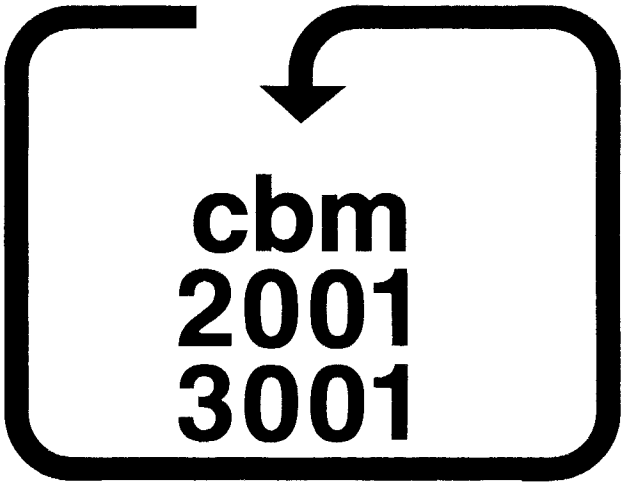


commodore - computer

Bedienungshandbuch

Einleitung

Diese Einleitung befaßt sich mit dem grundsätzlichen Kennenlernen Ihres Computers.

Sollten Sie mit dessen Bedienung schon etwas vertraut sein, so dürfte diese Anleitung für Sie kaum etwas Neues bieten. Wir halten es jedoch für unerläßlich, Sie mit der Grundbedienung vertraut zu machen. Meist ist es ja so, daß der Neuling an Kleinigkeiten verzweifelt, die dem Erfahrenen nur ein müdes Lächeln abringen.

Stecken Sie also den Schukostecker in die Steckdose und schalten das Gerät einmal ein. Der Einschalter befindet sich auf der Rückseite des Rechners links unten. Nach kurzer Zeit haben Sie ersten Kontakt; der Rechner meldet sich auf dem Bildschirm mit der Meldung:

```
# # # COMMODORE BASIC # # #
7167 BYTES FREE
READY.
```

Die Zahl 7167 (oder 15359 bei den Geräten der Serie 3016 bzw. 31743 beim 3032) sagt Ihnen übrigens, wie groß der Schreib-Lese-Speicher Ihres Computers ist.

Die Meldung # # # COMMODORE BASIC # # # sagt Ihnen, in welcher Sprache der Rechner programmiert werden kann. Auf die Sprache BASIC kommen wir ausführlich in Kapitel 1 zu sprechen.

Sollte wider Erwarten der Einschaltvorgang nicht korrekt ablaufen, so versuchen Sie bitte folgendes:

1. Drehen Sie den Helligkeitsregler für den Bildschirm entgegen dem Uhrzeigersinn bis zum Anschlag. Sollte die Meldung des Rechners jetzt sichtbar sein, so drehen Sie den Regler wieder soweit zurück, bis die Schrift normal hell sichtbar ist.
2. Schalten Sie den Rechner wieder aus. Ziehen Sie den Netzstecker aus der Steckdose.

   Überprüfen Sie die Sicherung, die Sie mit einem Schraubenzieher herausdrehen können (die Sicherung befindet sich direkt neben dem Einschalter). Sollte die Sicherung noch in Ordnung sein, so bleibt Ihnen nichts anderes übrig, als Ihren Fachhändler aufzusuchen.

Nachdem Sie nun den Rechner eingeschaltet haben, wollen wir kurz einmal den Begriff der Dialogfähigkeit des Rechners erläutern. Die Dialogfähigkeit des Rechners besagt nichts anderes, als daß:

1. Sowohl eine Eingabemöglichkeit, als auch eine Ausgabemöglichkeit des Rechners in Sicht- und Griffweite des Bedieners sein müssen (bei Ihrem Gerät sind das der Bildschirm und die Tastatur).
2. Daß sowohl der Rechner fragen kann und Sie antworten als auch Sie Fragen oder Aufgaben stellen können und der Rechner antwortet.
3. Der Rechner Ihnen offensichtliche Fehler, die Sie bei der Programmierung gemacht haben, möglichst genau lokalisiert, so daß eine Korrektur leicht und schnell möglich ist.

Alle diese Möglichkeiten bietet Ihnen der Computer. Er kann somit ohne Einschränkungen als dialogfähig bezeichnet werden.

Sehen wir uns einmal die Tastatur genauer an. Grundsätzlich gliedert sich die Tastatur in sieben Untergruppen:

1. **Die alphabetischen Tasten**

   Drücken Sie einmal leicht auf die Taste Q. Wie zu erwarten, erscheint auf dem Bildschirm an der Stelle, wo sich das blinkende Quadrat gerade noch befand, ein Q und das Quadrat wandert eine Stelle weiter nach rechts (es nennt sich übrigens CURSOR).

   Der Computer reagiert natürlich auf alle Buchstaben, Zahlen und Sonderzeichen genauso, wie auf das Q.

2. **Die numerischen Tasten**

   An sich ist eine Trennung dieser Tasten von den anderen nicht nötig. Da in einem Computer sehr oft Zahlen eingegeben werden, ist es aber sinnvoll, die Zifferntasten zusammenfassen, damit Sie ohne große Handbewegung schnell hintereinander erreichbar sind.

   Erstaunen mag Sie als EDV-Neuling wohl, was der Punkt in diesem Block zu suchen hat, während das Komma, bei jeder Zahl mit Nachkommastellen unerläßlich, ganz wo anders angebracht ist.

   Doch das ist schnell erklärt.

In den U.S.A., wo auch Ihr Computer herkommt, gibt es kein Dezimalkomma, sondern ein Punkt trennt Vor- und Nachkommastellen. Wenn Sie sich diese Besonderheit schon jetzt einprägen, werden Sie später viele Eingabefehler an Ihrem Rechner umgehen können.

3. **Die grafischen Zeichen** (Siehe Anhang VIII für die –2–Tastatur)

   Bestimmt haben Sie die seltsamen Hieroglyphen auf der Tastatur schon verwundert. Das sind grafische Zeichen, mit denen man z. B. Tabellen oder einfache Bilder herstellen kann. Um diese Zeichen auf dem Bildschirm darzustellen, machen Sie folgendes:

A) Drücken Sie eine der beiden SHIFT-Tasten (im alphabetischen Tastenblock untere Reihe links und rechts).

B) Halten Sie diese Taste gedrückt und drücken Sie gleichzeitig beispielsweise die Taste Q. Auf dem Bildschirm erscheint das zugehörige Sonderzeichen (statt eines Q beispielsweise ein weißer Kreis). Siehe auch 1.2.36.

4. **Die CURSOR-Steuertasten**

   Was der Cursor (sprich Körser) ist, haben wir bereits einmal kurz angesprochen. Wir wollen an dieser Stelle jedoch einmal genauer darauf eingehen.

   Prinzipiell hat der Cursor 2 Funktionen:

   Erstens teilt der Computer Ihnen dadurch mit, daß er auf eine Eingabe per Tastatur von Ihnen wartet.

   Zweitens zeigt er Ihnen an, wo auf dem Bildschirm diese Eingabe angezeigt wird. Das durch die Tastatur gewählte Zeichen erscheint genau an der Stelle, wo sich der Cursor gerade befindet.

   Nun ist es oftmals nötig, den Cursor zu bewegen, ohne daß ein Zeichen geschrieben wird, oder Sie haben sich verschrieben und wollen ein Zeichen nachträglich einfügen, ändern oder löschen. Dazu dienen die Cursor-Steuertasten.

   Diese befinden sich in der oberen Reihe des numerischen Tastaturblocks.

   Sehen wir uns ein Beispiel an:

   Drücken Sie gleichzeitig

   [SHIFT] + [CLR HOME]

   Der gesamte Bildschirminhalt wird gelöscht und der Cursor steht in der oberen Zeile links.

   Geben Sie nun einen beliebigen Text ein, z. B.:

   DER cbm IST EIN COMPUTER

   Nach jedem Buchstaben bewegt sich der Cursor eine Stelle weiter nach rechts und steht zum Schluß hinter dem R von Computer.

   Wir wollen nun den Text ändern, daß er zum Schluß heißt:

   DER cbm – MEIN COMPUTER

   Zuerst löschen wir das IST und ersetzen es durch –.

   Drücken Sie dazu die Tasten

   [SHIFT] [← CRSR →]

Wie Sie sehen, bewegt sich der Cursor um eine Stelle nach links. Er steht jetzt auf dem R von Computer. Wiederholen Sie den Vorgang, bis der Cursor auf dem T von IST steht.

Nun drücken Sie zweimal die Taste

[INST DEL]

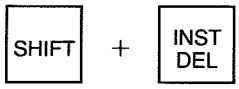
Nach jedem Tastendruck verschwindet das sich jeweils links neben dem Cursor befindliche Zeichen und der Rest der Zeile wird, einschließlich dem Cursor, um eine Stelle nach links nachgeführt.

Geben Sie jetzt einfach das Zeichen – ein; es überschreibt das vom IST übrig gebliebene T. Mit der Taste

(aber ohne 

) bewegen Sie jetzt den Cursor auf das E von EIN.

Mit den Tasten

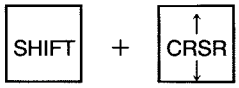


machen Sie nun Platz für das noch fehlende M. Dabei wird der Rest der Zeile ab dem Cursor um eine Stelle nach rechts verschoben. Zum Schluß gehen Sie mit der Taste

wieder an das Ende des Satzes.

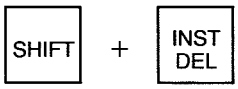
In diesem Beispiel haben wir den Cursor in einer Zeile hin- und her bewegt. Genauso leicht ist es aber auch möglich, in die darunter liegende Zeile

oder darüber liegende Zeile



zu springen.

Zwei Besonderheiten an dieser Stelle noch zu dem Bildschirm: wenn beim Einfügen mit den Tasten



der Rest der Eingabe das Ende der Zeile erreicht hat, so wird er in die nächste tiefere Zeile schreibrichtig übertragen. Über mehr als eine Zeile ist das aber nicht möglich. Das Einfügen von Zeichen wird dann blockiert. Das geschieht aus folgendem Grund:

In seinem Speicher behandelt der Rechner zwei Bildschirmzeilen á 40 Zeichen wie eine Zeile á 80 Zeichen. Mehr als 80 Zeichen können also nicht gleichzeitig behandelt werden. Die zweite Besonderheit stellen Sie fest, wenn Sie mit dem Cursor in die unterste Zeile »fahren«. An sich müßte der Cursor jetzt bei weiteren Versuchen, noch tiefer zu fahren, stehen bleiben. Z. B. am oberen Rand, wenn Sie noch höher fahren wollen. Das geschieht zwar auch, der Bildschirminhalt wird um eine Zeile nach oben geschoben und die oberste Zeile des Bildes verschwindet. Man nennt das »der Bildschirm rollt« oder auch

ROLL MODE

im Gegensatz zum stehenden Bild, das man

BLOCK MODE

nennt.

Die verschwundene Zeile ist endgültig gelöscht, da der Roll Mode nur in einer Richtung arbeitet und zwar nur von unten nach oben.

Sie selber haben gesehen, welche Vorteile die Cursor-Steuertasten haben. Ihr Gebrauch wird Ihnen schnell zur Gewohnheit werden.

Daher noch einmal eine Zusammenfassung

[←→ CRSR]

CURSOR RIGHT (Cursor nach rechts)
: Der Cursor bewegt sich eine Stelle nach rechts;

[SHIFT] [←→ CRSR]

CURSOR LEFT (Cursor nach links)
: Der Cursor bewegt sich eine Stelle nach links;

[↑↓ CRSR]

CURSOR DOWN (Cursor nach unten)
: Der Cursor bewegt sich um eine Zeile nach unten;

[SHIFT] [↑↓ CRSR]

CURSOR UP (Cursor nach oben)
: Der Cursor bewegt sich um eine Zeile nach oben;

[INST DEL]

DELETE (Zeichen löschen)
: Das links von dem Cursor befindliche Zeichen wird gelöscht. Gleichzeitig wird der gesamte rechte Teil der Zeile zusammen mit dem Cursor um ein Zeichen nach links versetzt.

[SHIFT] [INST DEL]

INSERT (Zeichen einfügen)
: Ab der Position des Cursors wird der rechte Teil der Zeile um eine Stelle nach rechts verschoben. Der Cursor bleibt an der gleichen Stelle stehen, wo sich nun ein Leerzeichen befindet.

[CLR HOME]

HOME (Cursor in Grundstellung)
: Der Cursor wird in seine Grundstellung nach links oben versetzt.

[SHIFT] [CLR HOME]

CLEAR (löschen des Bildschirms)
: Zusätzlich zu der Home-Funktion wird der gesamte Bildschirminhalt gelöscht.

5. Die RVS-Taste

[OFF RVS] (Reverse = umgekehrt)

Mit dieser Taste können Sie alle Zeichen auf dem Bildschirm dunkel auf hellem Untergrund stellen. Man nennt das

INVERSE VIDEO (umgekehrter Bildschirm)

Diese Darstellungsart wird bevorzugt, um besonders wichtige Meldungen auf dem Bildschirm hervorzuheben. Die RVS-Taste hat drei Bedeutungen:

1. [OFF RVS]

Nach Drücken dieser Taste erscheinen alle folgenden Zeichen invertiert, bis die Funktion ausgeschaltet, oder die Taste »RETURN« gedrückt wird.

2. [SHIFT] + [OFF RVS]

Nach Drücken dieser Taste wird inverse video wieder abgeschaltet und alle weiteren Zeichen erscheinen wieder normal.

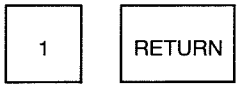
3. [OFF RVS]

Sollten Sie einmal Informationen auf dem Bildschirm nicht lesen können, weil im Roll-Mode die Zeilen zu schnell »vorbeilaufen«, so kann durch Drücken der Taste »OFF RVS« die Ausgabe so weit verlangsamt werden, daß ein Lesen in Ruhe erfolgen kann.

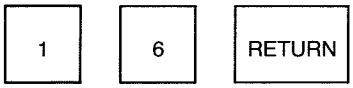
6. Die RETURN-Taste

Diese Taste (Alpha -Tastenblock, 2. + 3. Zeile rechts, rot) beendet eine Eingabe in den Rechner (nicht zu verwechseln mit dem Bildschirm, der ist nur ein Teil des Rechners). Ihr Computer kann nämlich nicht ahnen, wann die Information (man sagt in Fachkreisen dazu DATEN; auch wir werden uns jetzt daran halten) vollständig sind. Wenn Sie z. B. eine 1 eingegeben haben, weiß der Rechner nicht, ob vielleicht noch eine 6 folgt. Dazu beenden Sie alle Dateneingaben mit der Taste RETURN:

Z.B.:



(Eingabe der Zahl 1).

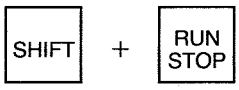


(Eingabe der Zahl 16).

7. Die RUN/STOP-Taste

Auch die RUN/STOP-Taste hat zwei Bedeutungen. Zum einen können Sie mit dieser Taste ein laufendes Programm an jeder beliebigen Stelle abbrechen. Das ist dann sinnvoll, wenn Sie in einem laufenden Programm gewisse Daten überprüfen wollen, die nicht von selber ausgegeben werden. Beispiele dafür geben wir Ihnen später.

Zum anderen können Sie auf einer Kassette gespeichertes Programm mit



in den Speicher des Computers einladen und automatisch starten.

1. DER COMPUTER UND SEINE SPRACHE

BASIC, die Sprache in der Ihr Computer arbeitet, ist eine leicht erlernbare, zum Teil aus englischen Worten, zum Teil aus mathematischen Zeichen bestehende Folge von Befehlen. Obwohl es eine der leichtesten Programmiersprachen ist, ist sie ungeheuer vielseitig und flexibel. Ursprünglich an der Dartmouth University entwickelt um Studenten die Möglichkeit zu geben, schnell und leicht Probleme an einem Computer zu bearbeiten, breitete sich diese Sprache sehr schnell in der ganzen Welt aus. Heute existiert BASIC in unzähligen Versionen auf fast jedem größeren Computer.

Bei unserem COMMODORE-BASIC stand auch das DARTMOUTH-BASIC Pate; gleichzeitig ist das COMMODORE-BASIC größer und leistungsfähiger geworden.

1.1. Die Komponenten der BASIC-Programmiersprache

Bevor wir uns mit der eigentlichen Sprache BASIC befassen, müssen wir uns zuvor über einige Schreibregeln einigen. Nachfolgend eine Liste der Konventionen und Ausdrücke, die wir in diesem Buch zu unserer und Ihrer Erleichterung einführen:

Eckige Klammern [ ]	– Ausdrücke in eckigen Klammern können, müssen aber nicht angegeben werden.
Zeichen	– Ein Buchstabe, eine Zahl oder ein beliebiges Sonderzeichen.
Konstante	– Eine Zahl innerhalb der Rechengrenzen Ihres Computers (siehe 1.1.1).
Formel	– Eine Konstante, eine Variable oder ein Ausdruck, bestehend aus Konstanten und /oder Variabalen (z. B. 8 + 3* 5/4).

Buchstabe	– Ein alphabetisches Zeichen von A – Z.
Zeilennummer	– Eine ganze Zahl von 0 bis 63999.
Text	– Eine Folge von 0 bis 255 beliebigen Zeichen.
Variable	– Siehe 1.1.2.
Integer	– Eine ganzzahlige Konstante oder Variable im Bereich von – 32768 bis 32767.
Byte	– Eine ganzzahlige Konstante oder Variable im Bereich von 0 bis 255.
Geschweifte Klammer { }	– Genau einer der in geschweiften Klammern angegebenen Ausdrücke muß benutzt werden.

1.1.1 Die Konstanten

Die Grundform einer Konstanten ist eine Dezimalzahl zwischen $1.7*10^{38}$ und $2.93*10^{-39}$.

Bis zu 9 Stellen einer Zahl können gespeichert werden und es kann damit gerechnet werden.

Beispiele für Konstanten:

25, 15.75, –87, 0, 3255, 45E7,14.2E–6, 13.12345678

In zwei der Konstanten tauchte der Buchstabe E auf. Der Buchstabe E besagt, daß die nachfolgenden Zahlen den Exponenten zur Basis 10 darstellen.

1.5E3 bedeutet $1.5*10^3 = 1.5*10*10*10 = 1.5*1000 = 1500$
1.27E–2 bedeutet $1.27*10^{-2} = 1.27/10/10 = 1.27/100 = 0.0127$.

Das Vorzeichen wird nur bei negativen Zahlen geschrieben.

Konstanten werden verwendet, um feste Werte zu fixieren.

Da der Computer nicht nur numerische Werte verarbeiten kann, sondern auch Textverarbeitung zuläßt, gibt es genauso wie numerische Konstanten auch Textkonstanten:

"A", "PETER IST LIEB", "ICH BIN EIN CBM", "125 LAMPEN".

Die Textkonstanten erkennt man daran, daß sie in Anführungszeichen eingebettet sind. Die Anführungszeichen selber gehören jedoch nicht zur eigentlichen Textkonstante.

1.1.2. Die Variablen

Eine Variable ist ein Symbol, dessen Wert sich ändern kann. Der Name einer Variablen besteht aus

entweder einem Buchstaben	z. B.: A, G, Z
oder einem Buchstaben und einer Zahl	z. B.: A1, F6,V9
oder zwei Buchstaben	z. B.: AX, PE, UT, UX

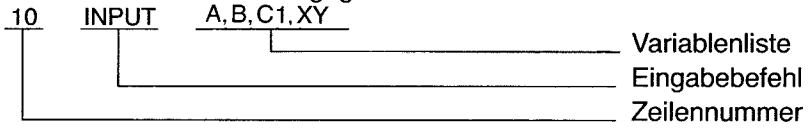
plus (wahlweise) der Kennzeichnung % oder $.

Eine Variable kann aus mehr als zwei Zeichen bestehen, es werden aber nicht mehr als zwei Zeichen abgefragt. Wenn Sie zum Beispiel zwei Variablen HANS und HABEL nennen, so werden diese wie eine Variable behandelt, da beide mit HA anfangen.

Die Variable kann innerhalb eines BASIC-Programms verschiedene Funktionen annehmen:

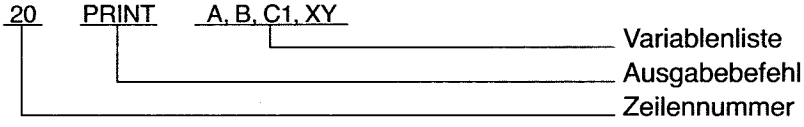
– Variablen repräsentieren numerische Werte
  z. B.: A=10 , A5=–5.6E–3 , XY=3.14

– Variablen können in den Rechner eingegeben werden
  z. B. 10 INPUT A,B,C1,XY



– Variablen können gleich gesetzt werden
  z. B. A = B, A5= XY, XY = B (A wird gleich B gesetzt, usw.)

Variablen können ausgedruckt werden
z. B. 20 PRINT A, B, C1, XY



Es gibt drei Arten von Variablen:

1. Fließkommavariablen

Diese Variablen können jeden beliebigen Wert innerhalb der Rechengrenzen des Computers annehmen.

Jede Variable, deren Name das Kennzeichen % oder $ nicht enthält, ist eine Fließkommavariable.

Beispiel: A = 7
B8 = 1.5E3
ZY = 9E9
X1 = –Ø.52187

2. Festkommavariablen

Diese Variablen können als Wert ganze Zahlen zwischen (einschließlich) –32767 und +32767 annehmen. Sie haben am Ende des Namens das Zeichen % stehen.

Beispiel: A% = 7
L1% = 4528
XØ% = Ø

3. Stringvariablen (Strings, Textvariablen)

Diese Variablen haben als Wert einen beliebigen Text aus Ø bis 255 Zeichen. Sie werden mit angehängtem $ gekennzeichnet.

Beispiel: A$ = "TEXT"
B$ = "55"
C$ = "DIES IST EIN STRING"
X1$ = "COMPUTER 3001"
N$ = " " (Bemerkung: dieser String besteht aus keinem Zeichen.)

Alle drei Variablenarten können noch indiziert werden. Man spricht dann von dimensionierten Variablen, Matrizen, Feldern oder Arrays. Siehe 1.2.18.

1.1.3. Arithmetische Operationen

Zu den arithmetischen Operationen gehören zunächst einmal die vier Grundrechenarten, die auf dem Computer eine etwas andere Darstellungsform besitzen als Sie es vielleicht von anderen Geräten oder Beschreibungen her kennen.

Nachfolgend die Darstellung dieser Zeichen:

Addition	+
Subraktion	–
Multiplikation	*
Division	/

Darüberhinaus existieren eine Vielzahl von mathematischen Funktionen, die standardmäßig vorhanden sind.

Betrachten wir diese Funktionen im einzelnen.

