR (NEXT ohne FOR)

Kann nur durch NEXT ausgelöst werden.

Logische Ursache ist meistens ein falscher Sprung.

14) NOT INPUT FILE (keine Eingabedatei)

Diese Meldung kann nur durch GET oder INPUT in Zusammenhang mit OPEN auf Gerät 1 oder 2 (Rekorder) ausgelöst werden.

Bei OPEN auf Gerät 1 oder 2 bedeutet der dritte Parameter schreiben oder lesen (0 = lesen, 1,2 = schreiben).

Wenn der dritte Parameter nicht 0 ist, wurde also eine Schreibdatei geöffnet, auf die lesend zugegriffen werden soll.

15) NOT OUTPUT FILE (keine Ausgabedatei)

Kann nur durch PRINT in Zusammenhang mit OPEN auf Gerät 1 oder 2 (Rekorder) ausgelöst werden.

Bei OPEN auf Gerät 1 oder 2 bedeutet der dritte Parameter schreiben oder lesen (0 = lesen, 1,2 = schreiben).

Wenn der dritte Parameter 0 ist, wurde also eine Lesedatei geöffnet, auf die schreibend zugegriffen werden soll.

16) OUT OF DATA (zuwenig Daten in DATAs)

Kann nur durch READ ausgelöst werden (s. auch DATA).

17) OUT OF MEMORY (zuwenig Speicher)

Für diese Meldung gibt es verschiedene Ursachen (s. Speichereinteilung, Stack). Drei Gruppen sind zu unterscheiden:

a) Tatsächlich zu wenig Speicher:

Programm + Variable + Stringinhalte sind zu groß für den vorhandenen Arbeitsspeicher (RAM). Grundsätzlich kann diese Meldung dann durch fast jede Anweisung ausgelöst werden.

b) Zu viele geschachtelte Schleifen (FOR-NEXT), GOSUBs und Klammern.

(s. Stack)

c) Laden von Maschinenroutinen:

Durch Laden von Maschinenroutinen können völlig unsinnige Werte in den BASIC Zeigern stehen. Das Betriebssystem glaubt dann, der Speicher wäre voll, obwohl dies nicht der Fall ist. Abhilfe schafft z.B. 'NEW'.

18) OVERFLOW (Überlauf des Zahlenbereichs)

(s. Datenarten/Gleitkomma)

19) RETURN WITHOUT GOSUB (RETURN ohne GOSUB)

Kann nur durch 'RETURN' ausgelöst werden. (s. RETURN und GOSUB)

20) REDIM'D ARRAY (Feld ist bereits dimensioniert)

Kann nur durch DIM ausgelöst werden.

Ein Feld (Matrix) darf nur einmal dimensioniert werden.

Beachten Sie vor allem, daß auch nach der automatischen Dimensionierung das Feld nicht nochmal dimensioniert werden kann.

21) STRING TOO LONG (String ist zu lang)

Kann in Zusammenhang mit Stringoperationen auftreten bzw. speziell bei INPUT§ und den Disk-Befehlen.

Ein String kann nicht mehr als 255 Zeichen enthalten.

Bei den Disk-Befehlen liegt die Grenze beim Dateinamen bei 16 Zeichen.

Bei INPUT§ wird STRING TOO LONG gemeldet, wenn versucht wird, mehr als 80 Zeichen ohne CR einzulesen.

22) SYNTAX (Format-Fehler)

Diese Meldung tritt immer dann auf, wenn der Interpreter etwas nicht versteht. Das kann daher kommen, daß man einen Befehl verwendet hat, der nicht existiert. Meistens hat man sich einfach verschrieben oder ein Zeichen zu wenig oder zu viel geschrieben.

Bei Klammern ist darauf zu achten, daß sie immer paarweise auftreten. D.h. in einem Ausdruck müssen ebensoviele öffnende wie schließende Klammern vorkommen.

Weitere häufige Fehler sind vergessene oder überflüssige Kommas oder Doppelpunkte.

Eine Ausnahme bildet SYNTAX ERROR bei READ bzw DATA und bei GET auf eine Zahlenvariable.

Auch bei BASIC-Zeilennummern größer als 63999 wird SYNTAX ERROR gemeldet!

Manchmal wird SYNTAX ERROR gemeldet, obwohl man einen ILLEGAL QUANTITY erwarten würde.

23) TOO MANY FILES (Zu viele Dateien)

Kann nur bei OPEN auftreten.

Maximal 10 Dateien können geöffnet sein. Bei der 11-ten wird TOO MANY FILES ERROR gemeldet.

Auch bei dieser Meldung werden **alle Dateien geschlossen** (s. FILE OPEN).

24) TYPE MISMATCH (Typen Durcheinander)

Strings und Zahlen sind vollkommen verschiedene Datenarten. Man kann also nicht einer Stringvariablen das Ergebnis eines Zahlenausdrucks zuweisen und umgekehrt. Das gleiche gilt für alle Anweisungen, die Parameter benötigen.

Da aber durchaus in einem Ausdruck Strings und Zahlen gleichzeitig vorkommen können, kann keine Liste mit Ursachen angegeben werden. Trotzdem sollen einige Beispiele angeführt werden:

```
A$ = B
A$ = 123
A  = "123"
MID$ ( A , 1 , 1 )
MID$ (A$ , B$, 2 )
ASC (AB)
```

25) UNDEF'D STATEMENT (Sprungziel nicht vorhanden)

Kann nur durch RUN..., (ON) GOTO..., GOSUB... oder THEN ausgelöst werden.

In jedem Fall wurde eine BASIC Zeilennummer angegeben, die im Programm nicht existiert.

26) **UNDEF'D FUNCTION (Funktion wurde nicht definiert)**

Kann nur durch 'FN(' ausgelöst werden.

Die zugehörige Funktion wurde nicht durch DEF FN (..) definiert.

Hinweise und Fragen

27) **REDO FROM START (alles nochmal eingeben)**

Diese Meldung tritt nur bei INPUT auf und unterbricht das Programm nicht.

Der Anwender hat im Bildschirmdialog ein nichtnumerisches Zeichen eingegeben, obwohl nur Zahlen zugelassen sind, weil hinter dem entsprechenden INPUT eine Zahlenvariable steht.

28) **EXTRA IGNORED (zusätzliches Datenelement wurde ignoriert)**

Diese Meldung tritt nur bei INPUT auf und unterbricht das Programm nicht.

Der Anwender hat im Bildschirmdialog ein Komma oder einen Doppelpunkt eingegeben, obwohl dies nicht verlangt war.

29) **ARE YOU SHURE (Sind Sie sicher?)**

Diese Meldung tritt im Direktmodus bei HEADER und SCRATCH auf. Sie soll sicherstellen, daß nicht aus Versehen Disketten oder Dateien gelöscht werden. Die Frage ist mit **N** für 'nein' bzw. **Y** für 'ja' (YES) zu beantworten.

III. BILDSCHIRMVERWALTUNG UND TASTATURBEHANDLUNG

Vorbemerkung: In den Erklärungen werden folgende Steuercodes in Stringkonstanten wie folgt dargestellt:

CURSOR NACH RECHTS	R	HOME	H
CURSOR NACH LINKS	L	CLR	C
CURSOR NACH UNTEN	U	RVS-Ein	E
CURSOR NACH OBEN	O	RVS-Aus	A

1. Einführung

Ihr Computer verfügt über eine leistungsfähige Bildschirmverwaltung und eine gepufferte Tastaturabfrage. Die grundsätzlichen Dinge, die auch der Anwender wissen muß, sind ganz am Anfang des Handbuches beschrieben. Der Teil III wendet sich an den Programmierer, für den tieferes Verständnis der Zusammenhänge erforderlich ist, wenn er alle Möglichkeiten des Tastatur-Bildschirm-Verkehrs ausnützen will.

Grundsätzlich sind zwei verschiedene Betriebsarten des Computers zu unterscheiden, was die Eigenschaften des Tastatur-Bildschirm-Verkehrs anbelangt:

a) **Eingabe**-Modus

Der Cursor blinkt auf dem Bildschirm und kann durch die Kontrolltasten über der Bildschirm bewegt werden. Aus zwei verschiedenen Gründen kann der Rechner in diesem Zustand sein:

a1) Der Computer ist im 'READY'-Zustand, es läuft also kein BASIC-Programm, weder als echtes Programm, noch als Direktmodus-Programm. In diesem Zustand ist der Computer bereit, entweder Anweisungen im 'Direktmodus' auszuführen oder Programmzeilen zu übernehmen.

a2) Der Computer hat im BASIC-Programm ein INPUT gefunden und wartet nun auf eine Eingabe. Wie erwähnt, ist das Bildschirmverhalten identisch mit a1. Nach Drücken von RETURN wird aber immer die Eingabe an die Variable hinter INPUT übergeben. Beachten Sie dazu die Eigenschaften von INPUT!

b) **Programm**-Modus

Ein BASIC-Programm läuft gerade. Kennzeichen dieses Modus sind, daß der Cursor nicht sichtbar ist und daß das Programm durch die STOP-Taste unterbrochen werden kann. In diesem Modus werden gedrückte Tasten nur im Tastaturpuffer registriert, führen aber nicht unmittelbar zu einer Reaktion des Programms oder des Bildschirms.

Nach dieser kurzen Vorstellung der drei Zustände des Rechners soll gezeigt werden, daß diese aus den gleichen Funktionsbausteinen zusammengesetzt sind. Dies soll Ihnen das Verständnis der auf den ersten Blick verwirrenden 'Verhaltens-Vielfalt' der Eingabezustände erleichtern:

Bei **Tastaturpuffer** ist beschrieben, daß für jede gedrückte Taste der ihr zugeordnete ASC-Wert in den Tastaturpuffer übernommen wird. Falls sich der Computer im Modus (a) befindet, wird der Code sofort aus dem Puffer genommen und das entsprechende Zeichen auf den Bildschirm geschrieben. Mit Hilfe der einzelnen Steuertasten kann der Bildschirm prinzipiell in beliebiger Weise beschrieben werden.

Bitte beachten Sie aber, daß die Bildschirmverwaltung die Zeichen, die auf den Bildschirm geschrieben werden, überhaupt nicht interpretiert (Ausnahme: Anführungsstriche). Erst durch die RETURN-Taste wird der Inhalt der aktuellen Bildschirmzeile übernommen.

Ganz anders reagiert der Computer im Modus (b). Hier wird nur jedes Zeichen in den Tastaturpuffer übernommen, aber weiter wird nichts gemacht. Sie haben dann zwei Möglichkeiten, die gedrückten Tasten zu verarbeiten. Zuerst müssen Sie mit **GET** jedes Zeichen aus dem Puffer in eine Stringvariable holen. Anschließend können Sie (müssen aber nicht) durch **PRINT** das Zeichen auf den Bildschirm drucken.

Das folgende kleine Programm zeigt Ihnen, daß es für den Bildschirmeditor prinzipiell gleichgültig ist, ob er Zeichen unmittelbar aus dem Tastaturpuffer übernimmt (Modus a) oder ob er sie durch PRINT übermittelt bekommt (Modus b):

```
10 GET G$ : PRINT G$; : GOTO 10
```

Wenn Sie nach RUN beliebige Tasten drücken, werden die Zeichen genauso ausgedruckt, als ob kein Programm laufen würde. Der einzige Unterschied besteht darin, daß der Cursor nicht blinkt, also die Schreibstelle nicht sichtbar ist, und die RETURN-Taste keine Übernahme-Wirkung hat.

2. Tasten, die nur als Tasten wirken

Für einige Tasten gilt nicht, daß deren Code - mit PRINT geschickt - dieselbe Wirkung hat, wie wenn die Taste im Eingabemodus gedrückt wird.

Die Taste SHIFT erzeugt überhaupt keinen ASC-Code, sondern wird nur vom Betriebssystem wahrgenommen, solange sie gedrückt sind.

Die Taste STOP erzeugt zwar keinen ASC-Code, jedoch wird die im folgenden beschriebene Sonderfunktion wieder durch unmittelbare Reaktion auf den Tastendruck - nicht auf den Code - ausgelöst.

Schließlich bleiben noch die Tasten RETURN und RUN, deren Sonderwirkung nur im Eingabemodus zur Geltung kommt, deren Code aber in den Tastaturpuffer gelegt werden kann (s. **Tastaturpuffer**).

2.1 SHIFT

Durch gleichzeitiges Drücken einer SHIFT-Taste und einer weiteren Taste wird wie bei der Schreibmaschine deren 'oberer Wert' übernommen. Obwohl die Anwendung von SHIFT an sich keine Probleme bringt, sollten Sie folgende Hinweise beachten:

a) Die Taste **SHIFT LOCK** blockiert gewissermaßen die (linke) SHIFT-Taste. Beachten Sie dabei, daß dann nur noch die 'obere Funktion' der Steuertasten zur Verfügung steht. Insbesondere bei RETURN kann dies böse Folgen haben (s. SHIFT RETURN).

b) Bei Buchstaben liegt der ASC-Code bei nicht gedrückter SHIFT-Taste unter 128, bei gedrückter SHIFT-Taste über 128. Dies ist unabhängig von der Darstellung auf dem Bildschirm der Fall.