- SGN Die SIGNUM (Vorzeichen)-Funktion.

  Der Funktionswert ist:
  für negative Zahlen –1
  für positive Zahlen +1
  für Null Ø

Syntax: SGN ({ Variable, Konstante, arithm. Ausdruck })

Beispiel: A = SGN (B)
PRINT SGN (– 45)
PRINT SGN (A + π * 2)

Den Wert in der Klammer bezeichnet man als Argument.

Anstelle von PRINT kann auch das Fragezeichen gesetzt werden.

Anwendung:

Auf sehr einfache Weise bestimmen, ob eine Zahl negativ, positiv oder Ø ist.

- INT Die INTEGER-Funktion
  Sie bildet bei positiven Zahlen die kleinste ganze Zahl, d.h. alle Nachkommastellen werden zu Ø gesetzt.

Bei negativen Zahlen bildet sie die kleinste negative Zahl, d.h. alle Nachkommastellen werden bis zur nächsten ganzen Zahl abgerundet.

Syntax: INT ({ Variable, Konstante arithm. Ausdruck })

Beispiel: A = INT (B)
? INT (B) ? INT (– 2.5) ? INT (X – Y)

Anwendungsbeispiel:

Auf- oder Abrunden eines Ergebnisses, wenn nur eine ganzzahlige Größe erlaubt ist, z.B. bei der rechnerischen Ermittlung einer Zahl von Schrauben in einem Behälter.

- ABS Die ABSOLUTWERT-Funktion
  Sie schneidet das Vorzeichen ab und bildet nur den absoluten Wert.

Syntax: ABS ({ Variable, Konstante, arithm. Ausdruck })

Beispiel: A = ABS (B)
? ABS (– 145)

Anwendungsbeispiel:

Berechnung einer Wurzel aus einer unbekannten Zahl. Da es mathematisch nicht möglich ist, aus einer negativen Zahl eine Wurzel zu ziehen, kann man zuvor den Absolutwert bilden.

- SQR Die SQUARE ROOT (Quadratwurzel)-Funktion
  Sie bildet die Quadratwurzel aus dem Argument.

Syntax: SQR ({ Variable, Konstante, arithm. Ausdruck })

Beispiel: A = SQR (B) ? SQR (16)

Anwendung:

Im Bereich der Mathematik.

- SIN Die SINUS-Funktion
  Sie berechnet den Sinuswert des Arguments im Bogenmaß. (Die mathematische Bedeutung der trigonometrischen Funktionen wird als bekannt vorausgesetzt.)

Syntax: SIN ({ Variable, Konstante, arithm. Ausdruck })

Beispiel: A = SIN (B)
? SIN (30)

Anwendung:

Im Bereich der Mathematik.

Ist das Argument in GRAD gegeben, muß es durch eine Umrechnung modifiziert werden.

Beispiel: A = SIN (30 * π / 180)

Ist das Argument in NEUGRAD gegeben, so lautet die Umrechnung:

Beispiel: A = SIN (30 * π / 200)

- COS Die COSINUS-Funktion
  Sie wird genauso behandelt wie die SIN-Funktion.

- TAN Die TANGENS-Funktion
  Sie wird behandelt wie die SIN-Funktion.

Die COTANGENS-Funktion existiert bei Ihrem Rechner nicht, da sie sehr einfach durch die TAN-Funktion gebildet werden kann. Die Beziehung lautet:

$$\text{COT}(X) = \frac{1}{\text{TAN}(X)}$$

Beispiel: Y = 1 / TAN (X)

Von den Umkehrfunktionen zu den bisher behandelten trigonometrischen Funktionen existiert lediglich die

- ATN ARCUSTANGENS-Funktion.
  Sie bildet die Umkehr-Funktion des TANGENS.

Syntax: ATN ({ Variable, Konstante, arithm. Ausdruck })

Beispiel: Y = ATN (X)

Anwendung:
Im Bereich der Mathematik.

Die Umkehrfunktionen zum SIN und COS lassen sich leicht mit dem ATN berechnen.
Die Beziehungen lauten:
ARC SIN (X) = ATN (X / SQR (1 − X ↑ 2))
ARC COS (X) = ATN (SQR (1 − X ↑ 2) / X)

- LOG der NATÜRLICHE LOGARITHMUS ermittelt die Logarithmen zur BASIS e = 2,7182818

Syntax: LOG ({ Variable, Konstante, arithm. Ausdruck })
Beispiel: Y = LOG (X)

- EXP der EXPONENT zur natürlichen Basis e ermittelt den Wert der Funktion $y = e^x$

Syntax: EXP ({ Variable, Konstante, arithm. Ausdruck })
Beispiel: Y = EXP (X)

- ↑ die Potenzierung zu einer beliebigen Basis mit beliebigen Exponenten.

Syntax: {Variable, Konstante, arithm. Ausdruck} ↑ {Variable, Konstante, arithm. Ausdruck}
Beispiel: Y = A ↑ X
? 2 ↑ 3

- RND die RANDOMIZE (Zufalls)-Funktion
  Sie bildet Pseudo-Zufallszahlen zwischen Ø und 1.

Syntax: RND ({ Variable, Konstante, arithm. Ausdruck })
Beispiel: ? RND (X)

a) X negativ
   RND mit negativem Argument ergibt immer den gleichen Funktionswert. Dies kann beispielsweise für Fehlersuche oder während der Programmentwicklung nützlich sein.

b) X = Ø
   ergibt einen Wert, der nur von der Zeit seit dem Einschalten des Computers abhängt.

c) X positiv
   liefert von der Zeit und von X abhängige Zufallsvariablen.

Beispiel: Es soll ein Würfel simuliert werden.

```
10 W = RND (– TI)
20 W = INT (6 * RND (1) + 1)
30 ? W
```

- π die Zahl PI.
  Sie ist fest gespeichert. $\pi = 3.14159265$

1.1.4 OPERATOREN

zu den Operatoren zählen:

a) die Vergleichsoperatoren
b) die logischen Operatoren

Vergleichsoperatoren

Sie dienen dazu, Zahlenwerte zu vergleichen. Dies bildet einen sehr wichtigen Bestandteil bei der Behandlung und Untersuchung von Zahlen.

Wenn z. B. aus einer Zahlenfolge alle Zahlen ermittelt werden sollen, die einen bestimmten Grenzwert überschreiten, wird ein Vergleichsoperator angewendet.

Die Frage lautet:

– Welcher Wert ist größer als der Grenzwert, bzw.
– Welcher Wert ist kleiner oder gleich dem Grenzwert.

Die mathematische Darstellungsform dieser Vergleichsoperatoren zeigt nachfolgende Tabelle:

BASIC SCHREIBWEISE	BEDEUTUNG
<	kleiner als
>	größer als
= <	gleich oder kleiner als
= >	gleich oder größer als
<> oder ><	nicht gleich
=	Gleichheit

Einige Beispiele mögen die Anwendung und Bedeutung der Vergleichsoperatoren hervorheben. Dabei benutzen wir schon jetzt die im Rechner gebräuchliche Schreibweise.

```
IF  A < B    THEN ? A        wenn A kleiner als B, dann drucke A
IF  A > B    THEN ? B        ...
IF  A < > B  ...             (oder: IF A >< B)
IF  A >= B   ...             Wenn A größer gleich B, dann ...
IF  A <= B   ...             ...
IF  A = B    THEN ? "A = B"  wenn A gleich B, dann drucke: A = B
```

LOGISCHE OPERATOREN

Sie sind der Booleschen Algebra entlehnt und finden unter anderem in der elektronischen Digitaltechnik (also die Technik, in der der Rechner arbeitet) Anwendung.

In den Programmiersprachen haben sie ebenfalls eine sehr große Bedeutung. Man kann mit Ihnen komplizierte Abfragen aufbauen.

Der Rechner kennt drei logische Operatoren, nämlich

AND (und – Bedingung)
OR (oder – Bedingung)
NOT (nicht – Bedingung)

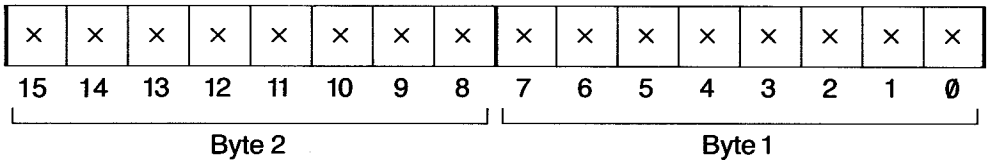
Diese Operatoren können angewendet werden auf beliebige Variablen.

1. AND, OR und NOT bei Festkommavariablen:

Bei ganzen Zahlen zwischen −32767 und 32767 arbeiten diese Funktionen bitweise in der folgenden Art:

X	Y	X AND Y	X OR Y	NOT X
Ø	Ø	Ø	Ø	1
Ø	1	Ø	1	1
1	Ø	Ø	1	Ø
1	1	1	1	Ø

Alle Festkommazahlen (Integerzahlen) sind im Computer binär verschlüsselt. Sie bestehen aus 16 Bits, die jeweils den Zustand Ø oder 1 annehmen können.



Daraus folgt, daß $2^{16} = 65536$ verschiedene Integerzahlen darstellbar sind.

Da wir aber bei ganzen Zahlen auch negative Zahlen darstellen wollen, wird Bit 15 als Vorzeichen benutzt. Wenn dieses Bit Null ist, so ist die Zahl positiv, ist Bit 15 gleich Eins, so ist die Zahl negativ.

Es bleiben damit $2^{15} - 1 = 32767$ verschiedene Möglichkeiten, positiv oder negativ.

Positive Zahlen sind in der Weise codiert, daß Bit n der Wert 2^n zugewiesen wird, wenn dieses Bit gleich 1 ist. Ansonsten hat Bit n den Wert Ø. Die Werte aller Bits werden addiert, um die Zahl zu erhalten.

Die Bitdarstellung einer negativen Zahl erhält man folgendermaßen:

Man bildet die entsprechende positive Zahl in Binärdarstellung invertiert dann alle Bits und addiert 1.

Beispiel:

Integer	Binär
− 1	1111 1111 1111 1111
− 2	1111 1111 1111 1110
− 100	1111 1111 1001 1100
− 119	1111 1111 1000 1001
− 127	1111 1111 1000 0001

AND, OR und NOT bilden aus den Argumenten die neuen Werte durch bitweises Vorgehen entsprechend der Wahrheitstabelle.

Beispiel: ? 5000 AND −127 ergibt 4992
? 5000 OR −127 ergibt − 119
? NOT 5000 ergibt −5001

wegen

```
00010011  10001000   ( 5000)
11111111  10000001   (−127)
--------------------
00010011  10000000   5000 AND −127
11111111  10001001   5000 OR −127
11101100  01110111   NOT 5000
```

2. „Wahr" und „Falsch"

Jeder Aussage wird vom Computer ein Wert zugeordnet, und zwar der Wert −1 (alle Bits = 1), wenn die Aussage trifft, und der Wert Ø (alle Bits = Ø), wenn die Aussage nicht zutrifft.

	Wert, wenn Bit = 1	Wert, wenn Bit = Ø
Bit 15	negativ	positiv
Bit 14	16384	Ø
13	8192	Ø
12	4096	Ø
11	2048	Ø
10	1024	Ø
9	512	Ø
8	256	Ø
7	128	Ø
6	64	Ø
5	32	Ø
4	16	Ø
3	8	Ø
2	4	Ø
1	2	Ø
Ø	1	Ø

Beispiel:

Integer	Binär			
Ø	ØØØØ	ØØØØ	ØØØØ	ØØØØ
1	ØØØØ	ØØØØ	ØØØØ	ØØØ1
100	ØØØØ	ØØØØ	Ø11Ø	Ø1ØØ
1000	ØØØØ	ØØ11	111Ø	1ØØØ
10000	ØØ1Ø	Ø111	ØØØ1	ØØØØ
32767	Ø111	1111	1111	1111
5000	ØØØ1	ØØ11	1ØØØ	1ØØØ

Beispiel: A$ = "AMEN" : ? A$ = "AMEN" : ? A$ = "AMENDE"

Hier wird der Variable A$ der String AMEN zugeordnet, dann soll der „Wert" zweier Aussagen gedruckt werden. Da die erste Aussage „wahr" und die zweite Aussage „falsch" ist, erhält man als Antwort: −1 und Ø.

Beispiel: ? A = NOTB
? X > Y
? X$ = A$ + "X"
usw.

3. AND, OR und NOT bei Aussagen

Die AND-Funktion ist dann wahr, wenn alle Einzelbedingungen wahr sind.

Beispiel: IF A = 1 AND B = 2 AND C = 3 THEN PRINT "FERTIG"

Der Computer wird das Wort FERTIG nur dann ausgeben, wenn die Variablen A, B und C die Werte 1, 2 bzw. 3 haben.

Beispiel: IF A$ = "J" THEN ? "JA"

Die OR-Funktion ist dann wahr, wenn mindestens eine der Einzelbedingungen wahr ist.

Beispiel: IF A$ = "J" OR A$ = "Y" THEN ? "JA"

Der Computer wird nur dann JA ausgeben, wenn A$ vorher den String "J" oder den String "Y" zugewiesen erhielt.

Die NOT Funktion kann nicht auf Strings angewendet werden, sondern nur auf Festkommavariablen, ist das Argument eine Dezimalzahl, so werden die Nachkommastellen nicht beachtet. Liegt das Argument nicht innerhalb der Grenzen für eine Integerzahl, so erhält man eine entsprechende Fehlermeldung (? ILLEGAL QUANTITY ERROR).

Beispiel: A = −3 : B = −3.54 : ? NOTA, NOTB
ergibt: 2 2
A = −5 : ? NOTA, NOT NOTA
ergibt: 4 −5
A = 123456 : ? NOTA
ergibt: ? ILLEGAL QUANTITY ERROR (Argument zu groß)

BEISPIELE FÜR BOOLE'SCHE OPERATIONEN

63 AND 16 = 16 — Da die binäre Schreibform von 63 111111, und von 16 10000 ist, ergibt sich nach der AND-Verknüpfung 10000 oder 16.

15 AND 14 = 14	15 ≙	1111
	14 ≙	1110
	Resultat	1110 ≙ 14
−1 AND 8 = 8	−1 ≙	1111111111111111
	8 ≙	0000000000001000
	Resultat	0000000000001000 ≙ 8
10 OR 10 = 10	10 ≙	1010
	10 ≙	1010
	Resultat	1010 ≙ 10
−1 OR −2 = −1	−1 ≙	1111111111111111
	−2 ≙	1111111111111110
	Resultat	1111111111111111 ≙ −1
NOT 0 = −1	0 ≙	0000000000000000
	Komplement davon	1111111111111111 ≙ −1

1.1.5 PRIORITÄTEN bei Formeln

Da in der Rechnerschreibweise Formeln und Anweisungen in einer Zeile geschrieben werden, muß der Rechner ein Schema besitzen, diese Formeln der Reihe nach abzuarbeiten.

Diese PRIORITÄT zur Abarbeitung mathematischer Formeln kann man sehr einfach darstellen. Oberste Priorität haben immer die mathematischen Operatoren, gefolgt von den Vergleichsoperatoren und den logischen Operatoren.

Im einzelnen läßt sich das wie folgt darstellen:

Mathem. Operatoren — **Höchste Priorität**

Funktionen

↑

* /

+ −

VERGLEICHSOPERATOREN

=

<> (oder ><)

>

<

> =

< =

LOGISCHE OPERATOREN

NOT

AND

OR — Niedrigste Priorität

Beispiel: ? 2 ↑ 3 + 6 ergibt 14, weil 2 ↑ 3 zuerst berechnet wird.

1.1.6 Besonderheiten des Gleichheitszeichens

Das Gleichheitszeichen hat in der EDV-Terminologie eine von der Algebra abweichende Bedeutung. Es bedeutet:

„Ist der Wert von"

und ist nicht zu verwechseln mit der algebraischen Bedeutung:

„Ist gleich"

Die Rechenanweisung

X = X + 1

wäre in der Algebra falsch, da sie auf den Widerspruch $\emptyset = 1$ führt.

In der Programmiersprache BASIC ist diese Anweisung jedoch korrekt, sie besagt nämlich:

X „ist der Wert von" alter Wert von X plus 1

oder

X = X * B — X „ist der Wert von" alter Wert von X mal dem Wert von B.

1.1.7 MATHEMATISCHE FORMELN ALS BEISPIEL FÜR RECHENANWEISUNGEN

Mathematische Formeln werden in BASIC etwas anders geschrieben als in der herkömmlichen Schreibweise.

Den besten Vergleich erhält man, wenn man diese unterschiedlichen Schreibweisen anhand von Beispielen einander gegenüber stellt.

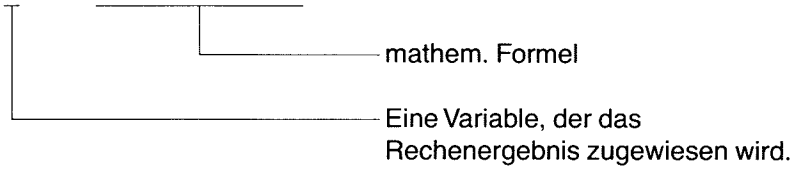
ALGEBRAISCH	BASIC
$\frac{a+b}{c+d}$	(A + B) / (C + D)
$\frac{a+b}{c}$	(A + B) / C
$\frac{a \cdot b}{c}$	A * B / C
$\frac{\frac{a}{b}}{c}$	A / B / C

$\frac{a}{\frac{b}{c}}$	A / (B / C)
$\frac{a}{b \cdot c}$	A / (B * C)
$(ab)^N C$	(A * B) ↑ N * C
b^{a+1}	B ↑ (A + 1)
$b^a + 1$	B ↑ A + 1
$\sqrt[5]{a^3}$	A ↑ Ø.6 $[\frac{3}{5} = Ø.6]$
$\frac{1}{\sqrt[2]{a^1}}$	A ↑ (−Ø.5) $[\frac{1}{2} = Ø.5]$

Die Rechenanweisung wird nun gebildet durch das Gleichheitszeichen, auf dessen Besonderheit bereits hingewiesen wurde.

Die Darstellung einer Rechenanweisung erfolgt genauso wie in der Algebra.

z.B. X = A + B + C



Diese Darstellungsform muß immer eingehalten werden und darf nicht verändert werden. So ist z. B. vorgeschrieben, daß links vom Gleichheitszeichen immer eine Variable stehen muß, während rechts entweder eine Variable **oder** eine mathematische Formel stehen muß. Es darf links kein mathematischer Ausdruck stehen.

1.1.8 Der Aufbau eines Programms in BASIC

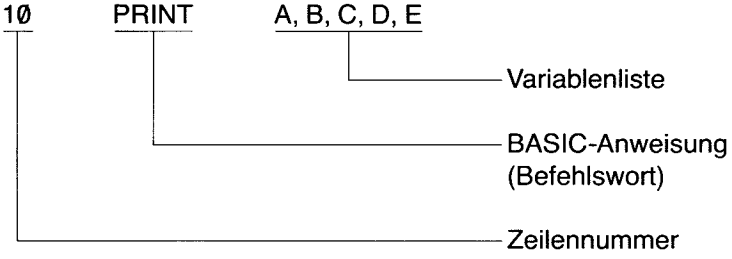
Um ein Programm in BASIC zu erstellen, benutzt man die nachfolgend beschriebenen BASIC-Anweisungen.

Auch hierbei muß man sich an eine bestimmte Ordnung halten, ähnlich wie bei den Rechenanweisungen.

Eine BASIC-Programmzeile besteht immer aus einer bestimmten Zeilennummer, die man auch als Adresse bezeichnen könnte, der BASIC-Anweisung selbst, sowie (sofern vorhanden) der Variablenliste. Die Programmzeile kann maximal 8Ø Zeichen lang werden.

Eine einfache Anweisung, nämlich die Werte von 5 Variablen auszudrucken, sieht recht übersichtlich aus.

1Ø PRINT A, B, C, D, E



Die Zeilennummer ist sozusagen der Fingerabdruck der BASIC-Anweisung. Sie gibt es nur einmal und ist daher unverwechselbar. Anhand dieser Zeilennummer finden wir die Anweisung immer wieder.

Sie ist immer ganzzahlig, in aufsteigender Reihenfolge und kann von

Ø bis 63999

definiert sein.

Die Anweisung selbst sagt aus, was mit der nachfolgenden Variablenliste, die im Übrigen aus verschiedenen Informationen bestehen kann, passieren soll.

Möglich sind Texte, Konstanten und Variablen.

Das normale Standard-BASIC sieht vor, daß mit diesen Informationen eine Programmzeile beendet ist. Eine neue Zeile sollte beginnen.

Bei Ihrem Computer ist es jedoch möglich, mehrere BASIC-Anweisungen getrennt durch einen Doppelpunkt, in eine Zeile zu schreiben. Maximal kann diese Zeile 80 Zeichen aufnehmen. Bedingt durch die Begrenzung von 40 Zeichen pro Zeile auf dem Bildschirm, bildet sich der Rechner automatisch eine Doppelzeile.

Beispiel: 10 PRINT A, B, C, D, E: PRINT Y: GOTO 100

1.2.1 Der REM Befehl

Wichtig ohne etwas zu tun . . .

Sobald die Probleme, die Sie mit Ihrem Rechner lösen, komplexer werden, wird Ihr Programm natürlich länger und damit meist unübersichtlicher. Ein gutes Hilfsmittel, um ein Programm gut überschaubar zu machen, ist der REMARK-Befehl:

n REM TEXT

n = Zeilennummer des Befehls
Text – beliebige Informationen

Die REM-Anweisung ist eine Mitteilung an den Rechner, die nachfolgenden Ausführungen, Anweisungen oder Bemerkungen beim Rechenvorgang nicht zu berücksichtigen. Sie hat auf den Programmablauf keinen Einfluß, sondern wird nur beim Auflisten des Programms ausgegeben, um dem Programmierer die Arbeit zu erleichtern. Zu beachten ist, daß die REM-Anweisung Speicherplatz benötigt und der nachfolgende Text nicht länger ist, als es die Programmzeile erlaubt (insgesamt 80 Zeichen).

Wird REM gemeinsam mit anderen Befehlen in einer Zeile verwendet, so muß REM der letzte Befehl der Zeile sein, da Befehle nach REM nicht ausgeführt werden.

Beispiele:

```
100 REM        Beginn des Unterprogramms
5810 REM       Schachprogramm
  12 PRINT A:  REM ERGEBNIS
```

1.2.2 Der END Befehl

Jetzt hört's auf . . .

Auch wenn das Programmieren noch soviel Spaß macht, einmal ist jedes Programm zu Ende. Um das logische Ende des Programms zu kennzeichnen, benutzen wir den Befehl:

n END

n = Zeilennummer des Befehls

Der Befehl END kann mehrmals in einem Programm benutzt werden, was vor allem bei Programmverzweigungen sinnvoll sein kann. Dieser Befehl bewirkt das Anhalten des Programms, ohne irgendwelche Daten zu verändern. Das Programm kann nach dem END durch den Befehl CONT (siehe 1.3.2) fortgesetzt werden.

Beispiel: 10 END

1.2.3 Der STOP Befehl

Das Programm kann verschnaufen . . .

Es soll ja vorkommen, daß ein Programm nicht gleich so arbeitet wie sein Schöpfer gern hätte. Dann sind oft einige Testläufe nötig. Bei diesen Tests kann man den Befehl

n STOP

n = Zeilennummer des Befehls

benutzen um das Programm anzuhalten und den Wert einiger Variablen zu erfragen. Der STOP-Befehl unterscheidet sich vom END-Befehl dadurch, daß beim Beenden des Programms die Meldung: BREAK IN . . . (Zeilennummer) ausgegeben wird.

Auch nach dieser Unterbrechung kann das Programm mit CONT (siehe 1.3.2) fortgesetzt werden.

Beispiel: 12Ø STOP

1.2.4 Der LET Befehl

Man kann ihn auch vergessen . . .

Bei Ihrem Rechner ist es genauso wie bei uns Menschen, seine Sprache ist älter als er. Bei älteren Rechnertypen war der Befehl

n LET Zuweisung	n = Zeilennummer des Befehls

nötig um Werte einer Variablen zuzuordnen. Unseren Rechner stört diese Anweisung nicht, sie ist aber auch nicht nötig und man sollte sie möglichst fortlassen, da sie 1 Byte unseres Speicherplatzes belegt.

Beispiel: 13Ø LET A = 1Ø * B

besser 13Ø A = 1Ø * B

1.2.5 Der PRINT Befehl

So unterhalte ich mich . . .

Soll der Rechner seinen Benutzer während des Programmlaufs ansprechen, ihm Daten angeben oder einen guten Morgen wünschen, benutzen wir den Befehl:

n { PRINT / ? } { Text / Variablenliste }

n = Zeilennummer des Befehls

Der PRINT-Befehl bewirkt das Ausgehen einer nachfolgenenden Text- und/oder Variablenliste.

Das Fragezeichen ist gleichbedeutend mit dem Wort PRINT und wird vom Rechner beim Listen des Programms durch PRINT ersetzt. Auszugebender Text muß in Anführungszeichen gesetzt werden.

Beispiele:
```
11Ø PRINT "LISTE-NR.", A ; D $
2Ø ? "PROGRAMMANFANG", C $
```

Nach einem PRINT-Befehl erfolgt ein Wagenrücklauf und ein Zeilenvorschub, so daß jeder PRINT-Befehl am Anfang einer neuen Zeile ausgeführt wird. Mit dem bloßen PRINT-Befehl erzeugt man also eine Leerzeile auf dem Ausgabegerät.

Der Computer läßt verschiedene Formatierungsarten zu; dabei ist zu beachten, daß ein String genausoviel an Platz belegt, wie er Zeichen hat. Eine Zahl belegt soviele Plätze, wie sie Ziffern hat, plus eine Spalte für das Vorzeichen.

Das Komma:

Jede zehnte Druckposition ist intern im Computer vortabuliert. Das Komma in einem PRINT-Befehl bewirkt, daß das folgende Zeichen in die nächste vortabulierte Position geschrieben wird. Zeilenvorschub und Wagenrücklauf werden unterdrückt.

Beispiel: ? , "XY", "XY"

```
          XY          XY
----------|----------|----------
↑         ↑          ↑
1         11         21
```

Beispiel:
```
A = 3 : B = - 4 : ? A , B
 3                - 4
 ↑                ↑
 2               11
```

Das Semikolon:

Das Semikolon in einem PRINT-Befehl bewirkt, daß Zeilenvorschub und Wagenrücklauf unterdrückt werden (Ausgabe ohne Zwischenraum).

Beispiel:
```
? "COM"; : ? "PUTER"
COMPUTER
```

Beispiel:
```
A = -3 : B = -4 : ? A; B; "TEXT"; A
 3  -4  TEXT 3
```

Der SPC-Befehl:

Der SPC(n) bewirkt den Ausdruck von n Leerzeilen. n kann zwischen Ø und 255 liegen.

n kann auch eine Variable oder ein berechneter Ausdruck sein.

Nachkommastellen bleiben unbeachtet.

Beispiel:
```
? SPC(3) "DREI" SPC(1) "FUENF"
   DREI FUENF
```

Beispiel:
```
? SPC(π) "DREI"
   DREI
```

Siehe Anmerkung zum TAB-Befehl.

Der TAB-Befehl:

Er hat das Format TAB(n), wobei n eine Zahl oder ein berechneter Ausdruck zwischen Ø und 255 sein kann. Stellt n keine ganze Zahl dar, so werden die Nachkommastellen nicht beachtet.

Der Befehl TAB(n) bewirkt den Ausdruck des folgenden Zeichens n Freistellen vom linken Bildschirmrand (oder Papierrand) entfernt.

Beispiel:
```
? TAB (4) 4; TAB (8) "X"
     4   X
```

Ist die durch den TAB-Befehl spezifizierte Druckposition bereits belegt, so wird in die nächste freie Position geschrieben.

Beispiel:
```
? TAB(4) "X" TAB(3) "Y"
    XY
```

Anmerkung: Bei der Verwendung von TAB und SPC dürfen die Daten nur durch Semikolon getrennt werden, da das Komma eine höhere Priorität besitzt als die Befehle TAB und SPC.

1.2.6. Der INPUT-Befehl

Und so verstehe ich Sie . . .

Um die Dialogfähigkeit des cbm zu verwirklichen, reicht der PRINT-Befehl nicht, wir benötigen auch ein offenes Ohr, den INPUT-Befehl:

n INPUT ["TEXT" ;] Variablenliste

(n = Zeilennummer des Befehls)

Der INPUT-Befehl bewirkt einen Stop des Programms und ermöglicht eine Dateneingabe für die Variablen, die hinter dem INPUT aufgeführt werden.

Stehen mehrere Variablen hinter dem INPUT, so müssen entweder alle Daten durch Komma getrennt werden und abschließend muß die RETURN-Taste gedrückt werden oder nach jeder einzelnen Eingabe muß die RETURN-Taste betätigt werden.

Beispiel: 20 INPUT A, C $, D

Eingabe	: 2, KARO, 3	RETURN
oder	: 2	RETURN
	KARO	RETURN
	3	RETURN

Das Programm zeigt die Bereitschaft zur Eingabe durch ein Fragezeichen oder durch Drucken des Textes, den wir hinter dem INPUT in Anführungszeichen vermerkt haben, plus Fragezeichen an.

Beispiel: 30 INPUT A
Ausdruck: ?

40 INPUT "GEBEN SIE A EIN"; A
Ausdruck: GEBEN SIE A EIN?

Als Variable können selbstverständlich auch Ganzzahl- und Textvariablen verwendet werden.

Beispiel: 10 INPUT "TEXTEINGABE"; B $

Geben Sie mehr als 80 Zeichen bei einem INPUT-Befehl ein, so ignoriert der Computer die ersten 80 Zeichen und speichert nur die nachfolgenden Zeichen ab.

Weiter darf während der Eingabe für eine numerische Variable kein Komma oder Doppelpunkt benutzt werden, da der Computer diese Zeichen als Eingabeende interpretiert.

Soll ein String eingegeben werden, der Komma oder Doppelpunkt enthält, so muß dieser String in Anführungszeichen geschrieben werden.

1.2.7 Der GET Befehl

Für Leute die nicht mehr warten wollen . . .

Der | n GET Variable | n = Zeilennummer des Befehls

Befehl ist ebenfalls ein Eingabebefehl.

Im Gegensatz zum INPUT hält das Programm bei einem GET nicht an und es kann nur eine Variable eingegeben werden. Es können Variablen oder Textvariablen eingegeben werden. Es wird nur ein Zeichen bzw. eine Ziffer übernommen.

Beispiel: 10 GET C
30 GET B $

Da der Rechner in der Lage ist, bis zu 9 Tastenbetätigungen zu speichern, wird dasjenige Zeichen übernommen, das während des Programmlaufes vor Erreichen des GET-Befehls gedrückt wurde.

Soll der Rechner auf Eingabe eines Zeichens warten, so kann das mit einer Warteschleife erreicht werden. (Siehe 1.2.15).

Beispiel:

```
    ⋮
20 GET A $
30 IF A $ = " " THEN 20
    ⋮
```

1.2.8 Der DATA Befehl

Mein Grundwissen . . .

Nun lernen wir einen Befehl kennen, mit dem man Daten speichern kann, auf die der Rechner im Bedarfsfalle zurückgreift. Es ist der Befehl:

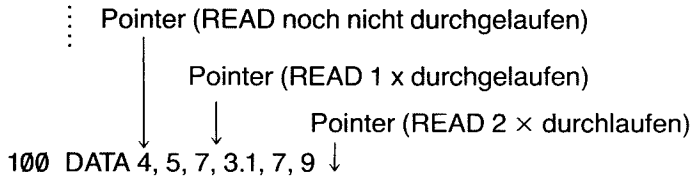
n DATA Liste	n = Zeilennummer des Befehls

Die Daten werden hierbei an beliebiger Stelle im Programm hinter DATA aufgereiht und später der Reihe nach von links nach rechts mittels READ gelesen. (siehe 1.2.9) Die Daten werden untereinander durch Kommas getrennt und dürfen daher weder Komma noch Doppelpunkt enthalten. Sollen diese beiden Zeichen oder führende Leerzeichen mittels DATA in eine Textvariable gebracht werden, so muß dieser Text in der DATA-Anweisung in Anführungszeichen stehen, was ansonsten bei Texten hinter DATA nicht der Fall ist.

Beispiel: 100 DATA 15, 3.7, 9, 12
150 DATA "DER:", COMPUTER

Die Daten hinter einem DATA können von verschiedenen READ-Anweisungen gelesen werden, da sich ein Pointer (Zeiger) „merkt", ab welchem Wert mit dem READ Befehl weiter gelesen wird.

Beispiel: 10 READ A, B, C



1.2.9 Falls ich es vergessen habe . . . . .

DER RESTORE BEFEHL

Beim DATA-Befehl haben wir erfahren, daß der Computer die Daten hinter dem DATA der Reihe nach liest. Sollen, nachdem bereits Daten gelesen wurden, nicht die nachfolgenden, sondern die ersten Daten gelesen werden, so benutzen wir den

n RESTORE	n = Zeilennummer des Befehls

Befehl.

Mit anderen Worten, dieser Befehl setzt den Pointer vor das 1. Zeichen der Daten.

Beispiel: 30 RESTORE

1.2.10 . . . . und so komme ich daran . . . .

DER READ BEFEHL

Der

n READ Variablenliste	n = Zeilennummer des Befehls

Befehl bezieht sich auf die DATA-Anweisung und kommt ohne diese nicht aus. Der READ-Befehl liest die Daten in der vorgegebenen Reihenfolge und ordnet sie den Variablen, die hinter ihm aufgeführt sind, in der gleichen Reihenfolge zu.

Beispiel: 10 READ A, B
20 DATA 102, 3, 5, 6
Wird der READ-Befehl zum ersten Mal durchlaufen so ist A = 102, B = 3;
beim zweiten Durchlauf von READ ist A = 5, B = 6.

Natürlich können auch Textvariablen gelesen werden. Falls der Pointer hinter dem letzten Zeichen der DATA-Anweisung steht und ein READ erfolgt, erscheint die Fehlermeldung:

?OUT OF DATA ERROR IN . . .

(Sie wissen natürlich schon, daß man diese Fehlermeldung mit RESTORE (siehe 1.2.9) umgehen kann).

1.2.11 Ich kann auch springen . . . . .

DER GOTO BEFEHL

Dieser Sprungbefehl wird als unbedingter Sprungbefehl bezeichnet, da er immer ausgeführt wird, sobald das Programm auf einen derartigen Befehl stößt, oder sobald ein GOTO direkt eingegeben wird.

n GOTO m

n = Zeilennummer in der der Befehl steht
m = Zeilennummer in die gesprungen wird

Dieses GOTO bewirkt, daß nicht der Befehl mit der nach n folgenden Zeilennummer bearbeitet wird, sondern daß das Programm seine Arbeit in der Zeilennummer m fortsetzt.

Beispiel:
```
10 PRINT "ANFANG"
20 GOTO 150
30 PRINT "ENDE"
      :

150 PRINT "SPRUNG"
      :
```

Eingabe von RUN (siehe 1.3.1) bewirkt folgende Ausgabe:
ANFANG
SPRUNG

Die Zeilennummer m muß vorhanden sein, sonst erfolgt eine Fehlermeldung. Variablen oder andere Zeichen als ganzzahlige positive Zahlen dürfen nicht benutzt werden.

Stehen in einer Programmzeile weitere Befehle hinter einer GOTO-Anweisung, so werden sie nicht ausgeführt.

1.2.12 Gewußt wohin . . . . .

DER ON . . . GOTO BEFEHL

Hatte das Programm beim GOTO nur eine Möglichkeit zu springen, so kann beim ON . . . GOTO Befehl der Sprung vom Wert einer Variablen abhängig gemacht werden.

```
n  ON  v  GOTO  m1, m2, . . . .
```

n = Zeilennummer des Befehls
v = Formel, die den Sprung bestimmt
m1, m2 = Zeilennummern zu denen gesprungen werden soll

Beispiel: 10 ON F GOTO 50, 100, 300
Ist F < 1 erfolgt kein Sprung
Ist F = 1 erfolgt Sprung nach 50
Ist F = 2 erfolgt Sprung nach 100
Ist F = 3 erfolgt Sprung nach 300
Ist F > 3 erfolgt kein Sprung

Anstelle der Variablen kann auch ein mathematischer Ausdruck stehen. Das Ergebnis des errechneten Ausdrucks wird vom Rechner zu einer ganzzahligen Größe umgewandelt, d.h. Nachkommastellen werden ignoriert.

Beispiel: 50 ON (A+B) – (C+D) GOTO 1000, 300, 700

Die Anzahl der Sprungadressen ist durch die maximale Zahl von 80 Zeichen pro Zeile begrenzt. Als Alternative bei sehr umfangreichen ON . . . GOTO-Befehlen bieten sich daher mehrere ON . . . GOTO's an, bei denen die Variable im zweiten Befehl dem Programm entsprechend berechnet wird.

Beispiel: 10 ON K GOTO 100, 200, 300, 400
30 ON K-4 GOTO 500, 600, 700, 800

1.2.13 Jetzt geht es in die Unterwelt . . . . .

DER GOSUB BEFEHL

Der folgende Befehl sollte für die Programmteile benutzt werden, die wiederholt, auch von verschiedenen Stellen des Programms aus, durchlaufen werden müssen. Diese Programmteile nennt man Unterprogramm oder auch Subroutine. Sie können an beliebiger Stelle im Programm stehen und werden mit dem Befehl

```
n  GOSUB  m
```

n = Zeilennummer des Befehls
m = 1. Zeilennummer des Unterprogramms

aufgerufen.

Nach Bearbeitung des Unterprogramms kehrt der Rechner zu dem Befehl zurück, der hinter dem GOSUB folgt. (siehe RETURN 1.2.16).

Beispiel:
```
10 GOSUB 100
20 PRINT "HAUPTPROGRAMM"
50 END: REM ENDE DES HAUPTPROGRAMMS
100 PRINT "UNTERPROGRAMM"
110 RETURN: REM ENDE DES UNTERPROGRAMMS
```

Die Eingabe von RUN bewirkt die
Ausgabe: UNTERPROGRAMM
HAUPTPROGRAMM

Ein Unterprogramm kann wiederum ein anderes Unterprogramm aufrufen, und dieses dann das nächste . . . . . Insgesamt können bis zu 26 Unterprogramme ineinander verschachtelt werden.

1.2.14 Verteiler für die Unterwelt . . . . .

DER ON . . . GOSUB BEFEHL

Der Befehl

```
n  ON  v  GOSUB  m1, m2 . . .
```

n = Zeilennummer des Befehls
v = Formel, die den Sprung bestimmt
m1, m2 . . . = Zeilennummern, zu denen gesprungen werden soll

entspricht dem ON . . . GOTO (siehe 1.2.12), mit dem Unterschied, daß hier keine einfachen Sprünge stattfinden, sondern zu Unterprogrammen verzweigt wird (siehe 1.2.13).

Beispiel: 10 ON Z GOSUB 1000, 1500, 2000
20 STOP

Der Sprung erfolgt abhängig von der Variablen Z in eines der 3 Unterprogramme, die natürlich mit RETURN abgeschlossen werden müssen. Nach Abarbeitung des betreffenden Unterprogramms wird die Zeile 20 ausgeführt.

1.2.15 Nun fällt die Entscheidung . . .

DER IF . . . THEN BEFEHL

Um abhängig von einer logischen oder mathematischen Aufgabe eine Entscheidung zu fällen, gibt es die Befehlsgruppe IF . . . THEN.

```
n  IF Vergleich THEN {m, Befehl}
```

n = Zeilennummer des Befehls
Vergleich = Vergleich einer oder mehrerer Variablen oder Formeln
m = Zeilennummer deren Befehl ausgeführt werden soll
Befehl = Befehl der ausgeführt werden soll

Beim Vergleich können Variablen (Auch Textvariablen), Konstanten oder Formeln untereinander verglichen werden.

Beispiel: IF A = 1 . . . / IF A$ = "JA" . . . / IF CC = 3 . . .
IF 3> B * C + 3 . . . / IF A <> C . . . /
IF D% > E % . . .

Ist die Vergleichsbedingung erfüllt, so wird der Befehl hinter dem THEN ausgeführt oder nach der dort stehenden Zeilennummer verzweigt, ansonsten fährt das Programm mit der nächsten Zeilennummer fort.

Beispiel: 10 IF A + B < = 150 THEN 30
20 PRINT "A + B > 150": GOTO 100
30 PRINT "A + B < = 150": GOTO 100

eleganter: 10 IF A + B < = 150 THEN PRINT "A + B < = 150": GOTO 100
20 PRINT "A + B > 150": GOTO 100

Wie in diesem Beispiel das PRINT, so können auch andere Befehle hinter dem THEN stehen. Als Besonderheit läßt das COMMODORE-BASIC mehrere IF-Abfragen in einer Zeile zu.

Beispiel: 10 IF A = C THEN IF A = 5 * B THEN . . . .

Ist hierbei bereits die erste Abfrage nicht erfüllt, wird sofort die nächst höhere Programmzeile ausgeführt.

1.2.16 Zurück geht's immer leichter . . . .

DER RETURN BEFEHL

Wir haben mit GOSUB, ON . . . GOSUB die Möglichkeit kennengelernt ein Unterprogramm aufzurufen. Um nun wieder ins Hauptprogramm zurückzukehren, muß der letzte Befehl jedes Unterprogramms:

n RETURN	n = Zeilennummer des Befehls

sein.

Diese Anweisung bewirkt, daß der dem Aufrufbefehl (GOSUB oder ON . . . GOSUB) folgende Befehl als nächster ausgeführt wird (siehe 1.2.13, 1.2.14).

Der Befehl RETURN muß Buchstabe für Buchstabe eingegeben werden und darf nicht mit der Taste RETURN verwechselt werden.

Beispiel:
```
40 A = 5 : B = 10
50 GOSUB 1000
60 PRINT "ERGEBNIS" U
      ⋮
1000 REM UNTERPROGRAMM
1010 U = (A+B) * 3
1020 RETURN
READY
RUN
ERGEBNIS 45
```

Solange die Unterprogrammtechnik für uns noch ungewohnt ist, werden wir vielleicht die Fehlermeldung: ?RETURN WITHOUT GOSUB ERROR erhalten. Sie tritt auf, falls ein RETURN erreicht wird, ohne daß ein GOSUB durchlaufen wurde. Dieser Fehler kann auftreten, wenn in ein Unterprogramm mit einer GOTO-Anweisung gesprungen wurde. Auch dann, wenn ein Unterprogramm ans Ende des Hauptprogramms geschrieben wird und am Ende des Hauptprogramms kein STOP oder END steht.

Beispiel:
```
10 REM    HAUPTPROGRAMM
             ⋮
50 GOSUB  1000
             ⋮
1000 REM  UNTERPROGRAMM
             ⋮
1100 RETURN
```

Bei diesem Beispiel durchläuft das Programm die Subroutine nicht nur nach dem Befehl GOSUB 1000 sondern auch nach Ende des Hauptprogramms. Der Befehl:

990 END (etwa 10 Leerzeichen) verhindert diesen Fehler.

1.2.17 Immer wieder das Gleiche . . . . .

DIE BEFEHLSGRUPPE – FOR NEXT STEP

Wollen Sie eine Schleife programmieren, d.h. eine Routine, die mehrmals mit einer veränderten Variablen durchlaufen wird, so benutzen Sie die Befehlsgruppe:

n_1 FOR {Laufvariable = Anfangswert} TO {Endwert} STEP {Schrittweite}

n_2 NEXT {Laufvariable}

n_1, n_2 = Zeilennummern der Befehle

Laufvariable – muß eine Fließkommavariable sein (darf keine Integervariable sein)
Anfangswert – kann eine Konstante, eine Variable oder ein mathematischer Ausdruck sein
Endwert – kann eine Konstante, eine Variable oder ein mathematischer Ausdruck sein
Schrittweite – kann eine Konstante, eine Variable oder ein mathematischer Ausdruck sein. Unterbleibt die Angabe für die Schrittweite, so ist diese automatisch +1.

Die FOR-NEXT Befehlsfolge bewirkt, daß der Programmteil zwischen n_1 und n_2 solange durchlaufen wird bis die Laufvariable größer als der Endwert ist. Hiernach wird die Anweisung nach n_2 bearbeitet. Bei jedem Durchlauf des NEXT wird die Laufvariable, die zuerst gleich dem Anfangswert ist, um die Schrittweite erhöht und das Programm bearbeitet die Befehle zwischen n_1 und n_2.

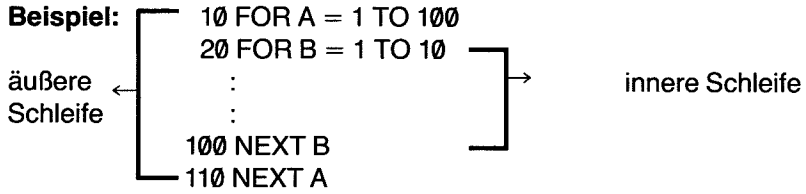
Beispiel:
```
10 FOR A = 1 TO 10 STEP 0.5
20 PRINT A
30 NEXT A
10 FOR K = 3*B TO C/5 STEP -1
    :
100 NEXT K
```

Die Laufvariable muß beim Rechner nicht im NEXT-Befehl angegeben werden. Also ist folgende Befehlsfolge auch korrekt:

```
100 FOR J = 5 TO 100
     :
200 NEXT
```

FOR-NEXT Schleifen können auch verschachtelt angeordnet werden.