2.2 RETURN

RETURN hat **zwei verschiedene Funktionen:**

1) **Bildschirmzeile übernehmen:**

Nur im Eingabemodus wird die aktuelle Bildschirmzeile an das Betriebssystem zur Auswertung übergeben (s. INPUT und Programmeditierung).

Falls sich der Computer im **Programmodus** befindet, hat RETURN zunächst keine Sonderfunktion, sondern bringt nur den Code 13 in den Tastaturpuffer. Solange dieses CR nicht aus dem Tastaturpuffer geholt wird, zeigt es überhaupt keine Wirkung. Sollte dieser Code vom Programm mit GET aus dem Puffer geholt und auf den Bildschirm 'gedruckt' werden (PRINT), so hat das nur die Wirkung, daß der Curosr am Anfang der nächsten Bildschirmzeile aufgesetzt wird.

2) **Cursor am Anfang der nächsten Bildschirmzeile aufsetzen:**

Der Cursor wird am Anfang der nächsten Bildschirmzeile aufgesetzt. Diese zweite Funktion wird auch von SHIFT-RETURN ausgeführt.

2.3 RUN / STOP

STOP unterbricht ein gerade laufendes Programm. STOP hat im Eingabemodus keine Wirkung.

RUN schreibt 'DI"*' auf den Bildschirm und lädt dadurch das erste Programm des Laufwerks 0 des Gerätes 8 in den Arbeitsspeicher (sofern das Floppy mit DOS 2. ausgestattet ist). Anschließend wird dieses Programm durch 'Ru' automatisch gestartet.

So praktisch beide Funktionen sind, wenn man sie braucht, so unangenehm wirken sie, wenn man diese Tasten aus Versehen drückt. STOP ist dabei noch relativ einfach (durch CONT) rückgängig zu machen.

Dagegen kann RUN das derzeit im Speicher stehende Programm löschen, indem es das erste im Laufwerk 0 lädt. Da dies wirklich ärgerlich sein kann, sollten Sie darauf achten, daß Sie die Taste RUN/STOP nicht aus Versehen drücken.

Bei RUN soll noch eine weitere Wirkung erwähnt werden: Wenn Sie RUN drüken, während Sie eine Programmzeile eingeben, steht anschließend 'DI"*' in dieser Zeile!

2.4 Verlangsamen des Bildschirmrollens (RVS)

Solange Sie die Taste **RVS** gedrückt halten, wird das Hochrollen des Bildschirm so verlangsamt, daß überschlägiges Mitlesen möglich wird.

3. Steuertasten / Steuercodes

Die in diesem Kapitel aufgeführten Steuercodes wirken unabhängig davon, ob sie durch Tastendruck oder durch PRINT zum Bildschirmeditor gelangen, in gleicher Weise. Deshalb wird zu jeder Taste auch der Code angegeben. Auf die Darstellung im CONTROL-Modus wird verzichtet, da sie je nach Graphik- oder Textmodus verschieden ist! Bitte entnehmen Sie die richtigen Zeichen der ASC-Code-Tabelle.

Die meisten Steuertasten wirken nur innerhalb einer Zeile und können deshalb auch bei Eingaben durch INPUT verwendet werden.

Mit den Tasten für vertikale Cursorbewegung können Sie aber von Zeile zu Zeile wechseln, und mit den Tasten HOME/CLR sogar den ganzen Bildschirm löschen. Beachten Sie bitte, daß im INPUT-Eingabemodus durch Verwendung dieser Tasten in der Regel Probleme entstehen (s. INPUT). Diese Tasten sind also meistens nur im Direktmodus sinnvoll. Dies gilt natürlich nur, solange nicht ein Programm diese Tasten als Sonderfunktionstasten (z.B. durch GET) verwendet.

3.1 CURSOR NACH RECHTS (CNR)

Taste: obere Reihe / zweite Taste von rechts / **ohne** SHIFT

ASC: 29

CNR bewegt den Cursor um eine Stelle nach rechts. Wenn er sich am Ende der Bildschirmzeile befindet, so setzt er am Anfang der darunterliegenden Zeile auf. Wenn er sich bereits am Ende der letzten Bildschirmzeile befindet, wird der Bildschirminhalt um eine Zeile hochgerollt.

3.2 CURSOR NACH LINKS (CNL)

Taste: obere Reihe / zweite Taste von rechts / **mit** SHIFT

ASC: 157 (29+128)

CNR bewegt den Cursor um eine Stelle nach links. Wenn er sich am Anfang der Bildschirmzeile befindet, setzt er am Ende der darüberliegenden Zeile auf. Wenn er sich bereits am Anfang der ersten Bildschirmzeile befindet, hat CNL keine Wirkung.

3.3 DELETE (DEL) delete = löschen

Taste: obere Reihe / erste Taste von rechts / **ohne** SHIFT

ASC: 20

DEL stellt den Cursor um eine Stelle nach links und zieht dabei den rechten Teil der Zeile ab einschließlich der Cursorposition nach links. Dadurch wird das Zeichen links von der alten Cursorposition überschrieben. Am Ende der Zeile wird ein SPACE nachgezogen.

Befindet sich der Cursor bereits am Anfang der Bildschirmzeile, so geht zwar der Cursor ans Ende der darüberliegenden Zeile, aber der Inhalt der Zeile, die er eben verlassen hat, wird nicht mitgezogen. DEL wirkt also nur innerhalb einer Bildschirmzeile.

3.4 INSERT (INS) insert = einfügen

Taste: obere Reihe / erste Taste von rechts / **mit** SHIFT

ASC: 148 (20+128)

INS schiebt den rechten Teil der Zeile um eine Stelle nach rechts, der Cursor bleibt aber an seiner alten Position stehen, so daß sofort nach dem Aufschieben der entstandene Platz durch neue Zeichen aufgefüllt werden kann.

Wenn der rechte Teil der Zeile (genauer: ein Zeichen, das kein SPACE ist) am rechten Zeilenende anstößt, wird nicht mehr weitergeschoben.

Für den Bereich, den INST aufgeschoben hat, gilt **CONTROL**-Modus! Durch RETURN oder SHIFT RETURN wird der CONTROL-Modus wieder aufgehoben.

3.5 Anführungszeichen (")

Taste: obere Reihe / dritte Taste von links / **ohne** SHIFT

ASC: 34

" schaltet den Bildschirmeditor für die aktuelle Zeile in den **CONTROL**-Modus. Durch RETURN, SHIFT RETURN oder ein weiteres " wird der CONTROL-Modus wieder aufgehoben.

3.6 RVS (REVERSE ON) **reverse = invertiert, umgekehrt**

Taste: zweite Reihe von oben / erste Taste von links / **ohne** SHIFT

ASC: 18

Nachdem der Bildschirmeditor den Code von RVS ON empfangen hat, bringt er alle weiteren Zeichen in umgekehrter Darstellung, also dunkel auf hellem Grund.

Der RVS-Modus bleibt solange eingestellt, bis entweder RVS-OFF, RETURN oder SHIFT RETURN gedruckt bzw. gedrückt wird.

3.7 RVS (REVERSE OFF)

Taste: zweite Reihe von oben / erste Taste von links / **mit** SHIFT

ASC: 146 (18+128)

RVS-OFF ist eine der Möglichkeiten, den **REVERS**-Modus aufzuheben.

3.8 SPACE (Leertaste)

Taste: breite Taste unter dem Haupttastenfeld

ASC: 32 oder 160

SPACE schreibt ein Zeichen auf den Bildschirm, das keinen einzigen Leuchtpunkt enthält, man 'sieht' also eine Lücke. Allerdings ist SPACE nicht 'kein Zeichen', sondern ein Zeichen wie jedes andere, das aber nicht leuchtet. Daß ein SPACE ein normales Zeichen ist, sieht man deutlich, wenn man es 'revers' druckt - dann ist es nämlich ein heller Fleck ohne einen dunklen Punkt.

Dennoch hat SPACE eine Sonderstellung: Wenn der Bildschirm 'gelöscht' wird, enthält er lauter SPACEs, bei DEL oder INS sind die 'leeren' Stellen, die eingefügt werden, SPACEs. Auch bei INPUT ist beschrieben, daß SPACEs u.U. unterdrückt werden. In manchen Zusammenhängen wird SPACE also tatsächlich wie 'kein Zeichen' behandelt. Das ändert aber nichts daran, daß jede 'leere Stelle' auf dem **Bildschirm** den **BSC**-Code 32 bzw. 96 hat - sowie jedes SPACE in einem **String** den **ASC**-Code 32 bzw. 160.

Noch eine Sonderstellung des SPACE ist wichtig: Bei jedem anderen Zeichen ist die Darstellung des Zeichens und des entsprechenden 'SHIFT-Zeichens' optisch verschieden. Bei SPACE und SHIFT-SPACE ist dies nicht so. Beide Zeichen sehen gleich aus, obwohl sie eindeutig verschiedene Codes haben. Da aber andererseits die vorher erwähnte Sonderstellung des SPACE nur für den Code 32 gilt, kann es zu Mißverständnissen zwischen dem Auge und dem Rechner kommen.

3.9 CNU (CURSOR NACH UNTEN)

Taste: obere Reihe / dritte Taste von rechts / **ohne** SHIFT

ASC: 17

CNU bewegt den Cursor in der gleichen Spalte um eine Zeile nach unten. Wenn sich der Cursor bereits in der letzten Bildschirmzeile befindet, wird der Bildschirminhalt dadurch um eine Zeile hochgerollt.

3.10 CNO (CURSOR NACH OBEN)

Taste: obere Reihe / dritte Taste von rechts / **mit** SHIFT

ASC: 145 (17+128)

CNO bewegt den Cursor in der gleichen Spalte um eine Zeile nach oben. Wenn sich der Cursor bereits in der ersten Bildschirmzeile befindet, hat CNO keine Wirkung.

3.11 HOME (CURSOR IN OBERE LINKE BILDSCHIRMECKE)

Taste: obere linke Taste über Zehnertastenfeld / **ohne** SHIFT

ASC: 19

HOME bringt den Cursor in die obere linke Bildschirmecke.

3.12 CLR (Bildschirm löschen)

Taste: obere linke Taste über Zehnertastenfeld / **mit** SHIFT

ASC: 147 (19+128)

CLR löscht den Bildschirminhalt. Exakter ausgedrückt, wird dabei der Bildschirm mit SPACEs (ASC 32) bedruckt.

3.13 SHIFT-RETURN (an Anfang der nächsten Zeile gehen)

Taste: zweite Reihe von unten / erste Taste von rechts / **mit** SHIFT

ASC: 141 (13+128)

SHIFT-RETURN hat im Gegensatz zu RETURN (ohne SHIFT) nicht die Wirkung, daß eine Bildschirmzeile dem Betriebssystem zur Auswertung übergeben wird, sondern setzt nur den Cursor am Anfang der nächsten Bildschirmzeile auf.

Wenn der Cursor sich bereits in der letzten Zeile befindet, rollt der Bildschirminhalt um eine Zeile hoch.

4. Bildschirm-Verwaltung

Da der Bildschirm nur 40 Zeichen pro Zeile hat, man aber die Möglichkeit hat, bis zu 80 Zeichen mit einer Eingabe zu übernehmen, wird die Organisation des Bildschirms laufend geändert.

Nach dem Löschen des Bildschirms sind alle Zeilen mit einer Länge von 40 Zeichen definiert. Das kann man nachprüfen, indem man mit RETURN oder SHIFT RETURN den Cursor von oben nach unten bewegt. Sobald man aber in die letzte Stelle einer Zeile ein Zeichen schreibt, wonach der Cursor an den Anfang der nächsten Zeile springt, wird die Aufteilung der Bildschirmzeilen geändert. Die zwei angesprochenen Bildschirmzeilen werden jetzt nämlich vom Rechner wie eine Zeile behandelt. Das Beschreiben kann dabei sowohl im Direkt-Modus oder vom Programm aus vorgenommen werden.

Das hat folgende Auswirkungen:

- Man kann hier 80 Zeichen auf ein Mal eingeben.

- Geht man jetzt mit RETURN über den Bildschirm, wird man feststellen, daß der Cursor hier einen Sprung über eine Bildschirmzeile macht.

Die Zeilen werden bei jedem Bildschirmlöschen wieder reorganisiert.

5. REVERS-Modus

Der Zeichengenerator des Bildschirms kann 128 verschiedene Zeichen darstellen. Jedes dieser Zeichen kann er entweder **normal** (hell auf dunkel) oder **revers** (dunkel auf hell) darstellen. Man kann aber durch PRINT nicht unmittelbar normale oder reverse Zeichen auf den Bildschirm drucken.

Vielmehr ist der Bildschirmeditor entweder im **Normal-Modus** oder im **REVERS-Modus.** Abhängig davon wird jedes Zeichen, gleich ob es durch **PRINT** gesendet wird, oder direkt von der **Tastatur** kommt, normal oder revers dargestellt.

Die Taste **RVS** ist der **Ein**schalter (RVS **ohne** SHIFT) bzw. **Aus**schalter (RVS **mit** SHIFT).

Wollen Sie in eine Stringkonstante einen String eingeben, der teilweise revers gedruckt werden soll, so erhalten Sie die folgende Darstellung der Stringkonstanten auf dem Bildschirm:

10 A$="NORMAL**R**REVERS**r**NORMAL"

Bitte beachten Sie, daß Sie Zeichen, die revers gedruckt werden sollen, nicht einfach revers in eine Stringkonstante schreiben können. Stattdessen muß vor dem Bereich, der revers gedruckt werden soll, in den String das CONTROL-Zeichen für RVS ON (ASC 18) und am Ende das für RVS OFF (ASC 146) geschrieben werden.