Beispiel:



Die Verschachtelung muß symmetrisch sein, d. h. die zuletzt eröffnete Schleife muß als erste wieder geschlossen werden. Die Befehle 100 und 110 können zu einem Befehl:

100 NEXT B,A zusammengefaßt werden.

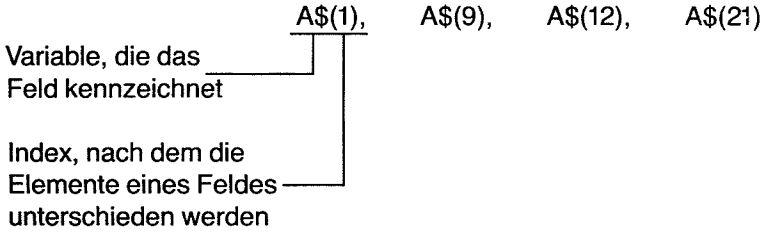
Bei verschachtelten Schleifen müssen unterschiedliche Laufvariablen benutzt werden.

1.2.18 Wieviel solls denn sein . . .

DER DIM-BEFEHL

Im Kapitel 1.1.2 haben Sie drei Formen von Variablen (A, A%, A$) kennengelernt. Es gibt nun eine Möglichkeit Gruppen einer dieser 3 Variablensorten zu einem „Feld" (Matrix, Array) zusammenzufassen. Ein solches Feld besteht dann aus mehreren Variablen einer Sorte (z. B. Textvariable), die indizierte Variablen genannt werden. Indizierte Variable, weil zu der normalen Variablen ein Index tritt. Während eine gewöhnliche Variable nur einen Speicherplatz repräsentiert, kennzeichnet man ein ganzes Feld von Speicherplätzen mit einer Variablen und unterscheidet die einzelnen Elemente durch Indizes. Das heißt indizierte Variablen aus einem Feld haben die gleiche Variable und unterscheiden sich nur durch ihren Index.

Beispiel: Indizierte Variablen aus einem Feld



Indizierte Variablen aus verschiedenen Feldern:

B1$(3), C%(5), A(12), A3(5), D3%(2)

Die im Beispiel gezeigten Variablen haben alle nur einen Index, deshalb nennt man diese Felder „eindimensionale Felder". Im COMMODORE-BASIC können aber auch mehrdimensionale Felder aufgebaut werden, d.h. Felder deren Elemente mehr als einen Index haben.

Diese Indizes werden innerhalb der Klammern durch Kommata getrennt.

Beispiel: Z%(1,Ø,2) X$(1,2) ZA(3,2,Ø,1)

Es kann auch jede Variable oder Formel deren Wert ≥ Ø ist als Index benutzt werden. (Nachkommastellen werden ignoriert!).

Beispiel: Y$(N,1) K%(A+2,B*3,Ø) KA(N↑3)

Hierdurch erreicht man eine indirekte Adressierung von Variablen, die durch ein Beispiel veranschaulicht werden soll:

```
10 FOR I = Ø TO 2
20 FOR J = Ø TO 2
40 FOR K = Ø TO 3
50 A(I,J,K) = RND (5)
60 NEXT K,J,I
70 STOP
```

Im Normalfall reserviert der Rechner pro Feld einen Speicherplatz für 11 indizierte Variablen pro Index. D.h., daß er für das im Beispiel benutzte dreidimensionale Feld Speicherplatz für 11*11*11 = 1331 Variablen reserviert. Das entspricht 6655 Bytes.

Diese Reservierung können wir jedoch mit dem folgenden Befehl ändern:

n DIM Name 1 (Zahl, Zahl, . . .), Name2 (Zahl, Zahl, . . .)

n = Zeilennummer des Befehls
Name = Variable durch die das Feld bezeichnet wird
Zahl = höchster Index, der benutzt wird (kann Variable oder Formel sein).

Für unser letztes Beispiel lautet der Befehl:

```
5 DIM A (2,2,3)
```

Er bewirkt, daß nur Speicherplatz für 3*3*4* = 36 Variablen reserviert wird.

Es ist auch möglich, mehr als 11 Speicherplätze zu reservieren:

Beispiel: 1Ø DIM A$(29)

Mit diesem Befehl wird ein Feld für 3Ø Textvariablen dimensioniert.

Weitere Beispiele:

```
DIM A% (2,N)
DIM B (2,J+3,K*2)
DIM C$ (1,BA,K↑3)
```

1.2.19 Einmal gewußt – immer gewußt . . .

DIE BEFEHLSGRUPPE DEF FN. ( )
FN. ( )

Soll eine bestimmte Berechnung mehrmals für verschiedene Zahlenwerte durchgeführt werden, so definieren wir eine Funktion durch den Befehl:

n_1 DEF FN Name (Variable) = Formel

n_1 = Zeilennummer des Befehls
Name = eine Variable fungiert als Name der Funktion
Variable = die Variable, die bei jedem Funktionsaufruf angegeben wird.
Formel = mathematischer Ausdruck.

Diese Funktion kann, nachdem sie definiert ist, beliebig oft direkt oder von verschiedenen Stellen des Programms durch den Befehl:

n_2 FN Name (Variable)

aufgerufen werden.

n_2 = Zeilennummer des Befehls (muß immer größer als n_1 sein)
Name = die Variable, die beim DEF-Befehl als Name gewählt wurde
Variable = die Variable oder die Konstante für die die Formel berechnet werden soll.

Beispiel:
```
10 DEF FNQ (Y) = Y ↑ 0.2
50 Z = FNQ (A)
```

Durch diese Befehle wird der Wert für die 5. Wurzel aus A im Speicherplatz Z abgespeichert.

Beispiel:

Soll in einem Programm mehrmals der Hyperbelsinus von verschiedenen Zahlen (X) berechnet werden, so definieren wir die Funktion mit dem Namen „H":

```
10 DEF FNH (X) = (EXP (X) – EXP (–X)) / 2
```

Wollen wir nun den Hyperbol-Sinus für 3 ausrechnen und ausdrucken so schreiben wir:

```
PRINT FNH (3)
```

Anmerkung: $\sinh(x) = \frac{e^x - e^{-x}}{2}$

1.2.20 Der Schlüssel nach draußen . . .

DER OPEN BEFEHL

Wie sie vielleicht wissen, können sie an Ihrem Rechner die verschiedenartigsten „Peripheriegeräte" anschließen. Zum Beispiel einen Ausgabedrucker oder ein Floppy-Disk-Laufwerk.

Um diese Geräte ansprechen zu können, müssen Sie ein sogenanntes logisches FILE, d.h. eine DATEI eröffnen, in die Daten geschrieben oder aus der Daten gelesen werden. Sie eröffnen eine DATEI durch den Befehl:

OPEN m_1, [m_2, m_3, "Name"]

m_1 = die logische Datei-Nr., ganze Zahl zwischen 1 und 255.
m_2 = Geräte-Nr. des anzusprechenden Gerätes
(wird 1 gesetzt, falls Angabe fehlt)

Ø = Tastatur
1 = # 1 Recorder
2 = # 2 Recorder
3 = Bildschirm
4 – 15 = externe Geräte

m_3 = Art der Datenbewegung (wird Ø gesetzt, falls Angabe fehlt)

Ø = lesen
1 = schreiben
2 = schreiben mit zusätzlichem END OF TAPE-Zeichen.

Name = kann angegeben werden um Dateien mit einem Namen direkt anzusprechen. Der Name darf aus höchstens 16 Zeichen bestehen und darf nicht in anderen Namen enthalten sein.

Es dürfen höchstens 10 Dateien gleichzeitig geöffnet sein! Wird diese Zahl überschritten, erfolgt die Fehlermeldung: ?TOO MANY FILES ERROR.

Beispiele:
30 OPEN 1, 2, 1, "DAT 1"

Der Befehl bewirkt das Öffnen der Datei 1 um auf der 2. Kassette Daten unter dem Namen DAT 1 abzuspeichern.

40 OPEN 2, 1, Ø

Dieser Befehl öffnet Datei 2 zum Lesen vom Recorder # 1.

Kürzer: 40 OPEN 2

1.2.21 und so schließe ich ab . . .

DER CLOSE BEFEHL

um zu verhindern, daß die maximale Anzahl der geöffneten Dateien (Files) überschritten wird, sollte jede Datei geschlossen werden, sobald die Lese- oder Schreiboperationen ausgeführt sind.

Dies geschieht durch:

n CLOSE m_1

n = Zeilennummer des Befehls
m_1 = die logische Datei-Nr. der zu schließenden Datei (ganze Zahl zwischen 1 und 255)

Beispiel: 100 CLOSE 2

Der Befehl bewirkt das Schließen der Datei-Nr. 2.

1.2.22 Ein Programm wird gespeichert . . .

DER SAVE BEFEHL

Das einmal in den Rechner eingegebene Programm kann auf einfache Weise gespeichert werden. Die einfachste Möglichkeit hierzu ist das Abspeichern des Programms mittels des Kassettenrecorders.

Sie beginnen den Vorgang mit dem Befehl:

SAVE ["Name"]

Name = höchstens 255 Zeichen können als Bezeichnung des Programms angegeben werden. (Sinnvoll beim Speichern mehrerer Programme auf einer Kassette).

Beispiel: SAVE "ABRECHNUNG"

Nach der Eingabe des SAVE-Befehls erscheint der Satz:

PRESS PLAY & RECORD ON TAPE # 1 auf dem Bildschirm des Rechners. Sie werden hierdurch aufgefordert die Tasten

RECORD	und	PLAY

des Kassettenrecorders gleichzeitig zu drücken. Sind die Tasten eingerastet, so erscheint OK WRITING gegebenenfalls die ersten 16 Zeichen des Namens des Programms. Nach Beendigung der Aufzeichnung erscheint READY und der Kassettenrecorder hält an.

Bei der Namensgebung eines Programmes ist zu beachten, daß ein Name nicht bereits im Namen eines anderen Programmes enthalten sein sollte (z. B. PROG und PROG 1), um Verwechslungen auszuschließen.

1.2.23 . . . und ein anderes geladen . . .

DER LOAD BEFEHL

Es gibt zwei Möglichkeiten ein auf der Kassette gespeichertes Programm in den Rechner zu laden.

Hier die erste:

Sie drücken gleichzeitig die Tasten:

SHIFT	RUN STOP

und auf dem Bildschirm erscheint LOAD, PRESS PLAY ON TAPE # 1. Kommt man der Aufforderung die Taste

PLAY

zu drücken nach, so kann man auf dem Bildschirm OK SEARCHING lesen. Sobald der Rechner den ersten Programmanfang nach der augenblicklichen Bandposition gefunden hat, zeigt der Bildschirm: FOUND bzw. FOUND Name, falls das gefundene Programm unter einem Namen abgespeichert wurde. (siehe 1.2.22). Nach Beendigung des Ladens wird das Programm automatisch gestartet.

Die zweite Möglichkeit

Sie geben Buchstabe für Buchstabe den Befehl:

LOAD ["Name"]

Name = muß nur eingegeben werden, falls sie ein bestimmtes Programm suchen. Werden auf der Suche nach diesen Programmen andere Programme gefunden, so werden diese auf dem Bildschirm durch „FOUND Name" registriert, jedoch nicht geladen.

Der Rechner beantwortet den LOAD-Befehl mit PRESS PLAY ON TAPE # 1 und dann das Drücken der Taste

PLAY

mit OK SEARCHING bzw. OK SEARCHING FOR Name. Hat er entweder das Programm mit dem gesuchten Namen oder, falls kein Name eingegeben wurde, den ersten Programmanfang gefunden, so erscheint: „FOUND Name", „LOADING" auf dem Bildschirm. Nach Beendigung des Ladevorganges erfolgt die Meldung „READY" auf dem Bildschirm und das Programm kann durch den Befehl: „RUN" (als Einzelbuchstaben) gestartet werden.

1.2.24 . . . geprüft und für gut befunden . . .

DER VERIFY-BEFEHL

Um zu kontrollieren, ob das Programm im Rechner mit dem abgespeicherten Programm identisch ist, dient der Befehl

VERIFY ["Name"]

Name = Name des abgespeicherten Programms (Falls Programm unter einem Namen abgespeichert ist).

Nach Drücken der Taste

PLAY

(siehe 1.2.23) meldet der Rechner entweder „OK", d.h. das Programm ist identisch oder aber „?VERIFY ERROR", d.h. das Programm ist nicht richtig abgespeichert und der Befehl SAVE muß wiederholt werden.

Soll ein Programm auf eine Kassette überspielt werden, die zum Beispiel schon zwei Programme enthält, kann man folgendermaßen vorgehen: Kassette zurückspulen

VERIFY (es erscheint ? VERIFY ERROR)
nochmals VERIFY (es erscheint ? VERIFY ERROR)

Nun steht der Aufnahmekopf hinter dem 2. Programm und es kann ein 3. Programm dahinter gespeichert werden.

1.2.25 Mein Wissen wird zum Teil gelöscht . . .

DER CLR-BEFEHL

Der Befehl

CLR

löscht sämtliche Variablen einschließlich Dimensionierung (siehe 1.2.18) und beinhaltet den RESTORE-Befehl (siehe 1.2.9). Weiter werden auch die Zähler der FOR-NEXT Schleifen und der GOSUB auf Ausgangsstellung gesetzt.
Beispiel: 12Ø CLR

1.2.26 Die Länge wird geprüft . . .

DER LEN ( )-BEFEHL

Der folgende Befehl dient dazu die Länge einer Textvariablen (z.B. A$) zu errechnen.

```
LEN ({Textvariable, "Text"})
```

Textvariable = hier muß die Textvariable stehen, deren Länge errechnet werden soll, oder der Text selbst (in Anführungszeichen).

Bei der Berechnung der Länge werden auch Leerzeichen (Blanks) mitgezählt.

Beispiel:
```
10 A$ = "COMPUTER"
20 PRINT LEN (A$)
READY
RUN
8
READY
```

Die Länge einer Textvariablen kann als Zahlvariable benutzt werden, d.h. es sind folgende Befehle denkbar:

```
A = LEN (A$) *3
10 FORK = 1 TO LEN (A$) . . .
```

1.2.27 Mit Texten kann man nicht rechnen . . .

DER VAL ( )-BEFEHL

Wie die Überschrift schon besagt, kann man mit Textvariablen nicht rechnen, auch wenn sie Ziffern enthalten. Mit dem Befehl:

```
VAL ({Textvariable, "Text"})
```

können die führenden Ziffern einer Textvariable in eine Zahl umgeformt werden, falls ihre ersten Zeichen +, −, Dezimalpunkt oder Ziffern sind. Beginnt die Textvariable mit nichtnumerischen Zeichen, so ist VAL (Textvariable) = 0.

Beispiele:
```
          ? VAL ("1357 COMPUTER")
Anzeige: 1357
          B$ = "−1.5E + 2DM"
          ? VAL (B$)
Anzeige: −150
          C$ = "COMPUTER 12"
          ? VAL (C$)
Anzeige: 0
```

Nur die Ziffern, die als erste Zeichen der Textvariablen stehen, werden umgeformt.

1.2.28 Verwandeln kann man auch . . .

DER STR$ ( )-BEFEHL

Wollen sie eine Zahl oder eine Variable in eine Textvariable umwandeln, so benutzen Sie den Befehl:

```
STR$ ({Variable, Zahl})
```

Beispiele:
```
10 A$ = STR$ (B)
20 C$ = STR$ (12.5)
```

1.2.29 So sieht ein ASCII-CODE aus . . .

DER ASC ( )-BEFEHL

Da der Rechner intern nur Zahlen verarbeiten kann, ist jedem Zeichen ein Zahlenwert zugeordnet. Diese Zuordnung geschieht nach dem ASCII (American Standard Code for Information Interchange), siehe Tabelle in 2.1.

Wollen wir kontrollieren welche Zahl dem ersten Zeichen einer Textvariablen im ASCII entspricht, so benutzen wir den Befehl:

```
ASC ({Textvariable, "Zeichen"})
```

Beispiel:
```
10 A$ = "ASCII"
20 ?ASC (A$)
READY
RUN
65
READY
```

Das obige Beispiel zeigt, daß dem Zeichen A die Zahl 65 im ASCII zugeordnet ist.

Beispiel: ?ASC("3")

1.2.30 . . . die passende Umkehrung . . .

DER CHR$ ( )-BEFEHL

Der CHR$-Befehl stellt die Umkehrung des ASC ( ) Befehls dar. Mit seiner Hilfe lassen sich aus den Zahlen 0 – 255 die dazugehörigen Zeichen ermitteln:

```
CHR$ ({Variable, Zahl})
```

Der CHR$ ( ) Befehl kann sich auf eine Zahl oder eine Variable beziehen.

Beispiel: A$ = CHR$ (65)

Das zum ASCII 65 gehörende Zeichen (A) wird ermittelt und in A$ gespeichert.

Mittels dieses Befehls können sie sich leicht eine ASCII Tabelle selbst erstellen:

```
10 FOR I = 0 TO 255
20 IF I = 147 THEN PRINT: GOTO 40
30 PRINT CHR $ (I), I
40 NEXT I
```

Merke: Die ersten 32 Zeichen sind Sonderzeichen die vom Rechner nicht angezeigt werden. Das Zeichen mit dem Code 147 entspricht Bildschirm löschen, es wird daher im obigen Programm übergangen.

1.2.31 Hätten Sie es gern links . . .

DER LEFT $ ( )-BEFEHL

Warum BASIC für seine leichte Textverarbeitung so gelobt wird, werden sie bei den folgenden vier Befehlen schnell erkennen.

Der erste Befehl

```
LEFT$ (Textvariable, Anzahl)
```

bewirkt, daß die im Befehl angegebene Anzahl von Zeichen, von links beginnend, abgespalten wird.

Die Ausgangstextvariable bleibt von diesem Befehl unberührt.

Textvariable = Textvariable, auf die LEFT angewandt wird.

Anzahl = Variable, Formel oder Konstante, die die Zahl der zu übernehmenden Zeichen angibt. Hat die Textvariable weniger Zeichen als angegeben, so wird sie ganz übernommen.

Beispiel:
```
10 A$ = "COMPUTER"
20 B$ = LEFT$ (A$,3)
30 ? B$
READY
RUN
COM
READY
```

weitere Beispiele:
```
30 B$ = LEFT$ (DA$,3*C)
50 PRINT LEFT$ (B$,2)
```

1.2.32 . . . oder bevorzugen Sie rechts . . .

DER RIGHT$ ( )-BEFEHL

Der Befehl . . .

RIGHT$ (Textvariable, Anzahl)

Textvariable = Textvariable auf die RIGHT angewandt wird.

Anzahl = Variable, Formel oder Konstante, welche die Zahl der zu übernehmenden Zeichen angibt. Hat die Textvariable weniger Zeichen als angegeben, so wird sie ganz übernommen.

. . . bewirkt, daß die im Befehl angegebene Anzahl von Zeichen, von rechts beginnend abgespalten wird. Die Ausgangstextvariable bleibt von diesem Befehl unberührt.

Beispiel:
```
10 A$ = "COMPUTER"
20 B$ = RIGHT$ (A$,5)
30 ? B$
READY
RUN
PUTER
READY
```

1.2.33 . . . und für Unentschlossene die . . . Mitte . . .

DER MID$ ( )-BEFEHL

Der Befehl . . .

MID $ (Textvariable, Zeichennr., Anzahl)

Textvariable = Textvariable, auf die MID angewandt wird.

Zeichennr. = Variable, Formel oder Konstante als Zeiger

Anzahl = Variable, Formel oder Konstante, die die Zahl der zu übernehmenden Zeichen angibt.

. . . bewirkt, daß von einem gewissen Zeichen an (Zeichennr.) eine bestimmte Anzahl von Zeichen (Anzahl) übernommen wird.

Beispiele:
```
1  A$ = "COMPUTER"
2  B$ = MID$ (A$,4,4)
3  ?B$
READY
RUN
PUTER
READY
10  D$ = MID$ (C$,I+2,D*3)
```

Eine kleine Spielerei:

```
10 A$ = "COMPUTER"
20 FOR I = 1 TO 8
30 PRINT MID$ (A$,1,I), I
40 NEXT
```

1.2.34 Der Schlüssel zur Maschinensprache, der SYS ( )-Befehl

Der SYS-Befehl dient dazu, innerhalb eines BASIC-Programmes Unterprogramme aufzurufen, die in der Maschinensprache geschrieben werden.

SYS (Startadresse)

Beispiel: 10 SYS (65490)

Hierbei ist die Zahl 65490 die Dezimaladresse des Speicherplatzes, zu dem die Programmausführung verzweigt. Die Startadresse muß eine positive Zahl und darf nicht größer als 65535 sein.

Ab dieser Adresse wird der Inhalt als Maschinenprogramm verstanden und auch ausgeführt.

Das Maschinenprogramm wird beendet, sobald der Maschinenbefehl RTS gelesen wird.

Die Programmausführung verzweigt wieder zurück zum BASIC-Programm, und zwar zu dem Befehl, der hinter dem SYS-Befehl folgt. Die Logik ist also die gleiche, wie bei dem Aufruf von BASIC-Unterprogrammen mit dem Befehl GOSUB.

Um nun Programme in der Maschinensprache zu erstellen, ist es notwendig, die Befehle des Mikroprozessors 6502 zu kennen. Hier sei als Literatur die Programmierfibel zum 6502 empfohlen (erhältlich über Ihren Commodore-Vertragshändler).

1.2.35 So geben Sie noch Daten mit . . . Der USR ( )-Befehl

Der USR-Befehl ist eine Erweiterung des SYS-Befehls. Er dient ebenfalls dazu, Unterprogramme aufzurufen, die in der Maschinensprache geschrieben sind. Als Erweiterung ist es mit dem USR-Befehl jedoch möglich, eine beliebige Variable an das Unterprogramm zu übergeben.

USR (Variable)

Beispiel: A = USR (A)

Beispiel: 10 ON USR (SIN(X) / SQR(Y)) GOTO 100, 200, 300

Bevor der USR-Befehl aufgerufen wird, sind jedoch einige Punkte unbedingt zu beachten.

1. Da aus dem USR-Befehl selbst nicht hervorgeht, wo das Unterprogramm steht, muß die Programmanfangsadresse in die Speicherstelle 1 und 2 geschrieben werden.

Beispiel: Das Unterprogramm beginnt bei der Speicherstelle 7000. Die Speicherstellen 1 und 2 müssen daher zuerst wie folgt mit POKE belegt werden:

```
POKE 2, INT (7000/256)      Höherwertiges Byte
POKE 1, 7000 AND 255        Niederwertiges Byte
A = USR  (10)
|         └──────────── Variable für das Unterprogramm
└────────────────────── Ergebnis vom Unterprogramm
```

2. Der Übergabebereich der Variablen liegt an den Speicherstellen 176 bis 183. Das ist nur wichtig für das Maschinenprogramm. Siehe Beschreibung des TIM-Monitorprogramms im Anhang.
3. Sind die Speicherstellen 1 und 2 einmal belegt worden, so brauchen sie nicht vor jedem weiteren USR-Befehl erneuert werden.
4. Grundsätzlich sollten Sie sehr vorsichtig mit den SYS- und USR-Befehlen umgehen, da ein falscher Sprung in ein Unterprogramm den Rechner zu nicht beeinflußbaren Operationen veranlaßt.