Drucken Sie dann A$ auf den Bildschirm, so erhalten Sie das gewünschte Ergebnis:

NORMAL**REVERS**NORMAL

6. CONTROL-Modus

6.1 Zweck des CONTROL-Modus

Der Controlmodus erlaubt die Eingabe von **Steuerzeichen.** Diese Zeichen werden auch **nichtdruckbare** Zeichen genannt, da sie auf dem Bildschirm bzw. auf Druckern nicht als Zeichen dargestellt werden, sondern irgendeine Funktion auslösen, wie z.B. Bildschirm löschen oder Fettdruck. Sie steuern oder kontrollieren also ein Ausgabegerät. Deshalb sind sie im ASCII-Code als **CONTROL**-Codes bezeichnet.

Die Bildschirmverwaltung erlaubt nun in einem besonderen Modus die Darstellung von Steuercodes als druckbare Zeichen, um die Eingabe von Steuercodes zu erleichtern.

Da durch die Vielzahl der CONTROL-Codes zwei grundsätzlich verschiedene Arten ihrer Eingabe in String-Konstante existieren, die aber letzlich denselben Effekt haben, soll zuerst erklärt werden, wie CONTROL-Codes in Stringkonstanten dargestellt werden. Vergleichen Sie dazu bitte auch das Kapitel (**Codes**) am Anfang des Handbuches, vor allem die Tabelle der ASC-Codes.

Einerseits existieren nur 128 verschiedene druckbare Zeichen, andererseits kann aber der Bildschirm durch die REVERSE-Darstellung 256 unterscheidbare Zeichen darstellen. In einer Stringkonstanten können nur die Codes der 128 nicht-reversen Zeichen dargestellt werden, sowie zusätzlich die Codes von maximal 2*32 Steuerzeichen. Diese Steuerzeichencodes werden als revers dargestellte Zeichen gezeigt.

Anders ausgedrückt, passiert bei der Übergabe einer Stringkonstanten (hinter Anführungsstrichen) durch RETURN folgendes: Alle nicht-reversen Zeichen werden in die Codes für die entsprechenden druckbaren Zeichen umgewandelt (32-95/ 160-223). Alle reversen Zeichen, die aus dem Bereich der Buchstaben (2*32 Zeichen) genommen sind, werden auf die Codes 0-31 und 128-159 abgebildet. Dagegen werden reverse Zeichen, die dem Bereich der Satzzeichen und Ziffern zugeordnet sind, einfach so umgewandelt, als wären sie gar nicht revers, sondern normal gedruckt.

Kurz zusammengefaßt gilt also: Hinter Anführungszeichen eingegebene, revers dargestellte Zeichen des Buchstabenbereichs (mit oder ohne SHIFT), werden in CONTROL-Codes umgewandelt.

Will man also Steuercodes in Stringkonstanten darstellen, muß man dafür sorgen, daß sie revers gedruckt sind. Generell genügt es, einfach durch die RVS-Taste in den REVERS-Modus zu gehen, das entsprechende Zeichen einzugeben und dann die Zeile durch RETURN zu übergeben.

Um die Eingabe von Steuercodes zu erleichtern, denen Tasten zugeordnet sind, wurde aber der CONTROL-Modus vorgesehen.

6.2 Wirkung des CONTROL-Modus

Im CONTROL-Modus haben die folgenden Tasten nicht mehr ihre normale Funktion, sondern ihr CONTROL-Zeichen wird auf den Bildschirm geschrieben:

Cursor: rechts / links / unten / oben
HOME / CLR
DEL (nur nach Einschalten durch **INST**)
INST (nur nach Einschalten durch **Anführungszeichen**)
STOP
RVS ON/OFF

6.3 Einschalten des CONTROL-Modus

Über zwei verschiedene Wege kann man in den CONTROL-Modus gelangen:

Durch Anführungszeichen

Nachdem man 1-mal Anführungszeichen auf den Bildschirm geschrieben hat, ist der CONTROL-Modus eingeschaltet, nachdem man nochmal Anführungszeichen getippt hat, ist er wieder ausgeschaltet.

Nach dem Einschalten durch Anführungszeichen wirkt **DEL** als Steuertaste, aber **INST** wird als Zeichen übernommen.

Durch Einfügen mit INST

Nachdem man durch INST einen gewissen Bereich der Zeile 'aufgeschoben' hat, ist für genau diesen Bereich der CONTROL-Modus eingeschaltet.

Nach dem Einschalten durch **INST** wirkt **INST** als Steuertaste, aber **DEL** wird als Zeichen übernommen.

6.4 Ausschalten des CONTROL-Modus

Über drei Wege kann der CONTROL-Modus ausgeschaltet werden:

Anführungszeichen

Falls der CONTROL-Modus durch Anführungszeichen eingeschaltet wurde, kann er durch ein zweites Anführungszeichenpaar ausgeschaltet werden. Wurde er aber durch INST eingeschaltet, wirken Anführungszeichen **nicht als Ausschalter.**

Verlassen des Einfügebereichs

Wurde der CONTROL-Modus durch INST eingeschaltet, so schaltet er automatisch bei Verlassen des eingefügten Bereichs ab.

RETURN

Sowohl RETURN als auch SHIFT-RETURN schalten in jedem Fall den CONTROL-Modus ab.

7. Programmeditierung

Der BASIC-Interpreter verfügt über einen **Programmeditor,** der in Zusammenarbeit mit dem **Bildschirmeditor** eine sehr komfortable Programm-Eingabe und -Änderung zuläßt.

Programme können eingegeben werden, wenn der Computer im **READY**-Modus ist, wenn also kein BASIC-Programm läuft.

Was passiert bei der Eingabe?

Jede Bildschirmzeile, die im READY-Modus durch RETURN an das Betriebssystem übergeben wird, wird zunächst in den BASIC-Eingabe-Puffer (**BEP**) kopiert. Dort wird sie vom Interpreter auf Interpreterformat umcodiert, d.h. statt der BASIC--Befehlswörter werden 1-Byte Abkürzungen eingesetzt.

Ins Programm einbauen oder sofort ausführen?

Anhand der ersten Zeichen der Zeile wird dann entschieden, ob die Zeile in den Programmspeicher übernommen, oder im **Direktmodus** ausgeführt wird: Steht am Anfang der Zeile eine **Nummer,** wird sie in den Programmspeicher übernommen, ansonsten sofort ausgeführt.

Programmzeile übernehmen

Sofern die Nummer am Anfang der Zeile eine gültige **BASIC-Zeilennummer** ist, wird der Programmspeicher durchsucht, bis die Stelle gefunden wurde, vor der diese Zeile einzufügen ist. Daraus ergibt sich eine sehr wichtige Eigenschaft des BASIC-Editors: Unabhängig von der zeitlichen Reihenfolge ihrer Eingabe werden Programmzeilen nur aufgrund der Zeilennummern **einsortiert.**

Programmzeile ersetzen

Existiert unter der angegebenen Nummer bereits eine BASIC-Zeile, so wird die alte durch die neue **ersetzt.**

Programmzeile löschen

Ist die Bildschirmzeile hinter der BASIC-Zeilennummer **leer,** so wird die entsprechende Zeile im Programmspeicher **gelöscht.** War die Zeile nicht vorhanden, hat die Eingabe keine Wirkung.

Zeilen verdoppeln / umnummerieren

Ohne weiteres kann einer BASIC-Zeile, die auf dem Bildschirm steht, eineandere Zeilennummer gegeben werden. Dadurch wird die Zeile **verdoppelt,** was bei gleich aufgebauten Zeilen (z.B. DATAs) oft Arbeit sparen kann.

Löscht man anschließend die alte Zeile, so hat man die betreffende Zeile dadurch umnummeriert.

Kleines Programm übernehmen

Da es gleichgültig ist, wie BASIC-Zeilen auf den Bildschirm kommen, ehe sie übernommen werden, kann man durch folgenden Trick eine kleine Programmroutine an ein bestehendes größeres Programm anhängen: Zuerst lädt man die kleine Routine in den Speicher und bringt sie durch **LIST** auf den Bildschirm. Dann lädt man das große Programm in den Speicher, ohne den Bildschirminhalt zu verändern. Anschließend geht man mit HOME an den Bildschirmanfang und übernimmt durch RETURN und REPEAT alle Zeilen auf dem Bildschirm.

Eine wichtige Einschränkung

Durch die Begrenzung des BEP auf 80 Zeichen wird nur der Inhalt einer einzigen Bildschirmzeile zur Auswertung übernommen. BASIC-Programmzeilen können also bei der Eingabe nicht länger als 80 Zeichen sein!

Beachten Sie in diesem Zusammenhang, daß LIST vor und nach der Zeilennummer ein SPACE ausdruckt. Da dies bei der Eingabe der Zeile nicht erforderlich ist, kann es passieren, daß eine Zeile, die bei der Eingabe gerade 80 Zeichen lang war, nach LIST in die nächste Zeile überlappt. Wenn diese Zeile dann ohne Änderung durch RETURN übernommen wird, fehlen ihr die **letzten** beiden Zeichen.

Die gleiche Problematik tritt auf, wenn Sie bei der Eingabe die Abkürzungen der BASIC-Anweisungen verwenden, da LIST die BASIC-Befehle in jedem Fall ausschreibt.

8. Tastaturpuffer

8.1 Einführung

Völlig unabhängig vom Zustand, in dem er sich gerade befindet, tastet der Computer 60 mal pro Sekunde seine Tastatur ab, und übernimmt den Code einer evtl. gedrückten Taste in den **Tastaturpuffer.** Bei der Einführung zur **Bild**schirmverwaltung / Tastaturbehandlung ist beschrieben, wie der Inhalt dieses Puffers in Abhängigkeit von verschiedenen Betriebszuständen behandelt wird.

Hier soll nur erklärt werden, welche Eigenschaften der Puffer hat und wie er manipuliert werden kann, um spezielle Effekte zu erreichen.

8.2 Eigenschaften

Der Tastaturpuffer kann normalerweise 9 Zeichen fassen. Bei jedem 10. Zeichen wird er wieder geleert wobei die bisher gedrückten Tasten verlorengehen. Da der Computer kein Signal gibt, wenn der Puffer voll ist, kann es bei langsamen BASIC-Programmen vorkommen, daß oft Tasten umsonst gedrückt worden sind, weil der Puffer übergelaufen ist.

8.3 Simulierte Tastatur

In einigen Fällen ist es wünschenswert, daß man dem Computer vom BASIC-Programm aus Befehle geben kann, die er nach Ende des Programms im Direktmodus ausführen kann. Hier soll nur die prinzipielle Vorgehensweise erklärt werden:

Sobald der Rechner in den Eingabe-Modus kommt, holt er alle Zeichen aus dem Tastaturpuffer, die sich bis dann angesammelt haben. Insbesondere kann er auch CR (Carriage Return) aus dem Puffer holen, und auf den Bildschirm bringen. CR bewirkt dann, daß die Zeile, in der der Cursor gerade steht, an das Betriebssystem zur Auswertung übergeben wird. Dies bewirkt wiederum, daß Befehle, die in dieser Zeile stehen, ausgeführt werden. Wenn nach Ausführung dieser Befehle noch weitere CRs im Puffer stehen, wird das nächste geholt und die nächste Cursor-Zeile ausgeführt.

Dadurch kann man durch geschickte Kombination von Zeilen, die vorher durch PRINT auf den Bildschirm gedruckt wurden und die entsprechenden CRs im Puffer ziemlich umfangreiche Befehlssequenzen im Direktmodus automatisch ablaufen lassen. Bei dieser Art der simulierten Tastatur wird also der eigentliche Text vor Programmende auf den Bildschirm gedruckt und nur die CRs werden in den Tastaturpuffer geschrieben.

Der Tastaturpuffer beginnt bei Zelle **623.** In **158** steht die Anzahl der gültigen Zeichen im Puffer. Wollen Sie also z.B. drei CRs in den Puffer bringen, so sind folgende Anweisungen nötig:

```
10 POKE 158 , 3    3 Zeichen sind im Puffer
20 POKE 623 , 13   1. CR
30 POKE 624 , 13   2. CR
40 POKE 625 , 13   3. CR
50 END             beendet das Programm und bewirkt Ausführung der drei
                   Zeilen
```

9. Anwählen von Zeile und Spalte

Die Bildschirmverwaltung unterstützt nicht unmittelbar die Positionierung des Cursors in einer bestimmten Zeile und Spalte. Durch zwei verschiedene Routinen, die hier vorgestellt werden sollen, kann aber eine schnelle Positionierung des Cursors erreicht werden.

9.1 String-Methode

Die String-Methode geht davon aus, daß beim Start des Programms zwei Strings mit **CURSOR NACH RECHTS** und **CURSOR NACH UNTEN** CONTROL-Codes gefüllt wurden. Dies kann mit den folgenden zwei Zeilen erreicht werden:

```
150 CU$="U":CR$="R":FOR I=1 TO 7:CU$=CU$+CU$:CR$=CR$+CR$:NEXT
151 CU$=LEFT$(CU$,25):CR$=LEFT$(CR$,40)
```

Die folgende Zeile bringt dann den Cursor in die entsprechende Zeile (CZ) und Spalte (CS), wobei Zeilen und Spalten ab 0 gezählt werden.

```
100 PRINT"H"LEFT$(CU$,CZ)LEFT$(CR$,CS);:RETURN
```

Diese Anweisung bringt den Cursor also erst durch HOME in die linke obere Ecke des Fensters und geht dann CZ-mal nach unten und CS-mal nach rechts.