   Danach können Sie nur durch Aus- und Einschalten den Normalzustand erreichen.

1.2.36 Ein Elixier für Starprogrammierer, die Befehlsgruppe POKE, PEEK ()

Die Befehle PEEK und POKE zählen zu den Sonderbefehlen, mit denen über das herkömmliche BASIC hinaus Speicherplatzoperationen durchgeführt werden können.

POKE – ermöglicht es, direkt in eine bestimmte Speicherstelle einen Wert hineinzuschreiben.

POKE Speicherstelle, Wert

Beispiel: POKE 59468,12

In die Speicherstelle (dezimal) 59468 wird hier der Zahlenwert 12 eingeschrieben. Dieses Beispiel ist gleichzeitig der Befehl für die Umschaltung von Grafikzeichen auf Kleinbuchstaben; d.h. die Shift-Funktion der Tastatur erzeugt jetzt in Verbindung mit einer Taste den zugehörigen Kleinbuchstaben, siehe Anhang VIII für die –2-Tastatur.

Dieser Zustand bleibt erhalten, bis entweder das Gerät ausgeschaltet wird oder bis der Umschaltbefehl

POKE 59468,14 folgt.

PEEK – Liest den Inhalt einer Speicherstelle, die dezimal angegeben wird.

PEEK (Speicherstelle)

Beispiel: ? PEEK (59468) Anzeige: 14

1.3 DIE AUSFÜHRUNGSBEFEHLE

Diese Befehle kann man als Steuerbefehle für ein Programm auffassen; wir bestimmen damit, was mit dem Programm geschehen soll, ob es

- mit **RUN** gestartet werden soll,
- mit **STOP** gestoppt werden soll,
- ob es mit **LIST** ausgedruckt bzw. am Bildschirm dargestellt werden soll,
- mit **NEW** gelöscht werden soll,
- mit **CONT** (CONTINUE) nach einem STOP-Befehl weitergestartet werden soll oder mit
- **GOTO** ab einer bestimmten Adresse gestartet werden soll, oder ob mit
- **CLR** alle Variablen gelöscht werden sollen.

Um diese Befehle kennenzulernen, müssen wir ein kleines Beispielprogramm in den Rechner eingeben.

Geben Sie bitte nachfolgendes Programm Zeile für Zeile in den Rechner ein. Nach jeder Zeile betätigen Sie die RETURN-Taste. Mit dieser RETURN-Taste wird die Programmzeile, die ja zunächst nur im Bildschirm steht, in den Arbeitsspeicher gebracht, wo sie später zur Ausführung gebracht werden kann.

```
10 INPUT A, B, C
20 PRINT A, B, C
30 STOP
40 PRINT „BEISPIEL“
50 END
```

1.3.1 Das Programm wird gestartet

Mit dem Befehl

RUN (RETURN)

wird das gesamte Programm gestartet. In dem Beispielprogramm erkennt man das daran, daß im Bildschirm ein Fragezeichen erscheint. Dieses Fragezeichen ist typisch für die Input-Anweisung der Zeile 10.

Nach Eingabe von 3 numerischen Werten

z.B. 1 (RETURN)
2 (RETURN)
3 (RETURN)

setzt das Programm automatisch den Ablauf fort.

MERKE: Mit dem RUN-Befehl werden alle bereits im Rechner befindlichen Variablen gelöscht!

Der RUN-Befehl kann auch modifiziert angewendet werden; z.B. wird mit

RUN Zeilennummer

das Programm ab der angegebenen Zeilennummer gestartet.
In dem Beispiel läßt sich dieses mit

RUN 40

testen.

1.3.2 Die Verschnaufpause wird beendet . . . . . .

Wird der Ablauf eines BASIC-Programms durch einen Stop-Befehl unterbrochen, so läßt sich das Programm sehr einfach ab dieser Stelle fortsetzen durch

CONT (RETURN)

MERKE: Der CONT-Befehl verändert **keine** Variablen oder Werte eines Programms.

1.3.3 Und so sieht Ihr Programm aus . . . .

Möchten Sie sich das im Rechner befindliche BASIC-Programm einmal in seiner Gesamtheit ansehen, benutzen Sie die Befehlsfolge

LIST (RETURN)

Auf dem Bildschirm erscheint nun das gesamte BASIC-Programm, sortiert nach aufsteigenden Zeilennummern.

Selbstverständlich lassen sich auch Teilbereiche des Programms ausgeben.

LIST 40 (RETURN) listet nur Zeile 40
LIST –30 (RETURN) listet bis Zeile 30
LIST 30– (RETURN) listet ab Zeile 30
LIST 20–30 (RETURN) listet von Zeile 20 bis 30
LIST 40–40 (RETURN) listet nur Zeile 40

1.3.4 Man muß nicht immer am Anfang starten ....

Sollte der Ablauf eines Programms einmal ab einer bestimmten Zeilennummer fortgesetzt werden **ohne** die im Rechner befindlichen Werte zu löschen bzw. **ohne** die Anweisung STOP zu benutzen, bietet sich die GOTO-Anweisung an.

Syntax: GOTO Zeilennummer

Beispiel: GOTO 40 (RETURN)

MERKE: Die GOTO-Anweisung verändert keine Variablen oder Werte.

1.3.5 So kann man altes vergessen .....

Soll der Rechner in den sogenannten Einschaltzustand gebracht werden, benutzt man die Befehlsfolge

NEW (RETURN)

d.h. das gesamte Programm und sämtliche im Rechner befindlichen Variablen werden gelöscht.

Die in 1.3. aufgeführten Ausführungsbefehle sind außer CONT auch programmierbar. Sie können also wie normale Befehle ins Programm geschrieben werden.

2. DIE BESONDERHEITEN DES 2001 UND DER 3001-COMPUTER

2.1. Die graphischen Symbole (siehe Anhang VIII für die –2-Tastatur.)

Neben den großen Buchstaben auf der Tastatur existieren noch eine Vielzahl von Sonderzeichen, nämlich die graphischen Symbole.

Diese Symbole werden dargestellt, wenn gleichzeitig die SHIFT-Taste und eine weitere Taste mit den dargestellten Sonderzeichen betätigt wird.

Zusätzlich ist es möglich, auch in Kleinschrift zu schreiben.

Der Rechner besitzt 3 sogenannte Zeichensätze:

1. Großschrift und Sonderzeichen
2. Graphische Symbole
3. Kleinschrift

Wobei 1. immer vorhanden ist und 2. oder 3. wahlweise einschaltbar sind.

Der Befehl zum Umschalten zwischen 2. und 3. Zeichensatz lautet:

POKE 59468,12 (Kleinschrift gewählt)
POKE 59468,14 (Graphische Symbole gewählt)

Bei der Umschaltung ist zu beachten, daß der gesamte Bildschirminhalt umgeschaltet wird.

Beispiel:
```
10 FOR I = 1 TO 255
20 IF I = 147 THEN PRINT: GOTO 40
30 PRINT CHR$ (I);
40 NEXT
50 POKE 59468,14: FOR I = 1 TO 999: NEXT
   POKE 59468,12: FOR I = 1 TO 999: NEXT: GOTO 50
```

Nachfolgende Tabelle gibt den Zusammenhang zwischen Dezimal, Hexa-Dezimal, ASCII und Bildschirmcode wieder.

DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM	DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM	DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM	DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM
0	00	NULL	@	33	21	!	!	66	42	B		99	63	c	
1	01	SOH	A	34	22	"	"	67	43	C		100	64	d	
2	02	STX	B	35	23	≠	≠	68	44	D		101	65	e	
3	03	ETX	C	36	24	$	$	69	45	E		102	66	f	
4	04	EOT	D	37	25	%	%	70	46	F		103	67	g	
5	05	ENQ	E	38	26	&	&	71	47	G		104	68	h	
6	06	ACK	F	39	27	'	'	72	48	H		105	69	i	
7	07	BL	G	40	28	(	(	73	49	I		106	6A	j	
8	08	BS	H	41	29	)	)	74	4A	J		107	6B	k	
9	09	HT	I	42	2A	*	*	75	4B	K		108	6C	l	
10	0A	LF	J	43	2B	+	+	76	4C	L		109	6D	m	
11	0B	VT	K	44	2C	,	,	77	4D	M		110	6E	n	
12	0C	FF	L	45	2D	–	–	78	4E	N		111	6F	o	
13	0D	CR	M	46	2E	.	.	79	4F	O		112	70	p	
14	0E	LF	N	47	2F	/	/	80	50	P		113	71	q	
15	0F	SI	O	48	30	0	0	81	51	Q		114	72	r	
16	10	DLE	P	49	31	1	1	82	52	R		115	73	s	
17	11	DC_1	Q	50	32	2	2	83	53	S		116	74	t	
18	12	DC_2	R	51	33	3	3	84	54	T		117	75	u	
19	13	DC_3	S	52	34	4	4	85	55	U		118	76	v	
20	14	DC_4	T	53	35	5	5	86	56	V		119	77	w	
21	15	NAK	U	54	36	6	6	87	57	W		120	78	x	
22	16	SYC	V	55	37	7	7	88	58	X		121	79	y	
23	17	ETB	W	56	38	8	8	89	59	Y		122	7A	z	
24	18	CAN	X	57	39	9	9	90	5A	Z		123	7B	{	
25	19	EM	Y	58	3A	:	:	91	5B	[		124	7C	:	
26	1A	SUB	Z	59	3B	;	;	92	5C	/		125	7D	}	
27	1B	ESC	[	60	3C	<	<	93	5D	]		126	7E	~	
28	1C	FS	/	61	3D	=	=	94	5E	↑		127	7F	DEL	
29	1D	GS	]	62	3E	>	>	95	5F	←					
30	1E	RS	↑	63	3F	?	?	96	60	'					
31	1F	US	←	64	40	@	–	97	61	a					
32	20	␣	␣	65	41	A	♠	98	62	b					

2.2. DIE EINGEBAUTE UHR . . . . ein Leckerbissen

Jeder Computer braucht, um seine Befehle auszuführen, einen Taktgenerator, der die Geschwindigkeit der Befehlsausführung bestimmt. Um die Genauigkeit des Generators zu erhöhen, verwenden manche Computerhersteller dazu Quarze, wie sie auch in Uhren verwendet werden. Bei Ihrem Rechner wird dieser Taktgenerator gleichzeitig als Uhr mit verwendet.

Diese Uhr kann in 2 Formen abgerufen werden.

Geben Sie folgendes ein:

PRINT TI$ (RETURN)

Die Antwort sind 6 Ziffern die Ihnen die Zeit seit Einschalten des Rechners angeben.

Die Ziffern bedeuten:

Ziffer 1 + 2: Stunden
Ziffer 3 + 4: Minuten
Ziffer 5 + 6: Sekunden

Geben Sie ein:

PRINT TI

Die Antwort ist eine Zahl, die Ihnen die Anzahl von 1/60 Sekunden seit Einschalten des Rechners angibt.

Wenn Sie die Uhr auf eine bestimmte Zeit stellen wollen, so verändern Sie die Variable TI$.

Beispiel: TI$ = "143000" (Die Uhr wird auf 14h.30 gestellt.)
TI$ = "080530" (Die Uhr wird auf 8h.05 u. 30 Sekunden gestellt.)

Ein Verändern der Variablen TI ist nicht möglich.

Der Anwendungsbereich der Uhr ist sehr leicht erklärt:

Zeitgenaue Meßwerterfassung
Terminkalender für den Tag
Abbrechen von Programmen nach einer bestimmten Zeit
Überprüfung von Zeitabläufen

Ein kleines Beispiel soll eine der Anwendungen der Uhr veranschaulichen.

```
100  INPUT "WIE SPÄT IST ES"; A$
110  IF LEN (A$) < 6 THEN 100
120  B$ = ""
130  FOR I = 1 TO LEN (A$)
140  IF MID$ (A$,I,1) <"0" or MID$ (A$,I,1)
     > "9"THEN 160
150  B$ = B$ + MID$ (A$,I,1)
160  NEXT I
170  IF LEN (I$) <> 6 THEN 100
180  TI$ = B$
190  PRINT "DRÜCKEN SIE [RETURN], WENN SIE"
200  PRINT "WISSEN WOLLEN, WIE SPÄT ES IST."
210  GET IN$: IF IN$ = " "THEN 210
220  IF ASC (IN$) >< 13 THEN 210
230  PRINT "ES IST JETZT:";
240  B$ = TI$
250  A$ = LEFT$ (B$,2) + "UHR", + MID$ (B$,3,2)
     + "MIN" + RIGHT$ (B$,2) + "SEK"
260  PRINT A$
270  GOTO 210
```

2.4 Die Möglichkeiten Peripherie anzuschließen

Das nachfolgende Bild illustriert die Anschlußmöglichkeiten für externe Geräte.

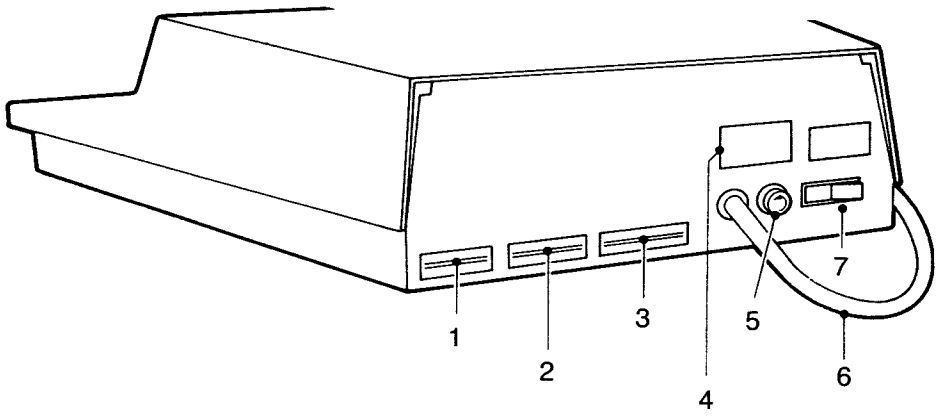


Abb. 1: Rückseite des Computers

1 = Anschluß für externen Recorder
2 = 8 Bit Parallel-Schnittstelle
3 = IEC-Bus-Schnittstelle
4 = Seriennummer
5 = 0,6 Amperé-Sicherung
6 = Netzkabel
7 = Netzschalter

2.4.1 DER RECORDER

Der externe Recorder wird über den vorhandenen Steckerkontakt auf der Rückseite des Rechners angeschlossen.

Die logische Behandlung des Kassettengerätes geschieht mittels der Anweisungen:

OPEN	Kapitel 1.2.20
CLOSE	
SAVE	bis
LOAD	
VERIFY	
PRINT #	
INPUT #	

Soll ein Programm auf dem Recorder gespeichert werden, so lautet die Befehlsfolge

SAVE "NAME",

oder

"SAVE",

Entsprechend:

LOAD "NAME",

bzw.

"LOAD",

Der 2. Recorder wird innerhalb des Gerätes an der Platine angeschlossen. Beim Zugriff auf diesen Recorder muß beim Öffnen der Datei der Gerätenummer 2 angesprochen werden; desgleichen beim Laden und beim Lesen.

Sollen Daten auf den Recorder geschrieben oder vom Band gelesen werden, so verwendet man PRINT# bzw. INPUT# .

Beispiel: 1Ø OPEN 5,1,1 (Siehe Kapitel 1.2.20)
2Ø FOR K = 1 TO 5Ø
3Ø PRINT # 5,K
4Ø NEXT K
5Ø CLOSE 5

Für den Lesevorgang sieht das Beispiel geringfügig anders aus.

Beispiel: 5 DIM A (5Ø)
1Ø OPEN 5,1,Ø
2Ø FOR K = 1 TO 5Ø
3Ø INPUT# 5,A(K)
4Ø NEXT K
5Ø CLOSE 5

Anmerkung: Beim 2001-Computer ist es notwendig, vor jedem OPEN-Befehl, der sich auf einen Recorder bezieht, folgende 2 Befehle einzufügen:
POKE 243,122: POKE 244,2 **vor** OPEN für Recorder # 1
bzw. POKE 243,58: POKE 244,3 **vor** OPEN für Recorder # 2
Bei den Geräten der 3001-Serie dürfen diese Befehle **nicht** verwendet werden.

2.4.2 Der 8 Bit Parallelanschluß

Der Parallelanschluß, (Abb. 1, Pkt. 3), auch User-Port genannt, besteht aus 12 Doppelkontakten (Abstand Kontakt zu Kontakt: 0,156 inch) für die verschiedensten Möglichkeiten des Sendens und Empfangens von Informationen.

Die unteren Kontakte des Parallelanschlusses (CA1, PAØ-7, CB2) sind direkt auf einen Interface-Adapter VIA 6522 geführt und können hexadezimal ab $E84Ø bzw. dezimal ab 59456 angesprochen werden.

Näheres siehe Anhang S (Schnittstellenbeschreibung).

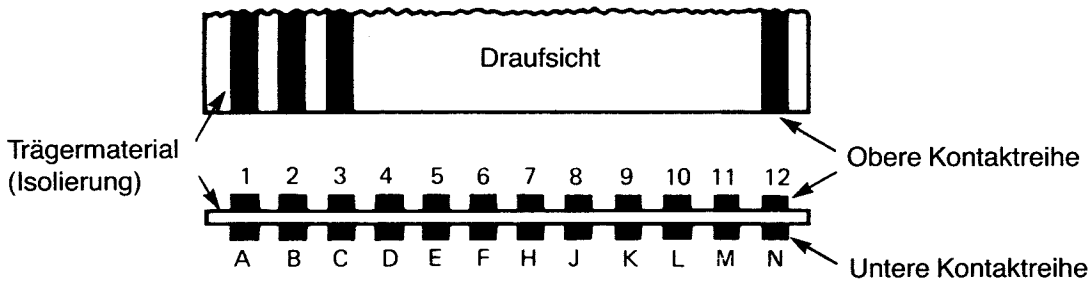


Abb. 2 8 Bit Parallelanschluß
Sicht auf Rückseite Computer.

2.5. DATEIVERWALTUNGEN

Im Zusammenhang mit dem IEC-Bus ist die Behandlung von Dateien und das Wissen darüber unentbehrlich.

Dateien werden benutzt, um Daten oder Informationen abzuspeichern und/oder wiederzubeschaffen. Das geschieht beispielsweise auch mit Programmen auf der Kassette. Eine Erweiterung dieser Erklärung ist es, jede Vorrichtung, die Daten empfangen oder erzeugen kann, als logische Datei zu benennen. Zum Beispiel kann ein Digitalvoltmeter auf Anfrage des Rechners seine Informationen über den IEC-Bus senden. Da aber verschiedene Geräte wie Kassette, Bildschirm, Tastatur, Voltmeter u. a. völlig verschieden aufgebaut sind, unterscheidet man bei diesen zwischen einer physikalischen und einer logischen Adresse dieses Gerätes.

a) Die physikalische Adresse eines Gerätes ist unabänderlich und vom Hersteller festgelegt. Bei Ihrem Modell kann die physikalische Adresse von Ø – 15 gehen. Folgende Adressen sind bereits belegt:

physikalische Adresse	**Gerät**
Ø	Tastatur
1	Recorder # 1
2	Recorder # 2
3	Bildschirm

Als Geräte 4 – 15 sind IEC-Bus-Geräte nach Belieben anschließbar.

b) Die logische Adresse einer Datei ist von 1 bis 255 von Ihnen frei definierbar. Sie legen die Zuweisung einer logischen Adresse beim Eröffnen dieser Datei fest. Diese Verknüpfung hat für Sie den Vorteil, daß Sie beim Ändern eines physikalischen Gerätes nur angeben, um welches Gerät es sich handelt. In dem folgenden Beispiel sehen Sie, daß eine Tabelle, von dem selben Unterprogramm generiert, einmal auf dem Bildschirm ausgegeben wird und das zweite Mal auf der Kassette:

```
10 OPEN 1,3,1: REM Bildschirm
11 GOSUB 100: REM Sprung ins Unterprogramm
12 CLOSE 1
13 Open 1,1,1: REM Recorder
14 GOSUB 100
15 CLOSE 1
16 END
100 FOR I = TO 25: REM Unterprogramm
101 PRINT # 1, I; SQR (I)
102 NEXT I
103 RETURN
```

Zusätzlich zu der physikalischen und logischen Adresse können Sie beim Eröffnen eines Gerätes noch eine Zweitadresse angeben. Die Zweitadresse gibt einem intelligenten Peripheriegerät beim Eröffnen zusätzlich noch an, in welcher Art es die Daten zu behandeln hat. Beim Drucker gibt es beispielsweise 6 verschiedene Arten, die Daten zu drucken:

Zweitadresse	**Druckart**
Ø	Normal drucken
1	Drucken unter Formatkontrolle
2	Übersetzen des Formates
3	Setzen variable Anzahl Zeilen/Seite
4	Benutzen der Fehlerdiagnose
5	Übertragen eines programmierbaren Zeichens

Bei der Kassettenstation gibt es 3 verschiedene Arten, Daten zu verwalten:

Zweitadresse	**Operation**
Ø	Lesen von Daten
1	Schreiben von Daten
2	Schreiben von Daten und eine Bandende-Kennung beim Schließen der Datei

Als letzte Information können Sie einen Dateinamen beim Eröffnen eines Gerätes mit angeben. Das ist sinnvoll, wenn dieses Gerät verschiedene Dateien verwaltet. Bei Ihrer Kassette ist es beispielsweise so, daß Sie mehrere Dateien hintereinander ablegen können. Um nun eine bestimmte Datei auf dieser Kassette abzuspeichern, können Sie diese mit einem Namen versehen. Dieser Name kann bis zu 128 Zeichen umfassen, es werden allerdings beim späteren Suchen dieser Datei nur die ersten 16 Zeichen abgefragt.

2.5.1. KOMMUNIKATION

Um nun Daten mit einem peripheren Gerät auszutauschen, gibt es 6 spezielle Kommunikationsbefehle. Diese sind zwar schon in Kapitel 2 beschrieben, sollen aber an dieser Stelle noch einmal vertieft werden.

a) OPEN logische Adresse [Physikalische Adresse [Zweitadresse ["Dateiname"]]]
b) CLOSE logische Adresse
c) PRINT # logische Adresse [Beliebige Daten]
d) GET # logische Adresse, Variable [Variable, Kostante, arith. Ausdruck]
e) INPUT # logische Adresse, Variable [Variable, Konstante, arithm. Ausdruck]
f) CMD logische Adresse

Um jedem die Möglichkeit zu geben, die nachfolgenden Prozeduren nachzuvollziehen, beschränken wir uns dabei nur auf Kassette, Bildschirm und Tastatur. Es sei jedoch erwähnt, daß jedes weitere an den IEC-Bus angeschlossene Gerät gleichartig arbeitet. Hierbei ist nur zu unterscheiden, ob es sich um ein Eingabegerät (Tastatur), Ausgabegerät (Bildschirm) oder Mehrdateigerät (Kassette) handelt.