Die Zeile 100 benötigt im Mittel etwa 20 ms, um den Cursor an die richtige Stelle zu bringen.

9.2 POKE-Methode

Das Betriebssystem verwaltet die aktuelle Position in zwei Zellen, nämlich die **Zeile** in **216** und die **Spalte** in **198.** Durch POKE kann Zeile und Spalte unmittelbar angegeben werden. Allerdings muß nach dem POKE der Zeile ein CR gedruckt werden, weil die Zelle **216** icht unmittelbar vom Betriebssystem gelesen wird, sondern erst beim Zeilenwechsel in zwei weitere Zellen die absolute Startadresse der Bildschirmzeile geschrieben werden. Die Berechnung dieser Werte in BASIC ist aber wesentlich zeitaufwendiger, als die Methode, ein CR zu drucken.

Die Routine sieht folgendermaßen aus:

```
200 IF CZ=0 THEN ?"H"; : GOTO 220
210 POKE 216,CZ : PRINT "O"
220 POKE 198,CS : RETURN
```

Die PRINT-Anweisung in 210 geht zuerst mit dem Cursor um eine Zeile nach oben, um das folgende CR zu kompensieren. In der 1. Bildschirmzeile funktioniert dies aber nicht, deshalb ist die Abfrage in 200 erforderlich.

Die Ausführungszeit liegt bei ca. 15 ms.

10. Bildschirmbehandlung mit POKE und PEEK

Für fortgeschrittene Programmierer kann es bei bestimmten Problemstellungen sinnvoll sein, ohne die Unterstützung von PRINT oder INPUT bzw. GET direkt auf den Bildschirm zuzugreifen. Deshalb soll hier kurz erklärt werden, warum und wie das vor sich gehen kann.

Der Bild-Wiederhol-Speicher, oft auch Bildschirmspeicher genannt, unterscheidet sich nicht vom sonstigen Schreib- / Lesespeicher (RAM). Die besondere Eigenschaft, daß sein Inhalt auf dem Bildschirm als Zeichen dargestellt wird, erhält er durch eine vom Prozessor unabhängige Verwaltung, die 60 mal pro Sekunde den Bild-Wiederhol-Speicher-Inhalt an die Video-Hardware übergibt.

Auf den Bildschirmspeicher kann man deshalb beliebig schreibend und lesend zugreifen. Für BASIC bedeutet dies, daß man mit **POKE** Zeichen auf den Bildschirm schreiben und mit **PEEK** Zeichen aus dem Bildschirm lesen kann.

Die Adressen des sichtbaren Bildschirmbereichs liegen zwischen **32768** und **33767** ($8000-$83E7). Der Bildschirmspeicher ist eigentlich genau 1K groß, aber nur 1000 Zeichen werden sichtbar gemacht. Die restlichen 24 Zeichen von 33768 bis 33791 ($83E8-$83FF) können aber trotzdem als Zwischenspeicher benützt werden. Beachten Sie aber, daß bei jedem Bildschirmrollen und beim Löschen des Bildschirms auch die Information der nicht sichtbaren 24 Bytes verändert wird.

Die folgende Schleife soll demonstrieren, wie mit POKE der ganze Zeichensatz auf den Bildschirm gebracht werden kann:

```
FOR I=0 TO 255 : POKE I+32768,I : NEXT
```

Mit POKE die Anweisung PRINT ersetzen zu wollen, ist nicht sinnvoll, da man dann die Wandlung von ASC in BSC (s. **Codes**) selbst durchführen muß. Allerdings kann es sinnvoll sein, an bestimmte Stellen des Bildschirms durch POKE einzelne Zeichen zu bringen, die z.B. als Anzeige für bestimmte Zustände eines Programms dienen können.

Entsprechend kann durch PEEK die Information einer bestimmten Bildschirmzelle einfacher abgefragt werden, als z.B. durch **GET**.

Wollen Sie z.B. wissen, welches Zeichen in der oberen linken Bildschirmecke steht, erfahren Sie dies durch die Anweisung:

```
10 ZE=PEEK(32768) : IF ZE=1 THEN PRINT "A"
```

Hiermit soll gleichzeitig darauf hingewiesen werden, daß der Code 1 auf dem Bildschirm als "A" dargestellt wird. Vergleichen Sie dazu die Tabellen des Kapitels **Codes**.

Um Mißverständnissen vorzubeugen, sei klar gesagt, daß POKE und PEEK im Gegensatz zu PRINT, INPUT und GET die Cursorverwaltung des Bildschirmeditors nicht beeinflussen.

IV. OVERLAY

1. Einführung

Mit 32 K Arbeitsspeicher läßt sich zwar einiges anfangen, aber bei größeren Problemlösungen mit umfangreichen Datenbeständen ist es in der Regel nicht mehr möglich, das ganze Programm gleichzeitig im Speicher zu halten. Man muß dann dazu übergehen, das Programm in mehrere Abschnitte zu unterteilen, von denen jeder unabhängig vom anderen eine Teilaufgabe bearbeiten kann. Ist dann die erste Aufgabe beendet, kann der nächste Abschnitt geladen werden usw.

Diese Art, große Programme in Teilstücken zeitlich nacheinander in den gleichen Arbeitsspeicherbereich zu laden, wird **Overlay** genannt, was **Überlagerung** bedeutet. Grundsätzlich sind mit dem derzeitigen Betriebssystem zwei verschiedene Varianten möglich:

Entweder sind nach dem Laden des nächsten Programmabschnitts alle Variablen gelöscht, oder sie sind noch alle vorhanden (und haben auch ihre Inhalte nicht verloren).

Je nach Problemstellung können beide Möglichkeiten sinnvoll sein, sie können sogar gemischt verwendet werden.

Wenn erstens die einzelnen Programmodule etwa gleich groß sind und zweitens die meisten verwendeten Variablen, vor allem die Felder von allen Modulen gebraucht werden, ist es sinnvoll, die Variablen zu erhalten. Man spricht dann von einem sogenannten **Warm**start.

Wenn dagegen entweder die Programmteile sehr unterschiedliche Größen aufweisen, oder sich die Datenstruktur im Arbeitsspeicher ständig ändert, also die Art, Anzahl und Größe der benötigten Felder von Modul zu Modul stark schwankt, wird man den Weg des **Kalt**starts gehen müssen.

Die Gründe für diese Unterscheidung sollen kurz erläutert werden:

Durch das Betriebssytem bedingt, muß bei einer Sequenz von warm gestarteten Modulen beim Start des ersten Moduls mindestens soviel Programmspeicher reserviert werden, wie vom größten Modul benötigt wird. Damit ist aber die Aufteilung des gesamten Arbeitsspeichers in Programm- und Datenspeicher festgelegt. Daraus folgt, daß bei unterschiedlich großen Programmodulen Datenspeicherplatz verschenkt wird.

Eine überschlägige Rechnung soll dies verdeutlichen: Das größte Programm habe einen Platzbedarf von 20K, weswegen für alle Programme der Overlay-Sequenz diese 20K reserviert werden müssen. Geht man von ca. 30K freiem Arbeitspeicher aus, bleiben etwa 10K Arbeitsbereich für die Daten aller Programme.

Ein Programmodul aus der Overlay-Sequenz sei nur 5K groß. Würde er für sich alleine laufen, würden für die Daten 25K zur Verfügung stehen. In der Overlay--Sequenz bleiben aber auch für den 5K-Programmblock nur 10K Datenbereich frei. Das bedeutet, daß 15K Speicherplatz ungenützt bleiben müssen.

Nun könnte man meinen, daß man nur den Programmteil, der 20K belegt, in mehrere kleinere Blöcke zu zerlegen braucht, um das Problem zu lösen. Es kann aber Gründe geben, die es verbieten, einen Programmblock zu zerlegen: Jedes Nachladen von Programmteilen benötigt zwischen 5 und 10 Sekunden für Modulgrößen um die 10K. Wenn man nun zwei Module hat, die zwar logisch getrennt sind, die sich aber gegenseitig z.B. jede Sekunde einmal aufrufen, ist es nicht vertretbar, sie zu trennen, da dann auf eine Sekunde Programmlaufzeit 10 Sekunden Ladezeit kommen würden.

Aus Laufzeitgründen kann man also auch bei modularer Programmierweise sehr unterschiedlich große Module bekommen. Natürlich kann man auf die Laufzeit nicht beliebig viel Rücksicht nehmen, da spätestens bei 30K Programmgröße längere Laufzeiten durch die Aufspaltung des Programms in Kauf genommen werden müssen.

Nun aber zurück zu dem Problem, daß der Warmstart eventuell mit dem Arbeitsspeicher zu uneffektiv umgeht. Beim Kaltstart wird nur exakt soviel Platz für das Programm belegt, wie es wirklich benötigt, so daß maximal viel Platz für die Daten zur Verfügung steht. Da aber beim Kaltstart alle Variablen gelöscht werden, setzt diese Methode voraus, daß alle benötigten Daten auf Floppy zwischengespeichert werden. Nach dem Start des nächsten Moduls werden diese Daten dann vom Floppy in den Arbeitsspeicher gelesen.

Nun ist vorher noch ein zweiter Grund für die Kaltstart-Methode angeführt worden, nämlich stark unterschiedliche Datenstrukturen. Das Betriebssystem kann zwar jederzeit neue Felder einführen, kann aber nicht einzelne Felder löschen. Es kann aber ohne weiteres vorkommen, daß man am Anfang unbedingt ein dreidimensionales Gleitkommafeld mit einem Platzbedarf von 10K benötigt, das später nie mehr benötigt wird. In diesem Fall gibt es keine andere Möglichkeit, als den nächsten Teil kalt zu starten.

Nach diesen Erläuterungen, warum man welche Methode benötigt, sollen nun die einzelnen Methoden vorgestellt werden. Sie werden sehen, daß beide Möglichkeiten, die Starts durchzuführen, Varianten des LOAD-Befehls sind, wobei die Wirkung der LOAD-Anweisung selbst identisch ist:

2. Wirkung von LOAD

LOAD kann entweder innerhalb eines Programms angegeben werden, oder im Direktmodus. In jedem Fall wird das angegebene Programm in den Arbeitsspeicher geladen. Beachten Sie aber die folgenden beiden Unterschiede:

Nach LOAD im **Direktmodus** wird der Zeiger auf das Programmende / Beginn der Variablen an die aktuelle Programmgröße angepaßt und das Programm wird nicht gestartet.

Nach LOAD im **Programm** läuft das neu geladene Programm sofort mit der **ersten** Programmzeile an und der Zeiger auf das Programmende / Beginn der Variablen wird nicht verändert.

LOAD im Programm bietet also automatisch die Möglichkeit des Warmstarts. Dagegen muß der Kaltstart in jedem Fall mit einem Kunstgriff gemacht werden. Eine Möglichkeit wäre die **simulierte Tastatur.** Man würde also LOAD ... und RUN in die entsprechenden Bildschirmzeilen schreiben, die entsprechenden CRs in den Tastaturpuffer schreiben und auf ein END laufen. Dadurch würde im Direktmodus der Kaltstart durchgeführt. Sie werden aber sehen, daß es eine elegantere Möglichkeit gibt, weswegen diese Anwendung der simulierten Tastatur nicht erklärt wird.

Eine Eigenschaft von LOAD **im Programm** soll nochmal deutlich herausgestellt werden: Die LOAD-Anweisung im Programm ist die **letzte Anweisung,** die in diesem Programm ausgeführt wird. Es hat also keinen Sinn, hinter dem LOAD noch ein GOTO ... anzugeben, in der Meinung, das nachgeladene Programm würde dann in dieser Zeile gestartet. Vielmehr wird das nachgeladene Programm **immer in der** ersten Zeile gestartet. Falls in diesem Programm abhängig von der Vorgeschichte zu verschiedenen Punkten verzweigt werden muß, kann am Anfang des Programms ein Sprungverteiler stehen.

3. Kaltstart

Nach dem Laden eines Programmes steht immer in den Zellen 201/202 die Endadresse + 1 des eben geladenen Programmes. Bei LOAD im Direktmodus wird der Inhalt dieser Zellen in 42/43 kopiert und CLR ausgeführt. Dadurch wird der Variablenbereich unmittelbar hinter dem Programm angelegt.

Wenn man nun in die erste Programmzeile eines Programmodules folgende Anweisungen schreibt, erreicht man Kaltstart für dieses Modul.

```
1  IF PEEK(42) = PEEK(44) AND PEEK(43) = PEEK(45) GOTO 3
2  POKE 42 , PEEK (201) : POKE 43 , PEEK (202) : CLR
3  REM PROGRAMMBEGINN
```

Start nach Änderungen:

Die Zellen (201/202) werden nicht geändert, wenn am Programm im Speicher Änderungen durchgeführt werden. Die Abfragen in Zeile 1 verhindern, daß die Programmlänge nach einer Änderung auf den alten Wert gesetzt wird. Dadurch würde entweder der hintere Teil des Programms zerstört, falls es größer geworden ist, oder das Programm würde 'aufgeblasen', falls es kleiner geworden ist.

Vorher wurde schon angedeutet, daß eventuell am Anfang eines Moduls ein Sprungverteiler nötig ist, um abhängig von z.B. verschiedenen aufrufenden Programmen verschiedene Startpunkte anzuspringen. Da wegen des Kaltstarts eine Variable als Träger der Information ausscheidet, kann man die Nummer für den Sprungverteiler z.B. in den BEP schreiben. Im aufrufenden Programm würde dann z.B. stehen:

```
135  POKE 512,SN : LOAD"PROG-2",8  (SN = Sprung-Nummer)
```

Am Anfang von PROG-2 würde dann entsprechend stehen:

```
1  IF PEEK(42) = PEEK(44) AND PEEK(43) = PEEK(45) THEN STOP
2  POKE 42 , PEEK (201) : POKE 43 , PEEK (202) : CLR
3  ON PEEK(512) GOTO 100, 200, 300, ....
```

Ein Start mit RUN über die Zeile 1 hat für diese Version keinen Sinn, deshalb das STOP in Zeile 1. Für Testläufe müßte man hier nämlich direkt die richtige Zeile (100, 200, 300 ..) aufrufen.

4. Warmstart

Wie schon erwähnt, wird beim Warmstart an der Aufteilung zwischen Programmspeicher und Variablenspeicher nichts mehr geändert. Deshalb ist absolute Voraussetzung für Warmstart, daß der reservierte Programm-Speicher-Bereich mindestens so groß ist, wie das größte zu ladende Teilprogramm. Die zweite Voraussetzung ist, daß Strings, die in folgenden Programmteilen gebraucht werden, nicht unmittelbar aus Stringkonstanten ihren Inhalt bezogen haben.

4.1 Programmspeicher reservieren

Weder das erste Programm, noch das Betriebssystem kann 'wissen', wie groß das größte der nachzuladenden Programme ist. Deshalb müssen Sie dies feststellen und dann im Startprogramm mit Hilfe dieser Information genügend Programm-Speicher reservieren.

Das größte Programm finden Sie folgendermaßen: Laden Sie nacheinander alle Programme des Overlay-Pakets in den Speicher und fragen Sie jedesmal durch **?FRE(0)** wieviel Speicher noch frei ist. Das Programm, bei dem am wenigsten Speicher frei ist, ist das größte. Bei diesem Programm fragen Sie nun durch

```
?Pe(42)pE(43)
 wert1 wert2
```

den genauen Inhalt des Programm-Ende-Zeigers ab. Die beiden Werte, die daraufhin auf den Bildschirm geschrieben werden, merken Sie sich.

Beim ersten Programm (Startprogramm) schreiben Sie in die erste Programmzeile folgende Anweisungen:

```
1 POKE 42 , wert1 : POKE 43 , wert2 : CLR
```

(wert1 und wert2 stehen stellvertretend für die beiden Zahlenwerte)

Diese Anweisung bewirkt, daß die Programm-Speicher-Grenze auf den Wert gesetzt wird, den das größte Programm benötigt.

Das Programm speichern Sie nun ab, **ohne es gestartet zu haben.** Würden Sie es nämlich starten, könnten Sie es nur noch im 'aufgeblasenen' Zustand abspeichern.

Beachten Sie außerdem folgende Regel:

Durch Overlay geladene Programme dürfen nie abgespeichert werden!

Würden Sie nämlich ein durch Overlay geladenes Programm abspeichern, würde dabei der ganze Speicherbereich bis zum Programm-Ende-Zeiger abgespeichert, was ja in der Regel nicht die wahre Programmgröße ist, sondern mehr. Abgesehen davon, daß dies Speicherplatzverschwendung auf der Floppy und Ladezeiterhöhung bedeuten würde, würde z.B. das kleinste Programm nach Hinzufügen von nur einem Byte größer als das (bisher) größte Programm werden und damit die Overlay-Sequenz zu Fall bringen.

Nun ist man aber beim Testen von Overlay-Programmen der Versuchung aussetzt, einen aufgetretenen Fehler sofort zu korrigieren. In diesem Fall müssen Sie aber **unbedingt** das entsprechende Programm nochmal 'von Hand' in den Speicher laden.

Allgemein empfehlen wir, sehr sorgfältig darauf zu achten, daß die Reservierung im Startprogramm wirklich nach jeder Änderung dem größten Programm entspricht. Beachten Sie dabei, daß sowohl das größte Programm durch eine Änderung größer werden kann, als daß auch ein kleineres Programm nach einer Änderung das neue größte Programm sein kann.

Die Folgen einer falschen Reservierung sind sehr unangenehm: Die Variablentabelle wird ganz oder teilweise durch das zu große Programm überschrieben, wird aber vom Variablensucher behandelt, als wäre alles in Ordnung. Die Folge sind exotische Variablennamen mit unsinnigen Inhalten. In der Regel führt dies relativ bald zu einem Programmfehler, dessen Ursache aber schwer zu finden ist, wenn man nicht sofort erkennt, daß er durch falsches Overlay entstanden ist.

Während der Entwicklungsphase von Programmen können Sie aber mehr Platz reservieren, als das bisher größte Programm eigentlich benötigt. Dadurch müssen Sie nicht nach jeder Änderung am größten Programm auch die erste Zeile des Startprogramms ändern.

'wert2', also der Inhalt von Zelle 43 gibt die 256 Byte-'Page' (Seite) an, in der das Programm endet. Sie können diesen Wert ohne weiteres vergrößern, jede Einheit mehr entspricht 256 Bytes mehr Programm-Speicherplatz. Allerdings wird dadurch der Variablen-Bereich um 256 Bytes kleiner, so daß Sie die Erhöhung nicht übertreiben dürfen.

4.2 Overlay und Strings

Um den Zusammenhang zwischen Overlay und Strings erklären zu können, ist ein kleiner Abstecher zur Stringverwaltung angebracht:

Stringverwaltung

Die Namen von Stringvariablen stehen in der allgemeinen Variablentabelle, wo auch die Gleitkomma- und Integervariablen stehen. Jede Variable belegt in dieser Tabelle 7 Bytes.

Während aber bei den Zahlenvariablen in diesen 7 Bytes außer dem Namen auch gleich der Inhalt gespeichert ist, kann dies bei Stringvariablen nicht der Fall sein, weil sie ja bis zu 255 Bytes fassen können.

Also steht beim String-Namen nur eine Beschreibung, wo der String abgelegt ist. Die Beschreibung besteht aus der Länge des Strings und einem Zeiger auf seinen Beginn.

In der Regel steht der Stringinhalt im Stringspeicherbereich zwischen dem Ende der Variablentabellen und dem oberen BASIC-Speicher-Ende. Zusätzlich wurde aber eine Speicherplatzoptimierung vorgenommen, die bewirkt, daß Zuweisungen aus Konstanten gesondert behandelt werden:

```
10 A$ = "ABCD"
```

Nach dieser Anweisung steht im Beschreiber von A$ ein Zeiger, der unmittelbar auf die Stelle des Programms zeigt, wo ABCD steht. Dadurch muß dieser String nicht in den Stringspeicherbereich kopiert werden, was ja gleichbedeutend damit wäre, daß er zweimal existiert, und damit Platzverschwendung wäre.

Gleiches gilt für Stringzuweisungen aus **DATA** durch **READ**.

Diese Methode der Zuweisung von Konstanten spart zusätzliche 2 Bytes pro String ein, die bei Strings im Stringspeicherbereich benötigt werden, um vom String aus einen Zeiger auf den Beschreiber zu haben. Dieser Zeiger ist nötig, um die dynamische Stringspeicherverwaltung zeitlich optimal ablaufen zu lassen.

Bemerkenswert ist noch, daß nach den Zuweisungen

```
10 A$="ABCD" : B$=A$
```

die Beschreiber beider Variablen auf denselben Ort zeigen, während dies nach

```
20 A$="ABCD"+"" : B$=A$
```

nicht der Fall ist.

Konsequenzen der Stringverwaltung für Overlay

Bei Overlay hat diese an sich positive Verwaltung von Stringkonstanten einen Fehler - sie funktioniert nicht. Der Grund ist einsichtig:

Die Zeiger von einigen Stringvariablen zeigen ins Programm. Das Programm wird dann gegen ein anderes ausgetauscht. Natürlich stehen an der Stelle, wo vorher die Stringkonstante stand, irgendwelche Anweisungen, die als Stringinhalt Unsinn ergeben.

Also muß man den Stringverwalter zwingen, auch Konstante in den Stringbereich zu übertragen, was nicht schwer ist. Findet der Verwalter nämlich nicht eine einfache Zuweisung, sondern eine Stringverknüpfung vor, so kopiert er den Ergebnisstring immer in den Stringbereich. Durch

```
10 A$="ABCD"+""
```

kann man ihn also 'überlisten'. Bei READ müßte dies etwa so aussehen:

```
20 READ A$ : A$=A$+""
```

4.3 Overlay und DEF FN( )

Ähnlich wie bei Strings wird bei DEF FN nur ein Zeiger auf die Programmstelle angelegt, an der die Funktion im Programm steht. Nach dem Nachladen eines anderen Programmes steht an der Stelle, wo nach dem Zeiger eigentlich die Funktionsdefinition stehen sollte, irgendein Programmteil. Da der Interpreter einfach ab der Zeigerstelle versucht, den (normalerweise) dort beginnenden numerischen Ausdruck abzuarbeiten, erkennt er irgendeinen Fehler und bricht die Bearbeitung ab. In der Regel dürfte ein SYNTAX ERROR oder ein ILLEGAL QUANTITY ERROR gemeldet werden, aber auch andere Meldungen sind denkbar.

Beachten Sie in diesem Zusammenhang, daß bei diesen Fehlermeldungen die Zeile des Funktionsaufrufes angegeben wird, die gar keinen Fehler enthält (s. FN).

Da es nicht wie bei Strings eine Möglichkeit gibt, die Funktionsdefinition anderweitig als im Programm abzuspeichern, hilft bei Overlay nur Neudefinition: In jedem Programmodul, der eine bestimmte Funktion benötigt, muß sie durch DEF FN definiert werden!

4.4 Overlay und Floppy-Dateien

In der Regel bleiben Floppy-Dateien offen, wenn durch Overlay ein neues Programm nachgeladen wird. Zwei Grenzen sind aber zu beachten:

LOAD öffnet eine PRG-Datei, die genauso wie jede andere Datei zwei Puffer im Floppy benötigt. Deshalb muß darauf geachtet werden, daß nicht durch diese Datei die maximal zulässige Anzahl überschritten wird.

Zum Zweiten wird eine evt. auf die Sekundäradresse 0 geöffnete Datei durch LOAD geschlossen, da LOAD diesen Kanal verwendet!

4.5 Overlay und Stack

Durch LOAD wird der Stack geleert. Das bedeutet, daß es keinen Sinn hat, LOAD innerhalb eines Unterprogramms oder einer Schleife zu haben beide Strukturen werden hinterher ignoriert.

In diesem Zusammenhang soll auch erwähnt werden, daß der DATA-Zeiger rückgesetzt wird.

Diese beiden Anmerkungen sind eigentlich überflüssig, wenn BASIC-Programme nachgeladen werden. Wenn man aber Maschinenprogramme lädt, könnte man durchaus auf die Idee kommen, die Namen aus DATAs zu lesen und in einer Schleife durch LOAD zu laden. Dies würde aber gleich aus zwei (oben erwähnten) Gründen nicht funktionieren.

V. REKORDERVERWALTUNG

1. Einführung

Der Rekorder ist ein sehr billiger Massenspeicher, der dementsprechend weniger leistet, als Commodore Floppy-Disk-Laufwerke. Trotzdem hat der Rekorder z.B. in Schulungsanwendungen seine Berechtigung.

Der Hauptunterschied zwischen Floppy und Kassette liegt darin, daß bei der Kassette alle Dateien hintereinander aufgezeichnet werden und auch nur nacheinander gelesen werden können. Man kann also nicht beim Rekorder eine bestimmte Datei verlangen. Vielmehr können ohne manuellen Eingriff nur Dateien gefunden werden, die hinter der derzeitigen Position des Bandes beginnen und auch nur dadurch, daß alle anderen Dateien überlesen werden. Im schlimmsten Fall kann es deshalb 30 Minuten dauern, bis die letzte Datei eines Bandes gefunden wurde.

Abgesehen von diesen zeitlichen Unterschieden werden aber Banddateien von BASIC genauso behandelt, wie serielle (SEQ) Floppy-Dateien.

Insbesondere sind die BASIC-Anweisungen für Band-Dateien identisch mit denen für Floppy-Dateien. Programme werden auch hier durch LOAD, SAVE und VERIFY behandelt. Entsprechend werden allgemeine Dateien durch OPEN und CLOSE, sowie PRINT§, INPUT§ und GET§ behandelt. Die grundsätzlichen Eigenschaften dieser Anweisungen gelten also auch bei Band-Dateien.

2. Der Rekorder

Die 'Datasette' ist speziell an den Computerbetrieb angepaßt, indem sie über einen Kontakt dem Computer melden kann, ob eine der Tasten PLAY / REW / FFWD gedrückt ist. Der Computer kann daraufhin die Spannungsversorgung für den Motor ein- und ausschalten.