2.5.2. DATEIVERWALTUNG PER KASSETTE

Die Kassettenstation Ihres Rechners ist prinzipiell identisch mit einem normalen Kassettenrecorder. Unterschiedlich ist lediglich das Sende-Empfangsteil für digitale Informationen und ein Schalter, der die Stellungen der Bedienungstasten kontrolliert. So kann der Rechner feststellen, ob die PLAY, REWIND oder FAST FORWARD-Taste gedrückt ist, er kann aber nicht feststellen, um welche dieser drei Tasten es sich handelt.

Sobald die Taste PLAY gedrückt ist, kann der Rechner den Motor ein- und ausschalten. Er nutzt dies aus, um Dateien in einzelne Blöcke zu unterteilen.

Die Struktur einer Datei auf der Kassette sichert Ihnen ein Minimum an Platzverbrauch, kombiniert mit einem Maximum an Schreibsicherheit. So werden beispielsweise alle Daten zweimal hintereinandergeschrieben, um bei einer Bandschwäche defekte Daten rekonstruieren zu können.

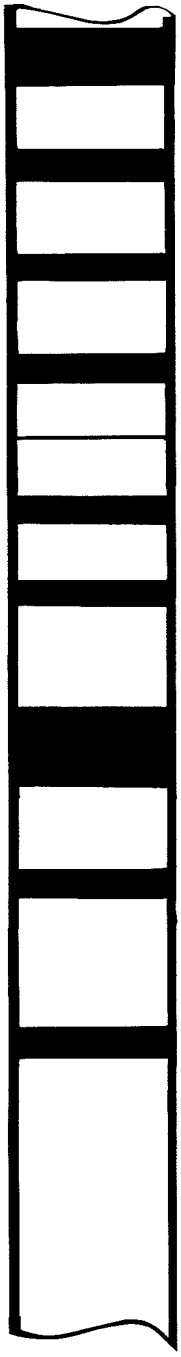
Geschwindigkeitsschwankungen des Bandes werden durch eine spezielle Anpassungstechnik ausgeglichen. So wird ein fester Ton zwischen jedem Datenblock und an den Anfang jeder Datei geschrieben. Durch die Länge dieses Tones stellt der Rechner fest, ob es sich um einen Dateikopf oder einen Dateiblock handelt.

Ein Dateikopf signalisiert dem Rechner, daß die Kassette an einem Dateianfang positioniert ist. Er enthält neben dem Dateinamen auch die Information, ob es sich um eine Datei oder ein Programm handelt.

Um nun den Zugriff auf die Information einer Datei zu beschleunigen, liest Ihr Rechner nicht nur die gewünschte Information, sondern immer einen kompletten Block von insgesamt 192 Zeichen, da das Starten und Stoppen des Kassettenmotors nach jeder einzelnen Information zu Zeitaufwendig ist.

Dieser Block wird in einen Pufferbereich eingelesen und von dort je nach Bedarf in die gewünschten Variablen gebracht. Beim Schreiben ist das Verfahren umgekehrt. Es werden die Informationen, so lange in den Puffer gebracht, bis dieser voll ist. Der gesamte Puffer wird dann gemeinsam auf das Band geschrieben.

Der Pufferbereich befindet sich für Kassette 1 an den Speicherstellen 634 – 825, für die 2. Kassettenstation von 826 – 1017. Für Programme werden diese Pufferbereiche nicht benutzt, da Programme nicht in mehreren Blöcken geschrieben werden. Nachfolgend ein Schema, daß Ihnen zeigt, wie eine Datei und ein Programm auf Kassette aufgezeichnet werden.



* 10 Sekunden Festton

192 Zeichen Dateikopf

2 Sekunden Festton

192 Zeichen Duplikat des Dateikopfes

* 2 Sekunden Festton

192 Zeichen 1. Datenblock

2 Sekunden Festton

192 Zeichen Duplikat 1. Datenblock

*

* 2 Sekunden Festton

192 Zeichen letzter Datenblock

2 Sekunden Festton

192 Zeichen Kopie des letzten Datenblocks

* 10 Sekunden Festton

192 Zeichen Programmkopf

2 Sekunden Festton

192 Zeichen Kopie des Programmkopfes

* 2 Sekunden Festton

10 – 32768 Byte Programm

* An diesen Stellen bleibt der Kassettenmotor stehen.

– Beispiel für eine Bandaufzeichnung –

2.5.3 EIN- AUSGABEOPERATIONEN BEI LOGISCHEN DATEIEN

Ein- Ausgaben teilen sich in 3 Schritte auf:

* Eröffnen der Datei – Mitteilung aller Informationen über diese physikalische und logische Datei
* Lesen oder Schreiben der Daten –
* Schließen der Datei – Trennen der Verknüpfung von logischer und physikalischer Datei

Die 3 Teile wollen wir im folgenden beschreiben:

*** ERÖFFNEN DER DATEI**

Die Eröffnung einer Datei dient dazu, dem Rechner alle Informationen über diese Datei mitzuteilen. Der Rechner selbst speichert diese Informationen in speziell dafür vorgesehene Tabellen, errichtet ggf. Pufferbereiche für diese Datei und sorgt dafür, daß diese Datei anhand der angegebenen logischen Adresse zukünftig angesprochen werden kann. Gleichzeitig sendet er Test- und Informationssignale an diese Datei, um ihre Funktionsfähigkeit zu prüfen und um sie auf Empfangs- bzw. Sendebereitschaft zu schalten.

Programmdateien, egal ob auf der Kassette oder auf anderen Geräten halten sich nicht an die Ein- Ausgabeschritte 1 – 3.

Um ein Programm in den Rechner einzuladen oder vom Rechner wegzuspeichern brauchen Sie jeweils nur einen einzigen Befehl,

zum Einladen:
LOAD "Programmname", physikalische Adresse

zum Abspeichern:
SAVE "Programmname", physikalische Adresse

*** LESEN ODER SCHREIBEN**

a) **LESEN**

Für das Lesen von Informationen einer Datei gibt es 2 Befehle:

INPUT # logische Adresse, Eingabevariable(n)
GET # logische Adresse, Eingabevariable

Der INPUT-Befehl veranlaßt den Rechner, solange zu warten, bis alle angegebenen Eingabevariablen von der Datei gesandt worden sind.

Im Gegensatz dazu holt der GET-Befehl lediglich ein Zeichen. Sofern kein Zeichen gemeldet ist, enthält die Eingabevariable eine Ø oder ein Null-Zeichen, je nachdem ob es sich um eine numerische oder eine Textvariable handelt. In jedem Fall wird das Programm ohne Pause fortgesetzt.

b) **SCHREIBEN**

Das Schreiben von Daten vom Rechner in eine Datei geschieht mit folgendem Befehl:

PRINT #logische Adresse, Variable(n)

Der PRINT-Befehl veranlaßt den Rechner, die angegebenen Variablen in der Reihenfolge ihres Auftretens an die Datei zu senden. Der Rechner wartet nach jedem gesandten Zeichen auf eine Fertigmeldung der Datei. Bleibt diese aus, z.B. durch Defekt des angesprochenen Gerätes, so „hängt" der Rechner an dieser Stelle. Nur durch Aus- Einschalten ist eine Reaktivierung möglich.

Besonderheit für Ausgaben:

Durch

CMD logische Adresse

ist es möglich, fast alle Bildschirmausgaben zu einer anderen Datei zu senden. Ausnahmen hierbei sind alle über Tastatur eingegebenen Zeichen, sowie Fehlermeldungen des Betriebssystems. Diese werden nach wie vor auf dem Bildschirm dargestellt. Vorteilhaft ist der CMD-Befehl beispielsweise für Programmlisten auf einem angeschlossenen Drucker:

```
OPEN 1, Druckeradresse (physikalisch)
CMD 1
LIST
PRINT #
CLOSE 1
```

(Durch den Befehl PRINT # log. Adresse wird ein vorhergehender CMD-Befehl wieder aufgehoben.)

*** SCHLIESSEN EINER DATEI**

Geschlossen wird eine Datei durch den Befehl:

CLOSE logische Adresse

Der Pufferbereich des Rechners wird entleert und alle Tabelleneintragungen gelöscht. Die angesprochene Datei ist nach dem CLOSE-Befehl logisch getrennt, d.h. die Kabelverbindung existiert zwar noch, alle Ein- Ausgabebefehle werden aber vom Betriebssystem verweigert.

2.5.4 FEHLERINFORMATION BEI EIN/AUSGABEN

Der Computer besitzt eine Variable ST, die nach einer Ein- Ausgabe eine Fehlerdiagnose beinhaltet. Durch Abfragen dieser Variablen ist es möglich, Fehler rechtzeitig zu entdecken.

Die Variable ST besitzt 8 Bit-Positionen. Jedes Bit beinhaltet eine getrennte Fehlermeldung. Durch Verknüpfung ist eine Erkennung von mehreren Fehlern möglich.

Folgende Fehlerquellen sind bei angegebenem Gerät erkennbar:

ST Bit	ST Zahlenwert	Recorder	IEEE-Bus
0	1	–	Zeitüberschreitung beim Schreiben
1	2	–	Zeitüberschreitung beim Lesen
2	4	Zu kurzer Block eingelesen	–
3	8	Zu langer Block eingelesen	–
4	16	nicht korrigierbarer Lesefehler	–
5	32	Prüfsummenfehler	–
6	64	Dateiende erreicht	Eingabeende
7	– 128	Bandende erreicht	Gerät nicht angeschlossen

Beispiel: Kassette: ST = 36 = 32 + 4

Es wurde ein zu kurzer Block eingelesen. Dadurch kam es zu einem Prüfsummenfehler.

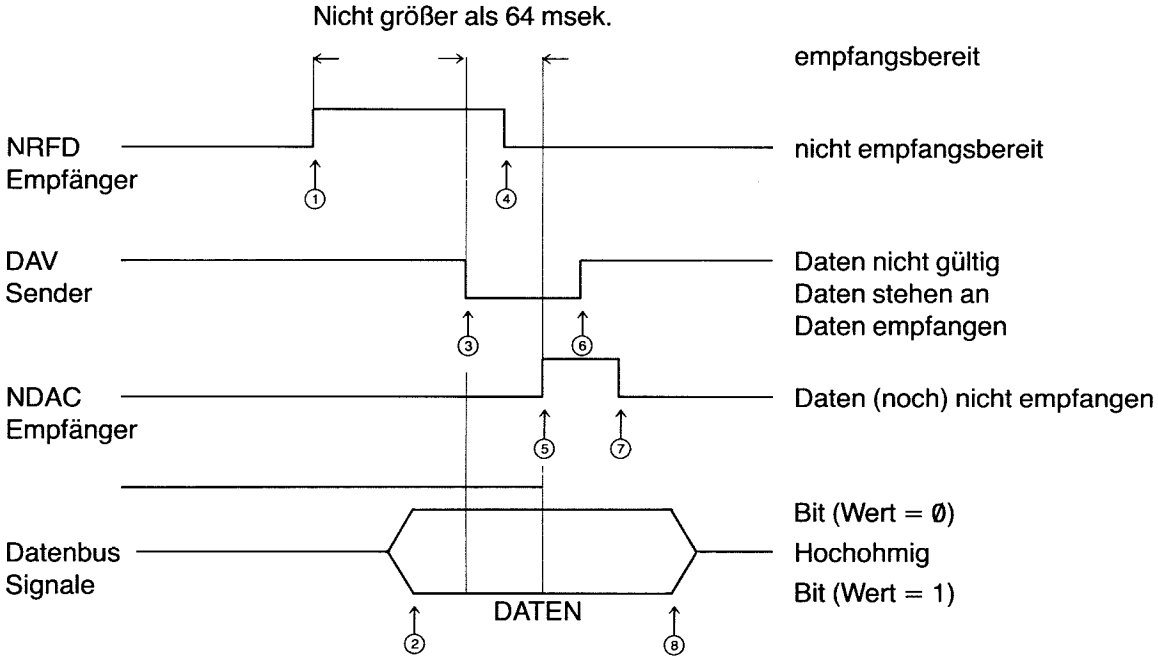
Wie sie sehen, ist eine gleichzeitige Erkennung mehrerer Fehler möglich.

2.6. Der IEC-Bus (IEEE-BUS)

Der IEC-Bus besteht aus 8 Datenleitungen, 3 Steuerleitungen und 5 Informationsleitungen.

Die Datenleitungen übertragen die eigentliche Information. Die Handshake-Leitungen sind zur Synchronisierung der Dateninformation zwischen Sender und Empfänger.

Der IEC-Bus benutzt ein Dreidraht-Handshake-Verfahren.
Der Ablauf geschieht folgendermaßen:



Der Sender steht permanent auf Sendebereitschaft. Meldet der Empfänger, daß er empfangsbereit ist (1), so stellt der Sender die Daten auf dem Datenbus zur Verfügung (2). Er meldet diese Bereitstellung durch die Meldung „Data valid" (Daten gültig) (3).

Sobald der Empfänger das DAV-Signal empfängt, nimmt er die Daten-Empfangsbereitschaft weg und meldet damit dem Sender, daß er vorläufig noch keine neuen Daten gebrauchen kann (4). Nach einer kurzen Verzögerungszeit, die vom inneren Aufbau des Empfängers abhängt, hat er diese Daten übernommen und meldet über die NDAC-Leitung diese Tatsache (5) an den Sender zurück. Daraufhin kann der Sender die Daten und das DAV-Signal vom Bus wegnehmen (6) (8). Wenn das DAV-Signal verschwunden ist, nimmt der Empfänger auch das NDAC-Signal weg (7); damit ist ein Übertragungszyklus beendet und der Bus ist in seinem Ruhestand.

Wenn der Empfänger jetzt die nächsten 8 bit aufnehmen kann, meldet er das über die NRFD-Leitung und der nächste Zyklus kann beginnen.

Die 5 Informationsleitungen haben folgende Bedeutung:

- IFC – Interface clear

  Über diese Leitung sendet Ihr Rechner beim Einschalten ein allgemeines Rücksetzsignal (100 msec), mit dem sich alle angeschlossenen Geräte ebenfalls in den Einschaltezustand versetzen können.

- REN – Remote enable

  Diese Leitung dient im allgemeinen dazu, Handbedienungen von angeschlossenen Geräten zu verweigern. Bei Ihrem Computer ist diese Leitung gegen Masse kurzgeschlossen.

- SRQ – Service Request

  Über diese Leitungen können angeschlossene Geräte zu sendende Informationen anmelden. Diese Leitung wird zwar vom Betriebssystem des Rechners nicht abgefragt, Sie jedoch können mit dem Befehl

  ? (PEEK(59427)AND128)/128

  diese Leitung auf Ø oder 1 abfragen.

- ATN – Attention

  Der Rechner hält diese Leitung auf logisch Ø, während er Adressen oder Befehl über die Datenleitungen sendet. Ist diese Leitung logisch 1, so befinden sich Daten auf diesen Leitungen.

- EOI – End or Identify

  Das EOI-Signal hat, wie der Name sagt, zwei Funktionen: 1. wird mit ihm das Ende einer Blockübertragung angezeigt und 2. wird es bei der Identifizierung von Geräten benutzt. Die letztere Anwendung ist allerdings bei Ihrem Computer nicht eingebaut.

Das „Ende"-Signal wird vom Rechner zusammen mit dem letzten Zeichen eines Datenblocks übertragen und dient dazu, eine Geräteadressierung wieder aufzuheben.

Folgende Kombinationen von ATN und EOI können auftreten:

ATN	EOI	
Ø	Ø	Normale Datenübertragung
Ø	1	Letztes Datenbyte eines Blockes
1	Ø	Übertragung einer Geräteadresse oder eines Gerätebefehls
1	1	Aufforderung zur Identifizierung nach einer SRQ-Anfrage (bei Ihrem Modell nicht verwendet)

Alle Daten-, Handshake- und Informationsleitungen haben TTL-Pegel.

3. Wieso versteht der Computer die Sprache BASIC

3.1 Die Sprache der Zentraleinheit

Die Zentraleinheit des Computers ist ein Mikroprozessor Typ 6502. Es ist einer der schnellsten zur Zeit auf dem Markt befindlichen 8 Bit Prozessoren.

Der Befehlsvorrat dieses Prozessors ist jedoch nicht mit dem Befehlsvorrat von BASIC zu verwechseln. Eine BASIC-Zeile kann viele tausend Prozessorbefehle zur Folge haben.

BASIC nennt man eine höhere Programmiersprache. Zum Unterschied zur Maschinensprache versteht fast jeder Rechner BASIC, während die Maschinensprache speziell auf den Prozessor zugeschnitten ist. Der Computer interpretiert die BASIC-Befehle, er führt sie jedoch nicht direkt aus. Ihr Rechner hat also einen BASIC-Interpreter. Zusätzlich ist es aber auch möglich, kleinere Programme direkt in der Maschinensprache zu schreiben. Wir wollen Ihnen nachfolgend ein kleines Beispiel dazu zeigen, ohne jedoch tiefer auf die Maschinensprache einzugehen.

Unser Maschinenprogramm soll uns eine beliebige Integerzahl in eine Binärzahl, so wie sie im Arbeitsspeicher steht, umwandeln und auf dem Bildschirm anzeigen.

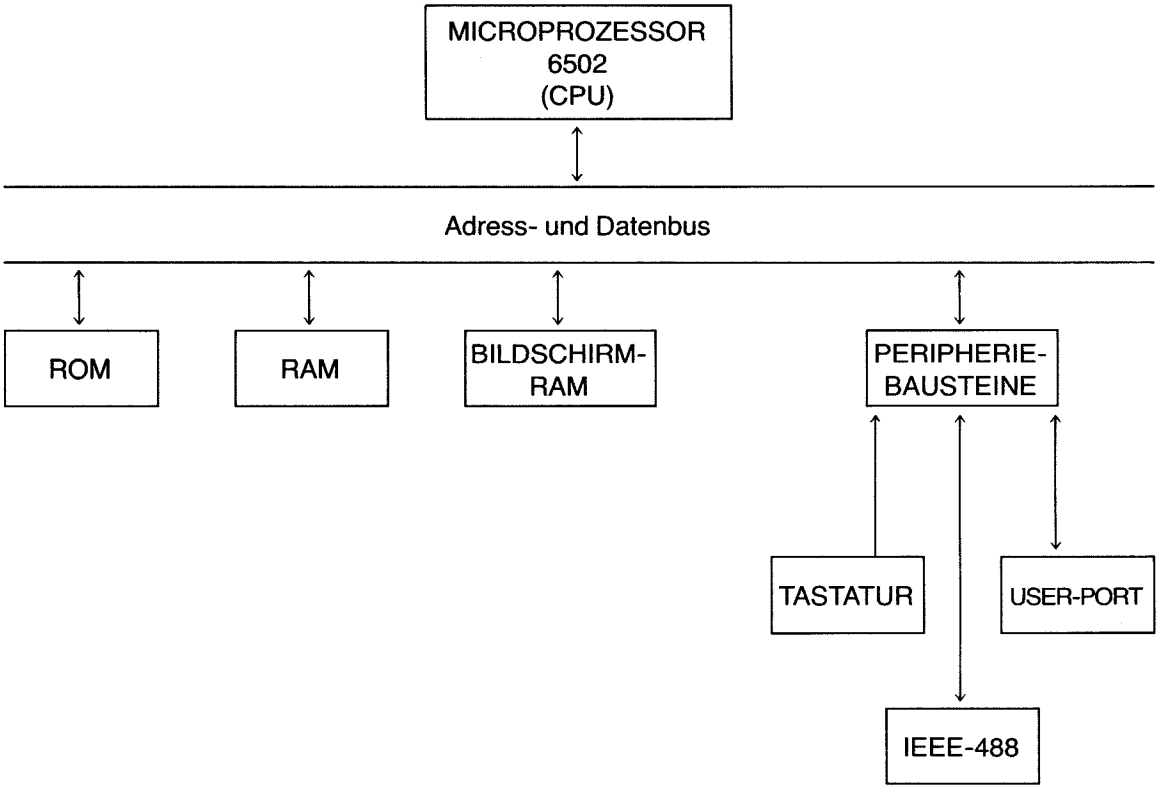
3.2. Der innere Aufbau des cbm

Nachfolgend sehen Sie ein Blockdiagramm Ihres Computers.

Alle Schnittstellen, auch für IEC-Bus, Kassette, Bildschirm, Tastatur sind mit dem Adressbus des Prozessors verbunden.

Es ist tatsächlich so, daß alle Schnittstellen im Rechner wie Speicherstellen behandelt werden, d. h. der IEC-Bus ist nicht hardware – sondern softwaremäßig realisiert. Ebenso wird die Dekodierung der Tastatur per Programm durchgeführt.

Zwar ist die Entwicklung des Betriebssystems für solch eine Lösung nicht gering, gleichzeitig ist aber ein solches System sehr flexibel, da lediglich eine Änderung des Programmes nötig ist, um z. B. eine völlig andere Tastatur an den Rechner anzuschließen.



3.3. Der logische Aufbau des Speichers

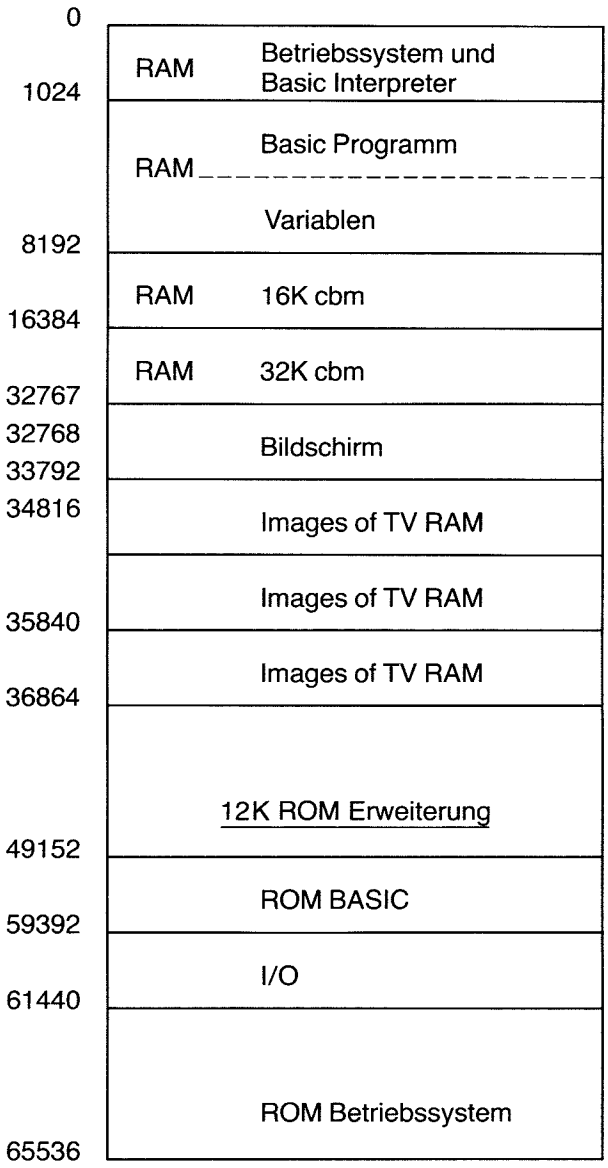
Aufgebaut ist der Adressbereich des Rechners in 256 Blöcke von je 256 Byte. Das sind insgesamt 64 KB, die der Prozessor 6502 addressieren kann.

Die Unterteilung in 256 Blöcke, sogenannte Pages (Seiten) entsteht dadurch, daß der Adressbus in 2 getrennte Busse von je 8 bit aufgeteilt ist. Die Pages Ø, 1 und 255 nehmen in diesem Zusammenhang eine Sonderstellung ein.

Page Ø dient hauptsächlich zur schnellen Verarbeitung und Speicherung von Daten. Zur Speicherung von Programmen ist sie weniger geeignet.

Page 1 ist prozessorbedingt zum Zwischenspeichern von Adressen und Registerdaten reserviert. In diesen Bereich sollten keine Daten willkürlich geschrieben werden, da der Rechner diesen Bereich permanent benötigt und verwaltet.

In Page 255 sind die letzten 6 Bytes als Sprungadressen für Unterbrechungen oder Einschaltvorgänge reserviert. Da dieser Bereich jedoch ohnehin durch ein ROM belegt ist, haben Sie keinerlei Einfluß auf diese Speicherstellen.



3.3.1 DER BILDSCHIRMBEREICH

Ab der Speicherstelle 32768 ist der Bildschirminhalt abgespeichert. Es gibt nun die verschiedensten Möglichkeiten, eine Information anzuzeigen. Einige davon stellen wir Ihnen nachfolgend vor. Jede Möglichkeit hat ihre Vor- und Nachteile. Sie sollten sich je nach Aufgabenstellung für die eine oder andere Möglichkeit entscheiden, oder eine Kombination wählen.

a) Relative Steuerung durch Print und Cursorsteuertasten sowie TAB ( ) und SPC ( ).

Die Befehle TAB ( ) und SPC ( ) sind in Kapitel 1.2.5 beschrieben. Zusätzlich haben Sie die Möglichkeit, den Cursor per Programm genauso wie von Hand zu steuern.

Beispiel: 10 REM " ● " = CURSOR CLR/HOME
12 REM "Q" = CURSOR NACH UNTEN
14 REM "] " = CURSOR NACH RECHTS
16 PRINT "● QQJA"
18 END

Sie geben also nach dem Anführungszeichen die gleiche Tastenfolge vor wie von Hand und beenden die Zeichenfolge mit einem 2. Anführungszeichen.

Bemerkung:
Alle Steuertasten mit Ausnahme der DELETE-Taste werden innerhalb von Anführungszeichen nicht ausgeführt sondern für eine spätere Ausführung abgespeichert.

b) Absolute Cursorsteuerung über Unterprogramm

Mit Hilfe des nachstehend aufgeführten Unterprogramms haben Sie die Möglichkeit, eine bestimmte Stelle im Bildschirm anzusteuern. Dazu geben Sie die Zeile in Variablen Z und die Spalte in der Variablen S an. Anschließend springen Sie mit GOSUB an den Anfang des Unterprogrammes.

Unterprogramm:

```
100 REM UNTERPROGRAMM zur Cursor-Steuerung
110 POKE 198,S-1
120 POKE 215,Z-1
130 POKE 197, INT (40* (Z-1)/256) + 128
140 POKE 196, (40*(Z-1)) AND 255
150 RETURN
```

Beispiel:
```
10 PRINT
20 Z = 4 : S = 3 : GOSUB 100
30 PRINT "A"
40 END
```

c) Direktes Schreiben in den Bildschirm

Die 1000 Plätze des Bildschirms entsprechen den RAM-Speicherplätzen 32768 (erstes Zeichen) bis 33767 (1000. Zeichen).
Die restlichen 24 Plätze im RAM (33768 bis 33791) können vom Benutzer belegt werden, werden aber mit CLR (Bildschirm löschen) jedesmal mitgelöscht.

Mit dem POKE-Befehl ist es möglich, eine bestimmte Stelle im Bildschirm zu beschreiben oder mit dem PEEK-Befehl zu lesen.

Beispiel: POKE 33111, 160

Beachten Sie jedoch, daß der Code im Bildschirm nicht mit dem ASCII-Code identisch ist. Siehe 2.1.

Zum Abschluß ein Vergleich der 3 aufgeführten Verfahren:

	Verfahren 1 PRINT/SPC/Cursor Steuertasten	Verfahren 2 Unterprogramm Vorgabe Z/S	Verfahren 3 POKE-Befehl
Geschwindigkeit	mittel	langsam	schnell
Bildschirmgüte	mittel	gut	schlecht
CODE	ASCII	ASCII + Zeile + Spalte	Bildschirmcode
Steuer- geschwindigkeit	langsam	mittel	schnell
Ansprechen eines best. Punktes	langsam	mittel	schnell
Anwendung:	Schreiben	Einschreiben in vorgegebene Formen	spez. Anwendungen

3.3.2 EIN- AUSGABEBEREICHE

Von Speicherstelle 59392 bis 61440 befinden sich die Ein-Ausgabebereiche für die Peripherie (Tastatur, Kassette, IEC-Bus, 8 Bit-Port sowie Steuerteile des Bildschirms), wobei bei manchen Bytes jedes Bit eine andere Funktion haben kann.

ACHTUNG:
Das willkürliche Beschreiben oder Lesen dieser Speicherstellen mit POKE oder PEEK kann unkontrollierbare Erscheinungen des Rechners zur Folge haben. Dieser Zustand kann nur durch Aus- und Einschalten beendet werden.

CHIP: 6520

Registername	Hex. Adresse	Dez. Adresse
PIAL	$E810	59408
PIAL 1	$E811	59409
PIAK	$E812	59410
PIAS	$E813	59411

CHIP: 6520

Registername	Hex. Adresse	Dez. Adresse
IEEI	$E820	59424
IEEIS	$E821	59425
IEEO	$E822	59426
IEEOS	$E823	59427

CHIP: 6522

Registername	Hex. Adresse	Dez. Adresse
PIA	$E84Ø	59456
SYNC	$E841	59457
P2DB	$E842	59458
P2DA	$E843	59459
TIL	$E844	5946Ø
TIH	$E845	59461
TILL	$E846	59462
TILH	$E847	59463
T2L	$E848	59464
T2H	$E849	59465
SR	$E84A	59466
ACR	$E84B	59467
PCR	$E84C	59468
IFR	$E84D	59469
IER	$E84E	5947Ø
SYNC1	$E84F	59471

Näheres siehe: cbm 3016 und 3032 ASSEMBLER LISTING (erhältlich beim Commodore-Vertragshändler).

3.3.3 DER BEREICH DES BETRIEBSSYSTEMS

Im Speicherbereich 61440–65535 befinden sich die Maschinensprachenprogramme für die spezielle Bedienung Ihrer Rechnerkonfiguration wie z. B. Anzeigen eines Zeichens im Bildschirm oder Empfangen eines Zeichens von der Tastatur.

Ebenfalls befinden sich in diesem Bereich gewisse Testroutinen, die die ordnungsgemäße Arbeit Ihres Rechners überprüfen.

Beispiel: SYS (65490)
(Drucken eines Zeichens im Bildschirm)

Beispiel: SYS (65487)
(Eingabe von Zeichen über Tastatur und Drucken dieser Zeichen im Bildschirm bis RETURN gedrückt wird.)

3.3.4 DER BEREICH DES BASIC INTERPRETERS

Der Basic-Interpreter ist das A und O Ihrer ganzen Anlage. Durch ihn erst ist es möglich, daß Sie einen Sinus berechnen können, oder bestimmte Werte abspeichern und jederzeit wiederfinden. Die Programme dieses Interpreters befinden sich an den Speicherstellen 49152 bis 57343. Diesen Bereich können Sie im Gegensatz zu dem restlichen Speicher mit PEEK nicht abfragen . . . verständlich, da Erstellungszeit und Kosten immens waren. Das Copyright für diesen Interpreter liegt bei der Firma Microsoft.

3.3.5 Dieser Interpreter braucht natürlich auch einen gewissen Bereich, wo er Daten und Informationen zwischenspeichern kann. Er benutzt dazu die Speicherstellen Ø–1Ø23. Dieser Bereich steht nicht frei zur Verfügung. Aus diesem Grunde haben Sie bei Ihrem Modell cbm auch nicht 32768 Bytes frei, sondern 32768 – 1Ø25 = 31743 Bytes.

Im folgenden finden Sie eine Liste der Belegung der Speicherstellen Ø–1Ø23.

cbm SPEICHERBELEGUNG: PAGE Ø

(Nicht aufgeführte Adressen werden gebraucht, haben aber keine eindeutige Funktion.)

! VORLÄUFIGE SPEZIFIKATION – ÄNDERUNGEN VORBEHALTEN!

Ø		Konstante $4C (JMP-Befehl in Maschinensprache)
1	2	Vektor für USR-Funktion L,H
3		Anfang-Trennungszeichen
5		Hilfszähler für BASIC
6		Flag für dimensionierte Variablen
7		Flag für Variablenart (Ø ≙ numerisch; 1 ≙ String)
8		Flag für Integervariablen
9		Flag für " und REM
10		Flag, ob Indizes erlaubt sind
11		Flag für INPUT oder READ
12		Flag für Vorzeichen von TAN
13		Flag, ob OUTPUT unterdrückt wird
14		Aktiver I/O-Kanal
15		Zeilenlänge des Terminals
16		Maximale Anzahl Zeichen pro BASIC-Zeile
17	18	Zeilen# (Zwischenspeicher)
19		Zeigt auf nächsten Descriptor
20	21	Zeiger auf den zuletzt benutzten String
22	29	Tabelle der Descriptoren für Variablen (je 2 Bytes)
30	31	Indirekter Index # 1,L,H
32	33	Indirekter Index # 2,L,H
34	39	Pseudoregister für Operanden von Funktionen
40	41	Zeiger auf den Beginn der BASIC-Befehle; L,H
42	43	Zeiger auf den Beginn der Variablentabelle; L,H
44	45	Zeiger auf den Beginn der Array-Tabelle; L,H
46	47	Zeiger auf das Ende der Variablentabelle; L,H
48	49	Zeiger auf Ende des letzten Strings; L,H
50	51	Zeiger auf den Beginn der Strings; L,H
52	53	Zeiger auf höchste RAM-Adresse; L,H
54		ist 2, wenn ein direkter Befehl ausgeführt wird
54	55	# der gerade ausgeführten Zeile; L,H
56	57	Zeilen# für CONT; L,H
58	59	Zeiger auf den nächsten Befehl; L,H
60	61	# der DATA-Zeile bei ERROR; L,H
62	63	Zeiger auf nächstes DATA; L,H
64	65	Herkunft des Input; L,H
66	67	Name der laufenden Variablen (1. und 2. Zeichen)
68	69	Zeiger auf die laufenden Variablen; L,H
70	71	Zeiger auf die laufende FOR-NEXT-Variable; L,H
72	73	Zeiger auf den laufenden Operator; L,H

74		Maske für laufenden Operator (>, = ,<)
75	76	Zeiger auf DEFFN; L,H
77	78	Zeiger auf den String-Descriptor; L,H
79		Länge des obigen Strings
80		Konstante
81		$2C (Konstante für INPUT)
82	83	Vektor für Funktionen; L,H
84	89	Fließkommaakkumulator #3
90	91	Zeiger #1 für Blocktransfer; L,H
92	93	Zeiger #2 für Blocktransfer; L,H
94	99	Fließkommaakkumulator #1 (u. a. für USR)
100		Vorzeichen der Mantisse aus Akku #1
101		Zählt die zum Normalisieren von Akku #1 nötigen Shifts
102	107	Fließkommaakkumulator #2
108		Overflowbytes des Arguments aus Akku #2
109		Vorzeichen der Mantisse aus Akku #2
110	111	Zeiger auf ASCII-Darstellung des Akku #2; L,H
112	135	CHRGET-Routine. Holt nächstes Zeichen des BASIC-Textes
118		Label CHRGOT
119	120	Zeiger auf nächstes Zeichen des BASIC-Textes
136	140	Nächste Zufallszahl
141	143	24h-Uhr in $^1/_{60}$ Sekunden
144	145	IRQ-Vektor (interrupt request) für internen Interrupt
146	147	BRK-Vektor
148	149	NMI-Vektor (nicht maskierbarer Interrupt)
150		I/O-Status
151		Letzte Taste
152		Flag für SHIFT (1 bei gedrückter Taste, sonst Ø)
153	154	Korrekturfaktor für die Uhr
155	156	*
157		Flag für VERIFY
158		Anzahl Bytes im Tastaturpuffer (ab 623)
159		RVS-Flag
160	166	*
167		Flag für Cursor ein/aus
168		Zähler für die Blinkdauer des Cursors
169		*
170		Zwischenspeicher für Zeichen während Blinken des Cursors
171	173	*
174		Zeiger in die Tabelle der logischen Files
175		Ersatzparameter (Geräte# für INPUT)
176		Ersatzparameter (Geräte# für OUTPUT)
177		Paritätsbyte für Band
178	185	*
186		Zur Synchronisation beim Lesen
187	188	Zeiger auf den aktiven Recorder
189		*
190	192	Fehlerkorrektur
193		*
194		Flag für Lesen von Kassette
195		Dauer der "shorts" vor dem Schreiben von Daten
196	197	Adresse der Position des Cursors
198		Cursorspalte
199	200	Startadresse für LOAD
201	202	Endadresse für LOAD
203	204	*
205		Flag für "
206	208	*
209		Länge des laufenden Files
210		logische # des laufenden Files

211		Sekundäradresse des laufenden Files
212		Primäradresse des laufenden Files
213	217	*
218	219	Adresse des laufenden Filenames
220	221	*
222		Blockzähler für Recorder (Lesen)
223		*
224	248	Tabelle der LSB's der Zeilenanfänge (Bildschirm)
249	255	*

cbm PAGE 1

256-317 (dezimal) werden zur Fehlerkorrektur beim Lesen vom Band benutzt, sowie als Puffer für Zeilenumwandlungen. Der Rest von Page 1 dient als Speicher im Zusammenhang mit GOSUB und FOR-NEXT, sowie als Hardware-Stack.

cbm PAGE 2

(Nicht aufgeführte Adressen werden gebraucht, haben aber keine eindeutige Funktion).
! VORLÄUFIGE SPEZIFIKATION – ÄNDERUNGEN VORBEHALTEN !

512	592	BASIC-Input-Puffer
512	513	Programmzähler des 6502
514		Statusregister des Prozessors
515		Akkumulatorinhalt
516		X Register
517		Y Register
518		Stapelzeiger
519	520	IRQ-Vektor für externen Interrupt
593	602	Tabelle der Filenummern
603	612	Tabelle der Gerätenummern
613	622	Tabelle der Sekundäradressen
623	633	Tastaturpuffer (Anzahl Zeichen steht in 158)
634	825	Puffer für Recorder#1
826	1017	Puffer für Recorder#2
1018	1023	*

3.3.6 SPEICHERUNG VON PROGRAMMEN

Eine Programmzeile besteht aus Zeilennummer und Befehlen. Alle Programmzeilen werden in Reihenfolge der Zeilennummer von 1Ø24 an aufwärts folgendermaßen gespeichert (n ist die Zeilennummer, (n + x) ist die auf n folgende Zeilennummer):

Das erste Byte ist Null. (A_n).

Es folgen 2 Bytes (B_n; C_n). Sie stellen die Verbindung zur nächsten Programmzeile her. Der Wert $B_n + 256 * C_n$ gibt die Adresse an, in der die Größe B_{n+x} steht. Sind B_n und C_n beide Null, dann ist n die letzte Zeilennummer des Programms.

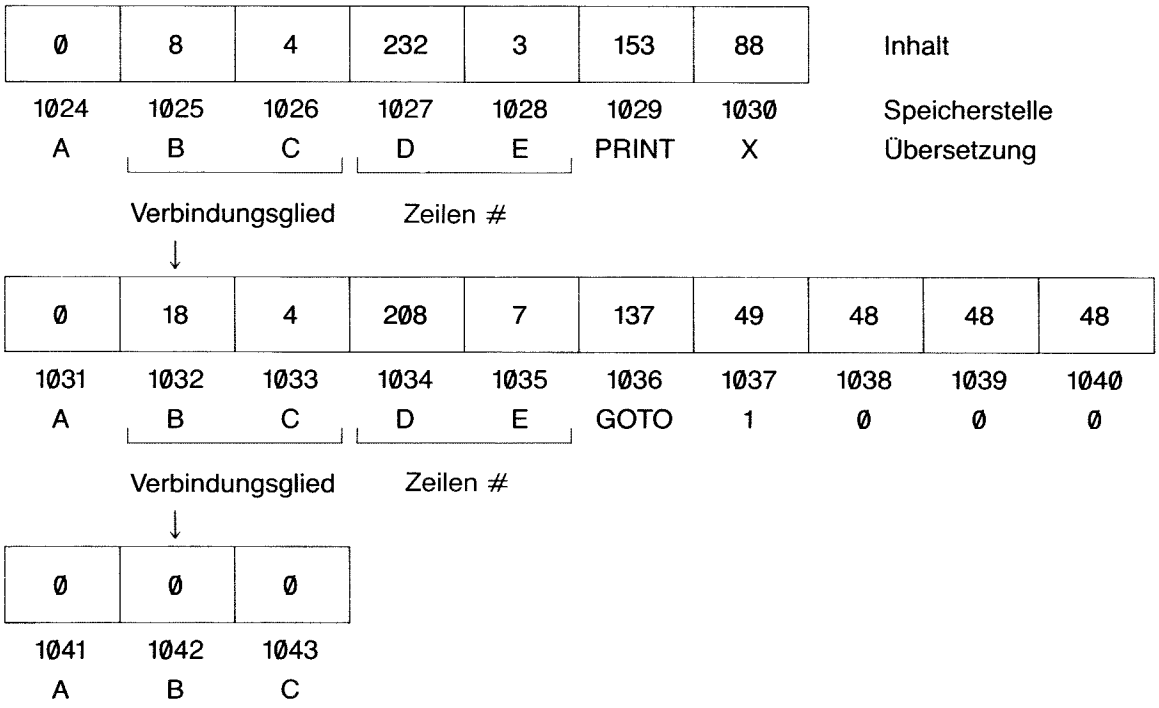
Die nächsten zwei Bytes (D_n; E_n) geben die Zeilennummer an: Zeilennummer = $D_n + 256 * E_n$.

Es folgen die BASIC-Befehle (codiert in 1 Byte pro Befehl bzw. 1 Byte pro Zeichen. Die Codeliste steht im Anhang unter Interpretercode).

Beispiel: Das Programm

```
1000 PRINTX
2000 GOTO 1000
```

wird folgendermaßen gespeichert:



3.3.7 Es stellt sich nun die Frage, wie der Rechner die Variablen verwaltet, die Sie eingeben.

Nehmen wir an, Sie haben den Rechner eingeschaltet und geben ein:

```
A$ = "ICH BIN DAS MODELL CBM"
```

Der Rechner meldet:

```
READY
```

Die Stelle, wo der Text „ICH BIN DAS MODELL CBM" steht, können Sie nun folgendermaßen finden:

```
PRINT PEEK (42) + PEEK (43) * 256 + 2
```

Antwort: 1029

An der Speicherstelle 1030 steht in diesem Fall, wie lang der eingegebene Text ist. Die beiden nachfolgenden Speicherstellen geben an, wo der Text steht.

```
? PEEK (1029), PEEK (1030) + PEEK (1031) * 256
```

Antwort: 23 8169

Der Text ist also 23 Stellen lang und steht ab Speicherstelle 8169.

Geben Sie ein:

```
FOR I = 8169 TO 8169 + 22: PRINT CHR$ (PEEK(I));:NEXTI
```

Antwort: ICH BIN DAS MODELL CBM

Sie haben gesehen, welche Vorgänge der BASIC-Interpreter durchführt, um eine Variable wiederzufinden.

In den Speicherstellen 124, 125 und 1031, 1032 stehen in unserem Beispiel sogenannte Pointer (Zeiger).

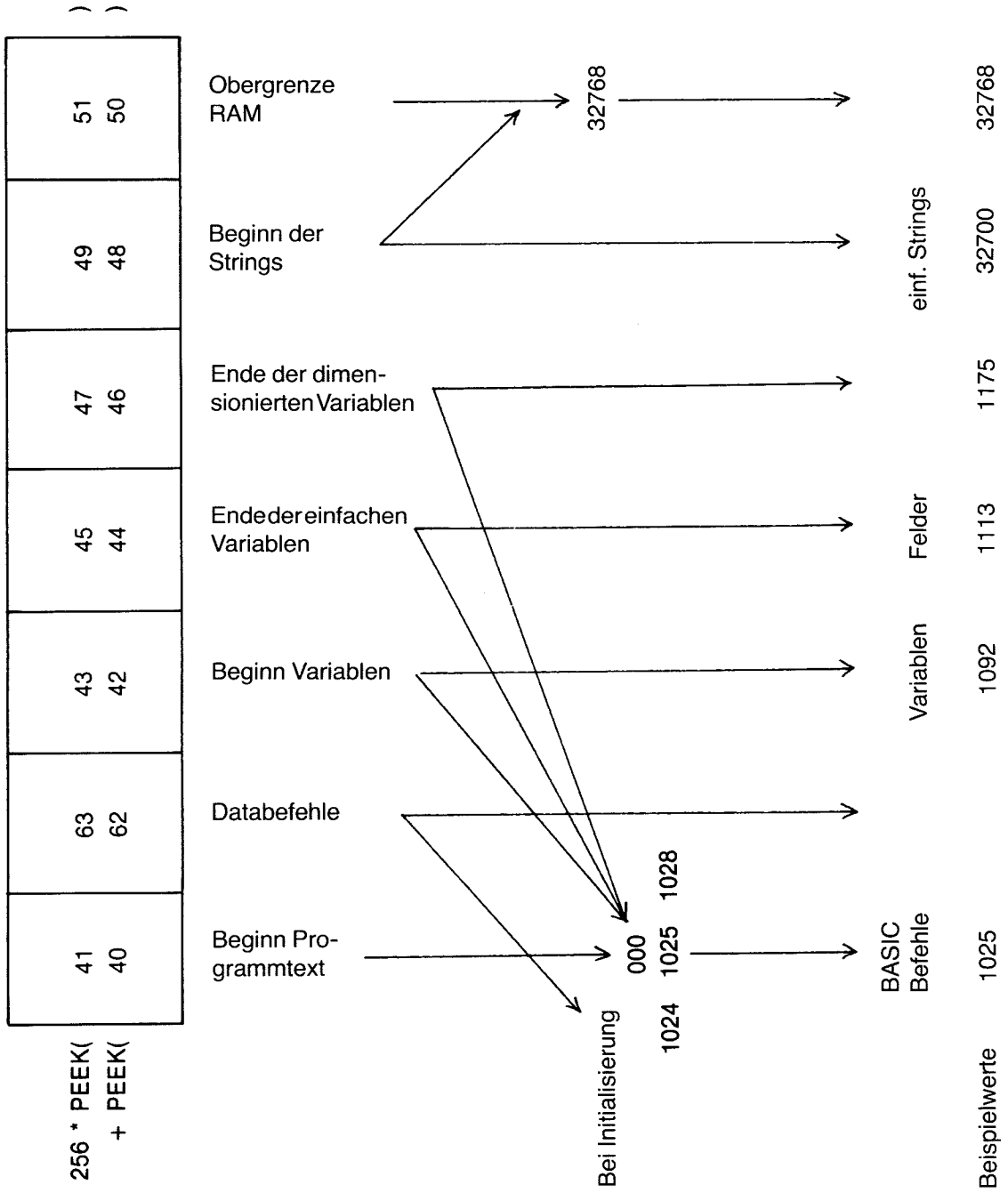
Pointer sind Informationen, die angeben, wo eine gesuchte Information steht.