Er kann aber nicht selbst die Tasten betätigen und auch nicht feststellen, welche der drei erwähnten Tasten gedrückt wurde. Ebenso kann er nicht feststellen, ob die RECORD-Taste gedrückt ist, oder nicht!

Vor allem diese letzte Eigenschaft sollten Sie beachten, da Sie dadurch durch falschen Tastendruck ohne weiteres Dateien löschen können, ohne daß der Computer Sie davor bewahren könnte. Die Elektronik des Rekorders schaltet nämlich unabhängig vom Computer auf Aufnahme und damit den Löschkopf ein, sobald REC und PLAY gedrückt ist.

Um solche unbeabsichtigten Datenlöschungen zu verhindern, können Sie den kleinen Zacken am linken hinteren Ende der Kassette rausbrechen. Dadurch läßt sich die Aufnahme-Taste (REC) nicht mehr drücken. Sollten Sie dies dann trotzdem noch wünschen, schieben Sie bei leerem Kassettenfach den Fühler links hinten nach hinten und drücken die REC-Taste. Die Taste halten Sie dann gedrückt, legen die Kassette ein, schließen das Fach und drücken PLAY.

3. Dateinamen

Der Dateinamen bei Band-Dateien ist eine beliebige Zeichenkombination mit einer Länge von maximal 128 Zeichen. Allerdings werden nur 16 Zeichen vom Betriebssystem gemeldet. Trotzdem werden aber 128 Zeichen ausgewertet und können mit Hilfe eigener Programmergänzungen sichtbar gemacht werden.

Beim lesenden Zugriff auf Dateien wird jeder gefundene Namen mit dem gesuchten verglichen. Sobald der linke Teil des gefundenen Namens mit dem ganzen gesuchten Namen identisch ist, wird er als richtig akzeptiert und dieses Programm wird geladen, bzw. diese Datei wird geöffnet.

4. EOT (END OF TAPE = Bandende)

Die Rekorderverwaltung des Betriebssystems weist eine Besonderheit auf, die bei anderen Geräten in dieser Form nicht zu finden ist. Beim Schreiben von Dateien oder Programmen kann am Ende ein EOT-Block angehängt werden. Dieser Block sagt dem Betriebssystem beim Lesen, daß das Band hier zu Ende ist.

Dazu ist anzumerken, daß die Mechanik des Kassettenlaufwerks zwar am mechanischen Bandende die PLAY-Taste auswirft, so daß das Band steht, daß das Betriebssystem aber daraus nicht schließt, daß das Band zuende ist. Vielmehr fordert es in diesem Fall nochmal zum Drücken der Tasten PLAY (und RECORD) auf.

Außerdem steht in der Regel die letzte Datei eines Bandes nicht bis unmittelbar zum Bandende. Haben Sie z.B. nur eine Datei auf dem Band, haben aber vor dem LOAD bzw. OPEN vergessen, es zurückzuspulen, so kann diese Datei überhaupt nicht mehr gefunden werden. Abgesehen davon, daß Sie vielleicht einige Minuten brauchen, um den Fehler zu bemerken, kann sich das Betriebssystem 'aufhängen', wenn es undefiniertes Zeug vom Band lesen muß.

Haben Sie aber am Ende der letzten Datei einen EOT-Block geschrieben, so bringt das Betriebssystem die Fehlermeldung **FILE NOT FOUND ERROR** und stoppt Band und Programm.

Der EOT-Block wird geschrieben, wenn die Sekundäradresse 2 ist (dritter Parameter bei SAVE und OPEN).

5. PRESS PLAY AND RECORD

Diese Meldung will Sie veranlassen, den Rekorder auf Aufnahme zu schalten. Entgegen der Reihenfolge in der Meldung müssen Sie **zuerst die RECORD-Taste** drücken und festhalten und erst dann die PLAY-Taste drücken.

6. SAVE

Beim Rekorder gibt es SAVE in folgenden Formen:

SAVE	ohne Namen auf Rekorder 1
SAVE dateiname	mit Namen auf Rekorder 1
SAVE dateiname , 2	mit Namen auf Rekorder 2
SAVE dateiname , 1 , 2	mit Namen und EOT auf Rekorder 1

Die allgemeine Form lautet:

SAVE dateiname , rekordernummer , sekundäradresse

Der **Dateiname** kann auch leer sein, dies ist aber nicht zu empfehlen.
Die **Rekordernummer** kann 1 oder 2 sein, 1 ist Ersatzwert.
Die **Sekundäradresse** kann 1 oder 2 sein, 1 ist Ersatzwert.

7. LOAD / VERIFY

Beim Rekorder gibt es LOAD in folgenden Formen:

LOAD	ohne Namen von 1 (nächstes Programm)
LOAD dateiname	mit Namen von Rekorder 1
LOAD dateiname , 2	mit Namen von Rekorder 2

Die allgemeine Form lautet:

LOAD dateiname , rekordernummer , sekundäradresse

Der **Dateiname** kann auch leer sein.
Die **Rekordernummer** kann 1 oder 2 sein, 1 ist Ersatzwert.

VERIFY ist in der Form **identisch** mit **LOAD.**

8. Dateibehandlung

8.1 OPEN

Bei OPEN ist nur die Sekundäradresse (sa) spezifisch für die Rekorder. Die Werte und ihre Bedeutungen sind:

0	Datei **lesen**
1	Datei **schreiben**
2	Datei schreiben mit **EOT**

8.2 Aus Datei lesen

Nach OPEN mit sa=0 meldet das Betriebssystem **PRESS PLAY ON TAPE** §1 (bzw. §2), sofern noch nicht PLAY gedrückt ist. Dann sucht es die Datei mit dem angegebenen Namen. Da das OPEN in der Regel im Programm ausgeführt wird, erfolgt keine weitere Meldung auf dem Bildschirm. Nach der Regel, die bei **Dateinamen** angegeben ist, wird dann diejenige Datei gesucht, deren Namen dem gesuchten entspricht. Die OPEN-Anweisung gibt die Kontrolle erst dann an BASIC zurück, wenn die richtige Datei gefunden wurde.

Nachdem die OPEN-Anweisung ordnungsgemäß ausgeführt wurde, kann durch **INPUT§** oder **GET§** aus der Datei gelesen werden. Beachten Sie dazu unbedingt das Kapitel **Status.** Beim Rekorder ist jeder Statuswert außer 0 und 64 ein **Fehler!**

Versuchen Sie, auf eine zum Lesen geöffnete Datei durch **PRINT§** zuzugreifen, so wird **NOT OUTPUT FILE ERROR IN ...** (Keine Ausgabe-Datei) gemeldet. (Diese Meldung wird nur bei sa=0 abgesetzt.)

8.3 In Datei schreiben

Nach OPEN mit sa=1 (oder 2) meldet das Betriebssystem **PRESS PLAY AND RECORD ..,** sofern noch nicht PLAY gedrückt ist. Dann schreibt es den Datei-Namen auf Band und gibt die Kontrolle an BASIC zurück (nach ca. 14 Sekunden).

Nachdem die OPEN-Anweisung ausgeführt wurde, kann durch PRINT§ in die Datei geschrieben werden.

Versuchen Sie, auf eine zum Schreiben geöffnete Datei durch **INPUT§** oder **GET§** zuzugreifen, so wird **NOT INPUT FILE ERROR IN ..** gemeldet (keine Eingabedatei). (Diese Meldung wird nur bei sa=1 oder 2 abgesetzt).

Versuchen Sie, auf eine zum Schreiben geöffnete Datei durch **INPUT§** oder **GET§** zuzugreifen, so wird **NOT INPUT FILE ERROR IN ..** gemeldet (keine Eingabedatei). (Diese Meldung wird nur bei sa=1 oder 2 abgesetzt).

8.4 Datenübertragungsrate und Datenkapazität

Die Datenübertragungsrate liegt bei ca. 30 Bytes pro Sekunde. Daraus ergibt sich bei einer C-60 Kassette eine Kapazität von ca. 50K-Bytes. Diese relativ niedrigen Werte kommen daher, daß aus Daten-Sicherungs-Gründen jede Information doppelt aufgezeichnet wird. Die hardwaremäßige Übertragungsrate liegt also eigentlich doppelt so hoch.

8.5 Rekorderpuffer

Im Gegensatz zu LOAD und SAVE, wo die Daten unmittelbar zwischen dem Arbeitsspeicher und dem Band ausgetauscht werden, muß die Information bei OPEN-Dateien in einem Rekorder-Puffer zwischengespeichert werden. Der Grund liegt darin, daß das Band nicht wegen jedem Byte gestartet und gestoppt werden kann - das würde ja 30 Start/Stop Zyklen pro Sekunde entsprechen.

Die Puffer für die beiden Rekorder liegen von Zelle 634-825 für Rekorder 1 bzw. 826-1017 für Rekorder 2. Sie fassen also jeweils 192 Bytes. Beim Schreiben (PRINT§) werden die Daten im Puffer gesammelt und automatisch auf Band geschrieben, sobald der Puffer voll ist. Entsprechend wird beim Lesen vom Band immer ein ganzer Datenblock in den Puffer übertragen. Dann wird solange gelesen, bis der Puffer leer ist und dann automatisch der nächste Datenblock geholt.

Bitte beachten Sie, daß die DISK-Befehle den Inhalt des Recorderpuffers 2 teilweise zerstören!

8.6 Zugelassene Daten

In eine Banddatei (OPEN, PRINT§) können mit Ausnahme von CHR$(10) (Line Feed) alle Codes geschrieben werden. Allerdings bewirkt dieser eine Code, der nicht mehr gelesen werden kann (er wird ausgelassen), daß durch PRINT§ keine reinen Binärdateien auf Band geschrieben werden können. Beachten Sie in diesem Zusammenhang auch die Einschränkungen von INPUT§ bei CHR$(0) (NUL) und CHR$(13) (Carriage Return).

8.7 Unterbrechen des Rekorders

Sie können den Rekorder jederzeit durch STOP unterbrechen, haben dann aber keine Möglichkeit, in dieser Datei weiterzulesen, oder in sie weiterzuschreiben. Deshalb muß auch der Anwender darauf achten, daß er nicht während Band-Schreib-/Lese-Operationen die STOP-Taste drückt, da zum Beispiel beim Schreiben der Verlust der eben geschriebenen Daten und eventuell der ganzen Datei verbunden ist.

9. Anschlüsse für die beiden Rekorder

Der Anschluß für den Rekorder 1 (Steckverbinder J3) befindet sich an hinteren rechten Ecke des Computers. Der Anschluß für Rekorder 2 (Steckverbinder J6) ist im Rechner auf der Platine links vorn angebracht.

Die Anschlüsse sind als Platinenstecker mit 6 Positionen und 12 Kontakten (Kontaktabstand 0,156" bzw. 3,96 mm) realisiert. Ein Codierschlitz befindet sich zwischen den Kontakten 2-3.

Diese Anschlüsse sollten ausschließlich für Kassettenrekorder von Commodore benutzt werden. Bitte beachten Sie, daß die Spannung von +5 Volt nicht als Spannungsquelle für andere externe Geräte vorgesehen ist.

Kontaktbelegung

Anschluß	Bez.	Beschreibung
A - 1	GND	Masse (Digital)
B - 2	+ 5 V	Spannungsversorgung für die Elektronik
Codierschlitz		
C - 3	MOTOR	+ 6 V (vom Computer gesteuert) für Motor
D - 4	READ	Lesen von Kassette
E - 5	WRITE	Schreiben auf Kassette
F - 6	SENSE	Diese Leitung meldet, ob eine der Tasten REW, FFWD oder PLAY gedrückt ist.

VI TIM-MONITOR

1. Die TIM-Befehle

M	Memory display	Anzeige des Speicherinhaltes
R	Register Display	Anzeige der Registerinhalte
G	Begin execution	Aufruf eines Programmes in Maschinensprache
X	Exit to BASIC	Rückkehr zu BASIC
L	Load	Programm laden
S	Save	Programm abspeichern

2. TIM-Zahlensystem

Generell zwei- oder vierstellige hexadezimale Ein- und Ausgabe.

3. Beschreibung der Befehle

3.1 Anzeige des Speicherinhalts .M XXXX YYYY

Anfangs- und Endadresse müssen vollständig als vierstellige Hex-Ziffern angegeben werden.

Um den Inhalt einer Speicherzelle zu verändern, wird der Cursor an die entsprechende Stelle bewegt, dann die Korrektur vorgenommen und mit der Taste RETURN bestätigt.

Beispiel:

```
.M C000,C010
.:C000  1D C7 48 C6 35 CC EF C7
.:C080  C5 CA DF CA 70 CF 23  CB
.:C010  9C C8 9C C7 74 C7 1F  C8
```

3.2 Anzeige der Registerinhalte .R

Die Inhalte der Register der CPU werden angezeigt:

PC	Programmzähler
IRQ	Interruptvektor
SR	Statusregister des Prozessors
AC	Akku
XR	Indexregister X
YR	Indexregister Y
SP	Stapelzeiger (Stack Pointer)

Beispiel:

```
.R
 PC   IRQ   SR  AC  XR  YR  SP
0401  6E2E  32  04  5E  00  EE
```

Änderungen können wie bei .M mit Hilfe des Cursors und der Taste RETURN vorgenommen werden. Bei jedem Eintritt von BASIC nach TIM werden die Registerinhalte gespeichert und bei der Rückkehr nach BASIC wiederhergestellt.

3.3 Programmausführung .G (XXXX)

Der Go-Befehl bewirkt einen Sprung zu der durch XXXX angegebenen Adresse. Wird XXXX nicht eingegeben, so dient der Inhalt des Programmzählers PC als Zieladresse.

Beispiel: .G C38B bewirkt Sprung nach Adresse $C38B

3.4 EXIT - Rückkehr zu BASIC .X

.X bewirkt einen Rücksprung zu BASIC. Die Speicherinhalte werden wiederhergestellt, somit befindet sich BASIC im selben Zustand wie vor dem Aufruf des Monitors.

3.5 Laden eines Programmes .L "NAME",XX

XX steht für die Nummer des Peripheriegeräts, Gerätenummer und Dateiname müssen angegeben werden. Die mit dem SAVE-Befehl definierten Adressen werden geladen.

Unterprogramme in Maschinensprache können auch von BASIC aus geladen werden. Dabei ist aber zu beachten, daß der Variablenpointer auf das zuletzt geladene Byte plus eins zeigt. Daher dürfen BASIC-Variablen nach LOAD nicht verwendet werden. Will man nach LOAD mit BASIC weiterarbeiten, so muß zuerst ein erneuter Sprung zum TIM erfolgen, dann eine ordnungsgemäße Rückkehr (RUN, dann .X).

Beispiel:
```
.L "NAME",08
SEARCHING FOR NAME
FOUND NAME
LOADING
```

3.6 SAVE-Speichern eines Programms .S "NAME",XX,YYYY,ZZZZ

XX steht für die Nummer des Peripheriegeräts (siehe .L), YYYY ist die hexadezimale Anfangsadresse und ZZZZ die Endadresse plus eins.

Beispiel: .S "NAME",08,033A,03F1

03F1 ist Endadresse plus eins, das letzte Datenbyte steht also in 03F0.

3.7 BREAK und Interrupts

Der Maschinenbefehl BRK ($00) bewirkt einen Software-Interrupt. Dies bedeutet, daß der Prozessor das gegenwärtige Programm unterbricht, (PC+2) und SR auf dem Stapel ablegt und dann an eine Stelle verzweigt, die durch den Vektor in $021B und $021C gegeben ist. TIM initialisiert diesen Vektor in der Weise, daß er auf TIM zeigt. Nach einem BRK-Befehl übernimmt TIM daher die Kontrolle druckt B* aus (entry via breakpoint im Gegensatz zu C*, entry via call), zeigt die Registerinhalte an und wartet auf die Befehle vom Operator.

Der oben erwähnte Vektor kann auch verändert werden, etwa um auf eine spezielle Routine zu verzweigen.

Beachten Sie:

Nach einem BRK, der auf TIM verzweigt, zeigt der PC auf das dem BRK-Befehl folgenden Byte.

Bei einem BRK, der nicht auf TIM verzweigt, zeigt der PC nach der Rückkehr (via RTI) auf das zweite Byte nach dem BRK-Befehl.

4. Aufruf von TIM

TIM ist fest im Betriebssystem eingebaut, muß also nicht geladen werden. Er kann durch

```
SYS 1024
```

oder mit dem Befehl

```
SYS 54386
```

aufgerufen werden. Der erste Fall stellt einen "entry via breakpoint" dar, der zweite einen "entry by call".

VII. USER-PORT

1. Was ist der USER-PORT?

Der USER-PORT ist eine Anschlußmöglichkeit für Spezialperipheriegeräte, die nicht über einen der IEC-Norm entsprechenden Anschluß verfügen. Darüberhinaus sind durch ihn Signale herausgeführt, die für Diagnosezwecke nötig sind. Er stellt aber auch die Signale für ein Monitor- bzw. TV-Interface zur Verfügung.

2. Kontaktbelegung des Steckverbinders J2

Die Leitungen für dieses Interface führen auf der Hauptplatine zu einem Anschluß mit 12 Positionen und 24 Kontakten, die einen Zwischenraum von 3,96 mm zwischen den Kontaktmittelpunkten haben.

Codierschlitze befinden sich zwischen den Kontakten 1-2 und 10-11. Beachten Sie, daß die Verbindungen 1 bis 12 (die obere Reihe der Kontakte) vor allem für die Kundendienstabteilung oder Fachhändler gedacht ist. Bei Verwendung der im ROM integrierten Diagnose-Routine wird ein Spezialsteckverbinder benutzt. Dieser schließt einige der oberen Kontakte mit den unteren kurz. Wir empfehlen dringend, die oberen Verbindungen 1 bis 12 nur mit äußerster Vorsicht zu benutzen.

Wenn Sie von hinten auf den Computer blicken, sehen Sie beim mittleren Anschluß von links nach rechts folgende Kontakte:

Kontakt	Signal	Beschreibung
OBEN:		
1.	GND	Masse (Digital)
Codierschlitz		
2.	TV-Video	Video-Ausgang für externen Bildschirm.
3.	IEEE-SRQ	Für Testroutine
4.	IEEE-EOI	Für Testroutine
5.	Diagnose?	Testroutine wird gerufen, wenn 'low' beim Einschalten
6.	§ 1 READ	Überprüfung der Lesefunktion von Rekorder 1
7.	§1,§2 WRITE	Überprüfung der Schreibfunktion beider Rekorderanschlüsse
8.	§ 2 READ	Überprüfung der Lesefunktion von Rekorder 2
9.	TV-VERT	Vertikalsynchronsignal (60 Hz)
10.	TV-HOR	Horizontalsignal
Codierschlitz		
11.	GND	Masse (Digital)
12.	GND	Masse (Digital)
UNTEN:		
A.	GND	Masse (Digital)
Codierschlitz		
B.	CA1	Flankengetriggerter Anschluß des VIA 6522.
C.	PA0	
D.	PA1	
E.	PA2	
F.	PA3	PA0 bis PA7 sind einzeln als Eingangs- oder
H.	PA4	Ausgangsleitungen programmierbar. Sie sind
J.	PA5	direkt mit dem VIA 6522 verbunden.
K.	PA6	
L.	PA7	
Codierschlitz		
M.	CB2	Anschluß CB2 des VIA 6522
N.	GND	Masse (Digital)

3. Der Interface-Baustein VIA 6522

Die Leitungen an der Unterseite des User-Ports führen zum "vielseitigen Interface-Adapter" auf der Hauptplatine. Es ist dies der Baustein 6522 von MOS-Technology. (Versatile Interface Adapter; VIA.)

Der parallele Port besteht aus acht programmierbaren Zweirichtungs-Leitungen (PA0 bis PA7), einer Eingabe-Handshake Leitung (CA1), die auch für andere flankengetriggerte Eingaben verwendet werden kann und einer sehr starken Verbindung CB2. Diese hat die meisten Fähigkeiten von CA1, kann aber zusätzlich auch als Ein- und Ausgabeleitung des Schieberegisters im VIA arbeiten.

Alle Signale des VIA, die nicht mit dem User-Port verbunden sind, werden vom Computer für interne Kontrollen benutzt. Der Anwender sollte auf keinen Fall diese Signalleitungen benutzen.

Näheres entnehmen Sie bitte dem Datenblatt des 6522.

4. Die Adressen des 6522 im CBM-Computer

Dez.	Hex.	$E840+	Adressierte Speicherstelle/Tätigkeit
59456	E840	0000	Output Register für I/O-Port B (ORB)
59457	E841	0001	Output Register für I/O-Port A (ORA) mit Handshake
59458	E842	0010	I/O-Port B Datenrichtungsregister (DDRB)
59459	E843	0011	I/O-Port A Datenrichtungsregister (DDRA)
59460	E844	0100	Lies LSB des Zählers im Timer 1 und schreibe es als LSB ins Latch vom Timer 1.
59461	E845	0101	Lies MSB des Zählers im Timer 1, schreibe es als MSB ins Latch vom Timer 1 und beginne zu zählen
59462	E846	0110	Hole LSB vom Latch des Timers 1.
59463	E847	0111	Hole MSB vom Latch des Timers 1.
59464	E848	1000	Lies LSB des Timers 2 und setze IFR zurück. Schreibe es als LSB ins Latch vom Timer 2 ohne Reset
59465	E849	1001	Lies MSB des Timers 2. Reset IFR beim Schreiben ins MSB des Latch.
59466	E84A	1010	Serielles I/O-Schieberegister (SR)
59467	E84B	1011	Hilfskontrollregister (ACR)
59468	E84C	1100	Peripherie-Kontrollregister (PCR)
59469	E84D	1101	Interrupt Flag-Register (IFR)
59470	E84E	1110	Interrupt Enable-Register (IER)
59471	E84F	1111	Output Register für I/O-Port A ohne Handshake

Programmierung des VIA 6522

Die Datenleitungen PA0 bis PA7 können programmiert werden, um jeweils für Eingabe oder Ausgabe zu arbeiten. Dies wird durch

POKE 59459,X

erreicht, wodurch die Zahl X in das Datenrichtungsregister A geschrieben wird. Die Bits 0 bis 7 der Zahl X dienen zur Programmierung der Leitungen PA0 bis PA7. Eine Null in einem Bit der Zahl X programmiert dabei die entsprechende PA-Leitung als Input.

Die Programmierung braucht nur einmal am Anfang des Programms zu erfolgen. Dann übernimmt POKE 59471 die Ansteuerung der Output-Leitungen und PEEK (59471) liest alle Inputs.

Beispiel für Programmierung der Leitungen PA0 bis PA7 des User Ports:

Befehl	Binär	bewirkt
POKE 59459,255	11111111	PA0 bis PA7 sind Output
POKE 59459,0	00000000	PA0 bis PA7 sind Input
POKE 59459,240	11110000	PA0 bis PA3 sind Input, und PA4 bis PA7 sind Output

VIII. IEC-BUS

1. Einführung

Der IEC-Bus ist die wichtigste Verbindung Ihres Computers zur Außenwelt, da an Ihm alle Commodore-Peripheriegeräte (Floppy, Drucker, Plotter, Modem), sowie auch Peripheriegeräte anderer Hersteller angeschlossen werden können. Commodore--Geräte werden vollständig von BASIC aus verwaltet, dazu müssen Sie sich also nicht mit Details der IEC-Norm belasten. Fremde Geräte können in der Regel ebenfalls von BASIC aus bedient werden. Die in diesem Kapitel gegebenen Informationen sind also nur dazu da, um Ihre eventuelle Neugierde zu befriedigen.

2. Kontaktbelegung der Steckverbindung J1

Die Verbindungen zu dem aus 12 Positionen und 24 Kontakten bestehenden Platinenstecker gehen von der Hauptplatine des Computers aus. Wenn Sie von hinten auf Ihren Computer blicken, ist der IEC-Anschluß der linke der beiden breiten Anschlüsse.

Der Standard-Steckverbinder nach IEEE-488 ist am Computer nicht vorhanden. Stattdessen wird ein Direktstecker mit 12 Positionen und 24 Kontakten verwendet. Kontaktabstand 0,156" (3,96 mm). Fan-out und Impedanzanpassung entsprechen IEEE-488, (eine TTL-Last, 130 pF).

Kontakt-benennung	Standard IEEE	IEEE-Bezeichnung
OBEN:		
1	1	DI01
2	2	DI02
Codierschlitz		
3	3	DI03
4	4	DI04
5	5	EOI
6	6	DAV
7	7	NRFD
8	8	NDAC
9	9	IFC
Codierschlitz		
10	10	SRQ
11	11	ATN
12	12	GND (Chassis)
UNTEN:		
A	13	DI05
B	14	DI06
Codierschlitz		
C	15	DI07
D	16	DI08
E	17	REN
F	18	GND (DAV)
H	19	GND (NRFD)
J	20	GND (NDAC)
K	21	GND (IFC)
Codierschlitz		
L	22	GND (SRQ)
M	23	GND (ATN)
N	24	GND (DI01-8)

3. Eigenschaften des IEC-Bus

Bestimmte technische Einschränkungen sollten beim Anschluß von Geräten an den IEEE-Bus beachtet werden:

a) Die größtmögliche Entfernung eines Gerätes vom Computer beträgt 20 Meter.
b) Der größte Abstand zwischen einzelnen Geräten soll nicht mehr als 5 Meter betragen.
c) Es können maximal 15 Geräte gleichzeitig am Bus abgeschlossen sein.

Der IEC-Bus, auch IEEE 488 Bus genannt, besteht im wesentlichen aus 16 Signalleitungen, die von der Funktion her in drei Gruppen aufgeteilt sind:

a) Daten-Bus,
b) Kontroll-Bus
c) Management-Bus.

Der IEC-Bus kann drei Arten von Geräten versorgen:

a) Talker (Sprecher)

Beispiele für Talker sind Floppy, Meßwerterfassungsgeräte und Modem-Schnittstellen.

Zu jedem Zeitpunkt kann nur ein Gerät Daten an den Datenbus übermitteln.

b) Listener (Zuhörer)

Beispiele für Listener sind Floppy, Drucker und Plotter.

Es können mehrere Geräte zur gleichen Zeit Daten vom Bus empfangen.

c) Controller (Kontrollgerät)

Nur der Computer darf den IEC-Bus kontrollieren.

Es darf nur ein Controller am Bus angeschlossen sein.

4. Die Funktion der einzelnen Leitungen

4.1 Der Datenbus

Der Datenbus umfaßt 8 Zweirichtungsleitungen, (aktiv low) für die Datensignale DIO1 bis DIO8. Die Übertragungsart ist byteseriell und bitparallel. Das am langsamsten arbeitende Gerät kontrolliert zu jedem Zeitpunkt die Übertragungsgeschwindigkeit.

Das höchstwertige Bit (MBS) liegt auf Leitung DIO8. Peripherieadressen und Kontrollinformationen werden ebenfallls über den Datenbus übertragen. Sie werden von Daten dadurch unterschieden, daß während der Übertragung ATN = wahr gilt.

Bei der Datenübermittlung an die Geräte können alle möglichen Bitmuster als Daten verwendet werden.

4.2 Der Kontrollbus (Transfer-Bus)

Diese drei Leitungen kontrollieren die Übertragung von Daten über den Datenbus. Diese Signale werden im Handshake-Verfahren verwendet. Zum Tansfer-Bus gehören:

a)	NRFD	nicht aufnahmebereit für Daten	(Not Ready For Data)
b)	NDAC	Daten nicht angenommen	(Not DAta Accepted)
c)	DAV	Daten gültig	(DAta Valid)

Beachten Sie, daß DAV aus dem Talker, und NRFD und NDAC aus dem Listener kommen.

Zeitliche Beschränkungen

Um kein Daten zu verlieren, sollten die folgenden zeitlichen Beschränkungen beachtet werden:

a) Der Computer als Listener erwartet DAV = LOW innerhalb von 64 Millisekunden, nachdem er NRFD hochgesetzt hat.
b) Der Rechner als Talker erwartet NDAC = HIGH innerhalb von 64 Millisekunden nachdem er NRFD hochgesetzt hat.

Werden diese zeitlichen Grenzen überschritten, stoppt der Computer die Übertragung und setzt das Statuswort (ST) (s. Status).

Durch POKE 1020,128 kann diese Zeitüberwachung unterdrückt werden.

4.3 Der Management-Bus

Diese aus fünf Signalleitungen bestehende Gruppe kontrolliert den Zustand des Daten-Busses und definiert seine Signale; das können Daten, Adressen oder Kontrollinformationen (Gerätebefehle) sein.

Die fünf Management-Signale sind:

ATN	Attention	Wählt Daten- oder Kommandoübertragung aus
EOI	End Or Identify	Zeigt an, daß das letzte Datenbyte übertragen worden ist. (Ende oder Kennzeichnung)
IFC	Interface clear	Initialisiert den Datenbus. Talker- und Listenerzuteilung wird aufgehoben. Das gleiche Signal wie Reset im Computer.
SRQ	Service request	Gerät teilt Controller mit, daß Bedienung nötig ist. Wirkt nicht auf den Ablauf des BASIC-Programms.
REN	Remote ENable	Dieses Signal liegt bei Ihrem Computer immer auf LOW.

5. Definition der Logiksignale

"Wahr", also "logisch Eins" wird durch eine Spannung nahe Null realisiert (Negative Logik). Aufgrund der Buskonstruktion mit gemeinsamen Kollektorausgängen kann so jedes Gerät den "Wahr"-Zustand erzeugen und erhalten, auch wenn alle anderen Geräte anders wollen.

6. Registeradressen

Die folgende Tabelle zeigt die IEEE-488 Hardware-Adressen. Der Versuch, den Bus durch PEEK und POKE-Befehle zu kontrollieren, schlägt fehl, wenn die festgesetzten Zeit-Intervalle für 488-Geräte überschritten werden.

Hexadez. Adressen	Dezimale Adressen	Bits	IEEE	Betriebsart
E820	59424	0-7	DIO1-8	Eingang
E822	59626	0-7	DIO1-8	Ausgang
E821	59425	3	NDAC	Ausgang
E823	59427	3	DAV	Ausgang
		7	SRQ	
E810	59408	6	EOI	Eingang
E840	59456	0	NDAC	Eingang
		1	NRFD	Ausgang
		2	ATN	Ausgang
		6	NRFD	Eingang
		7	DAV	Eingang

7. Beschreibung der Bus-Signale

Gruppe	Signal	Name	Funktionelle Beschreibung
Manager	ATN	Attention	Der Controller setzt dieses Signal auf low, während er Befehle an den Datenbus gibt. Ist ATN = Low, so befinden sich nur Peripherie-Adressen und Kontrollmitteilungen am Datenbus. Ist ATN=HIGH, so können nur vorher zugeordnete Geräte Daten übermitteln.
Transfer	DAV	Data Valid (Daten gültig)	Ist DAV=LOW, dann sind die Daten am Datenbus gültig.
Manager	EOI	End Or Identify (Ende oder Kennzeichen)	Sobald das letzte Datenbyte übertragen ist, kann der Talker EOI senden. Der Computer setzt EOI immer auf low, während das letzte Datenbyte aus dem Computer übertragen wird.
Manager	IFC	Interface Clear	Der Computer sendet sein internes Resetsignal als IFC, um alle Geräte zu initialisieren. Wird der Computer ein- oder ausgeschaltet, dann geht IFC für ungefähr 100 Millisekunden auf LOW.
Transfer	NDAC	Data not accepted (Daten nicht angenommen)	Dieses Signal wird durch den Listener während des Einlesens auf low gehalten. Ist das Datenbyte eingelesen, setzt der Listener NDAC hoch. Dies signalisiert dem Talker daß die Daten angenommen wurden.
Transfer	NRFD	Not Ready for Data (nicht bereit, Daten aufzunehmen)	Wenn NRFD LOW ist, sind einer oder mehrere Listener nicht bereit für das nächste Datenbyte. Wenn alle Geräte aufnahmebereit sind, geht NRFD hoch.
Manager	SRQ	Service Request (Bedienungsaufruf)	Wird nicht von BASIC bedient, kann aber über Software realisiert werden.
Manager	REN	Remote enable	REN wird vom Bus-Controller auf low gehalten. Ihr Computer hält REN immer auf low.
Daten	DIO1 bis DIO8	Daten - Ein-/Ausgabe-Leitungen 1-8	Diese Signale stellen die Informationsbits am Datenbus dar. Liegt ein EIO-Signal auf low, so bedeutet dies, das betreffende Bit ist Null (positive Logik).
Allgemein	GND	Masse	Erdungsverbindungen: Es gibt 6 Masseleitungen für die Kontroll- und Managementbits, eines für den Datenbus und eine für Chasis.

IX INTERNER BUS (J4,J9)

Anschl.	Bez.	Beschreibung
J9 - 1	GND	Masse (digital)
J9 - 2	BA0	
J9 - 3	BA1	
J9 - 4	BA2	
J9 - 5	BA3	
J9 - 6	BA4	
J9 - 7	BA5	Gepufferte Adreßbits A0 bis A15
J9 - 8	BA6	
J9 - 9	BA7	
J9 - 10	BA8	
J9 - 11	BA9	
J9 - 12	BA10	
J9 - 13	BA11	
J9 - 14	BA12	
J9 - 15	BA13	
J9 - 16	BA14	
J9 - 17	BA15	
J4 - 10	SEL 2	2000 - 2FFF wählen je einen
J4 - 11	SEL 3	3000 - 3FFF 4 kBlock-Block an.
J4 - 12	SEL 4	4000 - 4FFF
J4 - 13	SEL 5	5000 - 5FFF
J4 - 14	SEL 6	6000 - 6FFF
J4 - 15	SEL 7	7000 - 7FFF
J4 - 16	SEL 8	8000 - 8FFF
J4 - 17	SEL 9	9000 - 9FFF
J4 - 18	SEL A	A000 - AFFF
J4 - 19	SEL B	B000 - BFFF
J4 - 25	GND	Masse (digital)
J4 - 22	RES	Resetleitung (Power-on-reset)
J9 - 19	IRQ	Interrupt-Request-Leitung des 6502
J9 - 21	PHI2	Takt 2.
J9 - 22	R/W	Lese-Schreib-Leitung des 6502, gepuffert
J4 - 2	BD0	
J4 - 3	BD1	
J4 - 4	BD2	
J4 - 5	BD3	Gepufferte Datenbits D0 bis D7
J4 - 6	BD4	
J4 - 7	BD5	
J4 - 8	BD6	
J4 - 9	BD7	
J9 - 18	SYNC	
J9 - 20	DIAG	
J9 - 22	R/W	
J9 - 23	R/W	Lese-Schreibleitung des 6502, gepuffert
J4 - 1	GND	Masse (digital)
J4 - 23	RDY	Ready-Leitung des 6502
J4 - 24	NMI	Nicht maskierbarer Interrupt
J9 - 25	GND	Masse (digital)

X INTERPRETERCODE

Der BASIC-Interpreter legt jede BASIC-Anweisung als 1-Byte-Code im Speicher ab. Diese Codes, sowie die Abkürzungen der BASIC-Anweisungen sind in der nachstehenden Tabelle aufgeführt. Bitte beachten Sie bei der Eingabe von Abkürzungen, daß die hier **groß** gedruckten Buchstaben beim Rechner **ohne SHIFT** und entsprechend die **klein** gedruckten **mit SHIFT** eingegeben werden müssen, auch wenn das Erscheinungsbild auf dem Bildschirm gerade anders herum aussieht.

Von 0 bis 127 gilt der ASC-Code

Code	BASIC-Anweisung	Abkürzung
128	END	En
129	FOR	Fo
130	NEXT	Ne
131	DATA	Da
132	INPUT§	In
133	INPUT	INPUT
134	DIM	Di
135	READ	Re
136	LET	Le
137	GOTO	Go
138	RUN	Ru
139	IF	IF
140	RESTORE	REs
141	GOSUB	GOs
142	RETURN	REt
143	REM	REM
144	STOP	St
145	ON	ON
146	WAIT	Wa
147	LOAD	Lo
148	SAVE	Sa
149	VERIFY	Ve
150	DEF	De
151	POKE	Po
152	PRINT§	Pr
153	PRINT	?
154	CONT	Co
155	LIST	Li
156	CLR	Cl
157	CMD	Cm
158	SYS	Sy
159	OPEN	Op
160	CLOSE	CLo
161	GET	Ge
162	NEW	NEW
163	TAB(	Ta
164	TO	TO
165	FN	FN
166	SPC(	Sp
167	THEN	Th
168	NOT	No
169	STEP	STe

Code	BASIC-Anweisung	Abkürzung
170	+	+
171	-	-
172	*	*
173	/	/
174	Pfeil nach oben	
176	OR	OR
177	kleiner als	
178	=	=
179	größer als	
180	SGN	Sg
181	INT	INT
182	ABS	Ab
183	USR	Us
184	POS	POS
185	FRE	Fr
186	SQR	Sq
187	RND	Rn
188	LOG	LOG
189	EXP	Ex
190	COS	COS
191	SIN	Si
192	TAN	TAN
193	ATN	At
194	PEEK	Pe
195	LEN	LEN
196	STR$	STr
197	VAL	Va
198	ASC	As
199	CHR$	Ch
200	LEFT$	LEf
201	RIGHT$	Ri
202	MID$	Mi
203	GO	GO
204	CONCAT	CONc
205	DOPEN	Do
206	DCLOSE	Dc
207	RECORD	REc
208	HEADER	He
209	COLLECT	COl
210	BACKUP	Ba
211	COPY	COp
212	APPEND	Ap
213	DSAVE	Ds
214	DLOAD	Dl
215	CATALOG	Ca
216	RENAME	REn
217	SCRATCH	Sc
218	DIRECTORY	DIr
219	erzeugt SYNTAX ERROR bei LIST	

220 - 254 noch nicht verwendet

255 PI (Symbol für Kreiszahl 3.14...)

XI TECHNISCHE DATEN DES CBM 4032

Höhe	: 36 cm
Breite	: 42 cm
Tiefe	: 47 cm
Gewicht	: 22,7 kg
Spannung	: 220 V +/- 10 % / 50 Hz
Verbrauch	: 250 W

Bildschirm

Diagonale	: 31 cm
Zeilen	: 25
Spalten	: 40
Zeichenauflösung	: 8 * 8 Punktmatrix vorprogrammiert
Darstellungsart	: grün auf dunklem Grund (programmierbar dunkel auf grünem Grund)
Sonderzeichen	: 64 grafische Zeichen

Programmierung

Sprache	: Commodore-BASIC festgespeichert in 18 kB ROM
Anzahl Befehle	: 91
Klartextmeldungen	: 26
Kapazität (RAM)	: 32 kB

Kassette	: Rekorder für Standard Musikkassetten
Aufzeichnungsformat	: Einspur-Frequenzmodulation nach eigenem Standard
Anschlüsse	: IEC-Bus 8-bit Parallel mit 2 Handshake-Leitungen 1. Kassettenrekorder 2. Kassettenrekorder Interner Bus
Prozessor	: 6502
Taktfrequenz	: 1 MHZ

commodore

Commodore GmbH
Postfach 426
D-6078 Neu-Isenburg

Commodore AG
Aeschenvorstadt 57
CH-4010 Basel

Steiner Computer
Vertriebsgesellschaft mbH
Fleschgasse 2
A-1130 Wien

200c-1181

Änderungen vorbehalten