1-Byte-Pointer können Werte von 0–255 annehmen. Um beispielsweise auf die Speicherstelle 8169 zeigen zu können, benötigt man 2-Byte-Pointer. 2-Byte-Pointer sind wie folgt verschlüsselt:

Speicherstelle = X + 256 * (x + 1)
X und x + 1 sind die Inhalte zweier aufeinanderfolgender Speicherstellen.

Nachfolgend eine Tabelle des Speicherplatzes und dessen Aufteilung mit Angabe der Pointer für diese Aufteilung.

*RAM-ZUTEILUNG UND POINTER



3.4 WIE EIN PROGRAMM ABLÄUFT

Sobald Sie RUN eingegeben haben, beginnt der Rechner die Abarbeitung der Befehle in aufsteigender Reihenfolge. Einen Befehl erkennt der Rechner an seiner Abkürzung als Zahl von 128 bis 202 (siehe Interpretercode). Findet er einen solchen Befehl, so führt er diesen Befehl aus. Wenn er einen Befehl ausführen soll, so prüft er, ob weitere Informationen zu diesem Befehl nötig sind.

Wenn ja, so holt er sich diese Informationen aus dem Speicher. Wenn nein, so führt er den angegebenen Befehl sofort aus. Sollte der Interpreter hierbei einen Fehler feststellen, so zeigt er diesen an und beendet seine Arbeit.

Stellt nun der Interpreter eine Variable fest, so sucht er diese in seiner Variablenliste. Findet er sie dabei nicht, so legt er sie neu an. Dabei verschiebt er mitunter seinen kompletten Speicher mehrere hundert Mal. Hat er einmal eine Variable angelegt, so findet er sie immer wieder. Auch nach Beendigung des Programms stehen die Variablen zur Verfügung, bis Sie entweder mit RUN das Programm neu starten oder das Programm ändern.

4. TIM: MONITOR FÜR DIE MASCHINENSPRACHE

TIM (Terminal Interface Monitor) wurde für die Mikroprozessoren der Serie 65xx entwickelt.

TIM BEFEHLE

M	display memory;	Anzeige des Speicherinhalts
R	display register;	Anzeige der Registerinhalte
G	begin execution;	Ausführung eines Programmes in Maschinensprache
X	exit to BASIC;	Rückkehr zu BASIC
L	load;	Laden eines Programms von Kassette
S	save;	Überspielen eines Programms auf Kassette

TIM UNTERPROGRAMME

JSR	WRT	$FFD2	type a character;	Ausgabe eines Zeichens
JSR	RDT	$FFCF	input a character;	Eingabe eines Zeichens
JSR	GET	$FFE4	get a character;	Holen eines Zeichens
JSR	CRLF	$FDD0	type a CR;	Schreibe CR = CHR$(13)
JSR	SPACE	$FDCD	type a space;	Schreibe Zwischenraum = CHR$(32)
JSR	WROB	$E775	type a byte;	Schreiben eines Bytes
JSR	RDOB	$E7B6	read a byte;	Lesen eines Bytes
JSR	HEXIT	$E7E0	ASCII to hex in A;	Umwandlung ASCII-Code nach hexadezimal

TIM ZAHLENSYSTEM

Generell hexadezimale Ein- und Ausgabe.

ANZEIGE DES SPEICHERINHALTS .M XXXX YYYY

Anfangsadresse (XXXX) und Endadresse (YYYY) müssen vollständig (als vierziffrige Hex-Zahlen) 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

```
          0  1  2  3  4  5  6  7
.:C000 1D C7 48 C6 35 CC EF C7
.:C008 C5 CA DF CA 70 CF 23 CB
.:C010 9C C8 9C C7 74 C7 1F C8
```

ANZEIGE DER REGISTERINHALTE .R

Die Inhalte der Register der CPU (Central processing unit, 65Ø2) werden angezeigt:

PC Programmzähler (program counter)
IRQ Interruptvektor
SR Statusregister des Prozessors
AC Akkumulator
XR Indexregister X
YR Indexregister Y
SP Stapelzeiger (stack pointer)

Beispiel: .R

PC	IRQ	SR	AC	XR	YR	SP
Ø4Ø1	E62E	32	04	5E	ØØ	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 (von TIM nach BASIC) wiederhergestellt.

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 C38B

EXIT-RÜCKKEHR ZU BASIC .X

. X bewirkt einen Rücksprung nach BASIC. Die Speicherinhalte bleiben dabei unverändert, und BASIC befindet sich im selben Zustand wie vor dem Aufruf des Monitors.

LOAD – EINLESEN DES PROGRAMMS VON KASSETTE .L "NAME", XX

XX steht für die Nummer des Peripheriegeräts, Gerätenummer und Filename müssen angegeben werden. Das OS (Operating system) reagiert wie bei BASIC. Die mit dem SAVE-Befehl definierten Speicheradressen werden geladen.

Unterprogramme in Maschinensprache können auch von BASIC 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",Ø1
PRESS PLAY ON TAPE #1
OK
SEARCHING FOR NAME
FOUND NAME
LOADING
```

SAVE – SPEICHERN EINES PROGRAMMS .S "NAME",XX, YYYY, ZZZZ

XX steht für die Nummer des Peripheriegeräts (siehe . L). YYYY ist die hexadezimale Anfangsadresse des aufzunehmenden Programms und ZZZZ die Endadresse plus eins.

Beispiel:
```
.S "NAME", Ø1, Ø33A, Ø76B
PRESS PLAY & RECORD ON TAPE # 1
OK
WRITING NAME
```

Ø76B ist Endadresse plus eins, das letzte Datenbyte befindet sich also in Ø76A.

BREAK UND INTERRUPTS

Der Maschinenbefehl BRK ($ØØ) 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 $Ø21B und $Ø21C 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 Befehle vom Operator.

Der oben erwähnte Vektor kann auch verändert werden, etwa um auf eine spezielle Routine zu verzweigen.

Beachten Sie: 1) Nach einem BRK, der auf TIM verzweigt, zeigt der PC auf das dem BRK-Befehl folgende Byte.

2) Bei einem BRK, der nicht auf TIM verzweigt, zeigt der PC nach der Rückkehr (via RTI) auf das zweite Byte nach dem BRK-Befehl.

AUFRUF VON TIM

Nach dem Laden des Programms kann TIM entweder mit dem Befehl

SYS 64785

oder mit dem Befehl

SYS (1Ø24)

aufgerufen werden. Der erste Fall stellt einen "entry by call" dar, der zweite einen "entry via breakpoint". (Die Speicherzelle 1Ø24 enthällt Null).

BEISPIEL:

Das folgende Programm mit dem Namen CHSET schreibt 64 ASCII-Zeichen auf dem Bildschirm. Es soll im Puffer für Recorder # 2 gespeichert werden:

```
                    * = $33A
                    CRLF = $FDØØ
                    WRT = $FFD2

33A  20 ØØ FD CHSET JSR CRLF
33D  A2 2Ø          LDX # $2Ø
33F  8A       LOOP  TXA
340  20 D2 FF       JSR WRT
343  E8             INX
344  EØ 6Ø          CPX #$6Ø
346  DØ F7          BNE LOOP
348  ØØ             BRK
349  4C 3A Ø3       JMP CHSET
```

Um dieses Programm einzugeben, werden mit dem Befehl

.M Ø33A, Ø34B

die betreffenden Speicherinhalte zur Anzeige gebracht und geändert . Schließlich erhält man die Anzeige:

```
.M Ø33A, Ø34B
.: Ø33A ØØ FD Ø4 A2 2Ø 8A 2Ø D2
.: Ø342 FF E8 EØ 6Ø DØ F7 ØØ 4C
.: Ø34A 3A Ø3
```

Der Befehl:

```
. G Ø33A
```

bringt das Programm jetzt zur Ausführung:

```
!"H$%& ( ) ↓+ , - . /Ø123456789 : ; <=>?§ABCDEFG
HIJKLMNOPQRSTUVWXYZ[/]†←!
```

Beachten Sie den Inhalt des Programmzählers (siehe BREAK und INTERRUPTS). Es ist jetzt möglich, ohne Angabe der Anfangsadresse das Programm mit dem Befehl:

```
. G
```

nochmals zu starten.

Um CHSET von BASIC aus aufrufen zu können, ersetzt man zunächst die BRK-Anweisung in $348 durch ein RTS (return from subroutine). $348 wird also von ØØ auf 6Ø geändert.

CHSET kann nun von BASIC aus mit SYS826 aufgerufen werden.

Will man den Befehl USR verwenden, so ändert man den zugehörigen USR-Vektor in $1 und $2 dermaßen, daß er auf die Subroutine CHSET zeigt:

```
. M ØØØ1, ØØØ2
              Ø   1 . . . .
  ØØØ1   3A  Ø3 . . . .
```

Nach der Rückkehr zu BASIC (. X) kann CHSET somit jederzeit mit SYS826 oder A = USR(Ø) aufgerufen werden.

```
LINE # LOC     CODE        LINE

2514  FD11                ;COPYRIGHT 1978 BY
2515  FD11                ;COMMODORE INTERNATIONAL LIMITED



2517  FD11                NCMDS   =8
2518  FD11  A9 43         CALLE   LDA #'C          ;CALL ENTRY
2519  FD13  85 B5                 STA TMPC
2520  FD15  D0 16                 BNE B3
2521  FD17  A9 42         BRKE    LDA #'B          ;BREAK ENTRY
2522  FD19  85 B5                 STA TMPC
2523  FD1B  D8                    CLD
2524  FD1C  4A                    LSR A            ;C SET FOR PC CORRECTION
2525  FD1D  68                    PLA
2526  FD1E  8D 05 02              STA YR           ;SAVE Y
2527  FD21  68                    PLA
2528  FD22  8D 04 02              STA XR           ;SAVE X
2529  FD25  68                    PLA
2530  FD26  8D 03 02              STA ACC          ;SAVE ACCUMULATOR
2531  FD29  68                    PLA
2532  FD2A  8D 02 02              STA FLGS         ;SAVE FLAGS
2533  FD2D  68            B3      PLA
2534  FD2E  69 FF                 ADC #$FF         ;PC-1 FOR BREAK
2535  FD30  8D 01 02              STA PCL
2536  FD33  68                    PLA
2537  FD34  69 FF                 ADC #$FF
2538  FD36  8D 00 02              STA PCH
2539  FD39  A5 90                 LDA CINV         ;SAVE CUURENT IRQ VECTOR
2540  FD3B  8D 08 02              STA INVL
2541  FD3E  A5 91                 LDA CINV+1
2542  FD40  8D 07 02              STA INVH
2543  FD43  BA                    TSX              ;SAVE CURRENT STACK POINTER
2544  FD44  8E 06 02              STX SP
2545  FD47  58                    CLI              ;CLEAR INTERRUPT DISABLE
2546  FD48  20 D0 FD      B5      JSR CRLF         ;PRINT ENTRY DATA
2547  FD4B  A6 B5                 LDX TMPC         ;TYPE OF ENTRY (B OR C)
2548  FD4D  A9 2A                 LDA #'*
2549  FD4F  20 84 E7              JSR WRTWO        ;WRITE '*C' OR '*B'
2550  FD52  A9 52                 LDA #'R          ;DISPLAY REGISTERS COMMAND
2551  FD54  D0 1A                 BNE S0           ;SKIP TO INTERPRET COMMAND

2553  FD56  A9 02         STRT    LDA #2           ;USER COMMAND INPUT
2554  FD58  85 77                 STA TXTPTR       ;COMING FROM TEXT BUFFER
2555  FD5A  A9 00                 LDA #0
2556  FD5C  85 DE                 STA WRAP         ;ADDR WRAP AROUND FLAG
2557  FD5E  A2 0D                 LDX #CR          ;START PROMPT WITH CRLF
2558  FD60  A9 2E                 LDA #'.          ;A PROMPTING ' '
2559  FD62  20 84 E7              JSR WRTWO
2560  FD65  20 EB E7      ST1     JSR RDOC         ;INPUT COMMAND LINE
2561  FD68  C9 2E                 CMP #'.          ;IGNORE PROMPTING ' '
2562  FD6A  F0 F9                 BEQ ST1
2563  FD6C  C9 20                 CMP #$20         ;SPAN BLANKS
2564  FD6E  F0 F5                 BEQ ST1
```

```
LINE #  LOC     CODE        LINE

2566   FD70   A2 07        S0      LDX #NCMDS-1     ;LOOKUP COMMAND
2567   FD72   DD E0 FD     S1      CMP CMDS,X
2568   FD75   D0 0B                BNE S2
2569   FD77   86 B4                STX SAVX         ;INDEX OF COMMAND IN TABLE
2570   FD79                ;INDIRECT JMP FROM TABLE BY
2571   FD79                ; PUSHING TARGET ADDRESS-1
2572   FD79                ; THEN RTS
2573   FD79   BD E8 FD              LDA ADRH,X
2574   FD7C   48                    PHA
2575   FD7D   BD F0 FD              LDA ADRL,X
2576   FD80   48                    PHA
2577   FD81   60                    RTS
2578   FD82   CA           S2      DEX
2579   FD83   10 ED                 BPL S1           ;LOOP FOR ALL COMMANDS

2581   FD85   6C FA 03              JMP (USRCMD)     ;ALLOW USER COMMANDS




2583   FD88   A5 FB        PUTP    LDA TMP0
2584   FD8A   8D 01 02              STA PCL
2585   FD8D   A5 FC                 LDA TMP0+1
2586   FD8F   8D 00 02              STA PCH
2587   FD92   60                    RTS

2589   FD93                ;DISPLAY MEM SUBR. SET AR=NUMBER
2590   FD93                ;OF MEMORY BYTES DISPLAYED
2591   FD93                ;TMP0=ADR OF MEM DISPLAYED
2592   FD93                ;
2593   FD93   85 B5        DM      STA TMPC
2594   FD95   A0 00                 LDY #0
2595   FD97   20 CD FD     DM1     JSR SPACE        ;WR N BYTES
2596   FD9A   B1 FB                 LDA (TMP0),Y     ;(TMP0)=ADR
2597   FD9C   20 75 E7              JSR WRDB
2598   FD9F   20 D5 FD              JSR INCTMP
2599   FDA2   C6 B5                 DEC TMPC
2600   FDA4   D0 F1                 BNE DM1
2601   FDA6   60                    RTS
2602   FDA7                ;READ AND STORE BYTE.
2603   FDA7                ;NO STORE IF SPACE OR TMPC = 0

2605   FDA7   20 B6        BYTE    JSR RDDB
2606   FDAA   90 0D                 BCC BY3          ;SPACE
2607   FDAC   A2 00                 LDX #0           ;STORE BYTE
2608   FDAE   81 FB                 STA (.MP0,X)
2609   FDB0   C1 FB                 CMP (TMP0,X)     ;VERIFY WRITE
2610   FDB2   F0 05                 BEQ BY3
2611   FDB4   68                    PLA              ;ERROR:CLEAR STACK
2612   FDB5   68                    PLA
2613   FDB6   4C F7 E7              JMP ERROPR
2614   FDB9   20 D5 FD     BY3     JSR INCTMP       ;INC TMP0 ADR
2615   FDBC   C6 B5                 DEC TMPC
2616   FDBE   60                    RTS
```

```
LINE # LOC      CODE        LINE

2618  FDBF  A9 02       SETR   LDA #<FLGS       ;SET TO ACCESS REGS
2619  FDC1  85 F8              STA TMP0
2620  FDC3  A9 02              LDA #>FLGS
2621  FDC5  85 FC              STA TMP0+1
2622  FDC7  A9 05              LDA #5
2623  FDC9  60                 RTS

2625  FDCA  20 CD FD    SPAC2  JSR SPACE
2626  FDCD  A9 20       SPACE  LDA #$20
2627  FDCF  2C                 .BYT $2C
2628  FDD0  A9 0D       CRLF   LDA #$D
2629  FDD2  4C D2 FF           JMP $FFD2

2631  FDD5              ;INCREMENT (TMP0,TMP0+1) BY 1
2632  FDD5  E6 F8       INCTMP INC TMP0          ;LOW BYTE
2633  FDD7  D0 06              BNE SETWR
2634  FDD9  E6 FC              INC TMP0+1       ;HIGH BYTE
2635  FDDB  D0 02              BNE SETWR
2636  FDDD  E6 DE              INC WRAP
2637  FDDF  60          SETWR  RTS
```

```
2639  FDE0              ;COMMAND AND ADDRESS TABLE

2641  FDE0  3A          CMDS   .BYT ':'          ;MODIFY MEMORY
2642  FDE1  3B                 .BYT ';'          ;ALTER REGISTERS
2643  FDE2  52                 .BYT 'R'          ;DISPLAY REGS
2644  FDE3  4D                 .BYT 'M'          ;DISPLAY MEMORY
2645  FDE4  47                 .BYT 'G'          ;START EXECUTION
2646  FDE5  58                 .BYT 'X'          ;WARM START BASIC
2647  FDE6  4C                 .BYT 'L'          ;LOAD MEMORY
2648  FDE7  53                 .BYT 'S'          ;SAVE MEMORY
2649  FDE8  FE          ADRH   .BYT >ZZ1
2650  FDE9  FE                 .BYT >ZZ2
2651  FDEA  FE                 .BYT >ZZ3
2652  FDEB  FE                 .BYT >ZZ4
2653  FDEC  FE                 .BYT >ZZ5
2654  FDED  FF                 .BYT >ZZ6
2655  FDEE  FF                 .BYT >ZZ7
2656  FDEF  FF                 .BYT >ZZ8
2657  FDF0  88          ADRL   .BYT <ZZ1
2658  FDF1  96                 .BYT <ZZ2
2659  FDF2  22                 .BYT <ZZ3
2660  FDF3  57                 .BYT <ZZ4
2661  FDF4  CE                 .BYT <ZZ5
2662  FDF5  86                 .BYT <ZZ6
2663  FDF6  10                 .BYT <ZZ7
2664  FDF7  10                 .BYT <ZZ8
```

```
LINE # LOC     CODE        LINE

2666  FDF8  0D          REGK   .BYT CR,'
2666  FDF9  20 20
2667  FDFD  20 50              .BYT ' PC  IRQ  SR AC XR YR SP'

2669  FE15  98          ALTRIT TYA
2670  FE16  48                 PHA
2671  FE17  20 D0 FD           JSR CRLF
2672  FE1A  68                 PLA
2673  FE1B  A2 2E              LDX #'.
2674  FE1D  20 84 E7           JSR WRTWO
2675  FE20  4C CA FD           JMP SPAC2

2677  FE23  A2 00       DSPLYR LDX #0
2678  FE25  BD F8 FD    D2     LDA REGK,X
2679  FE28  20 D2 FF           JSR $FFD2
2680  FE2B  E8                 INX
2681  FE2C  E0 1D              CPX #29
2682  FE2E  D0 F5              BNE D2
2683  FE30  A0 3B              LDY #';
2684  FE32  20 15 FE           JSR ALTRIT
2685  FE35  AD 00 02           LDA PCH
2686  FE38  20 75 E7           JSR WROB
2687  FE3B  AD 01 02           LDA PCL
2688  FE3E  20 75 E7           JSR WROB
2689  FE41  20 CD FD           JSR SPACE
2690  FE44  AD 07 02           LDA INVH
2691  FE47  20 75 E7           JSR WROB
2692  FE4A  AD 08 02           LDA INVL
2693  FE4D  20 75 E7           JSR WROB
2694  FE50  20 BF FD           JSR SETR
2695  FE53  20 93 FD           JSR DM
2696  FE56  F0 39              BEQ BEQS1

2698  FE59  20 EB E7    DSPLYM JSR RDOC
2699  FE5B  20 A7 E7           JSR RDOA          ;READ START ADR
2700  FE5E  90 34              BCC ERRS1         ;ERR IF NO SA
2701  FE60  20 97 E7           JSR T2T2          ;SA TO TMP2
2702  FE63  20 EB E7           JSR RDOC          ;SKIP DELIMITER
2703  FE66  20 A7 E7           JSR RDOA          ;READ END ADR
2704  FE69  90 29              BCC ERRS1         ;ERR IF NO EA
2705  FE6B  20 97 E7           JSR T2T2          ;SA TO TMP0, EA TO TMP2

2707  FE6E  20 81 F3    DSP1   JSR STOP1         ;TEST FOR STOP KEY
2708  FE71  F0 1E              BEQ BEQS1
2709  FE73  A6 DE              LDX WRAP
2710  FE75  D0 1A              BNE BEQS1
2711  FE77  38                 SEC               ;DOUBLE BYTE COMPARE
2712  FE78  A5 FD              LDA TMP2
2713  FE7A  E5 FB              SBC TMP0
2714  FE7C  A5 FE              LDA TMP2+1
2715  FE7E  E5 FC              SBC TMP0+1
2716  FE80  90 0F              BCC BEQS1         ;EA LESS THAN SA
2717  FE82  A0 3A              LDY #':
2718  FE84  20 15 FE           JSR ALTRIT
2719  FE87  20 6A E7           JSR WRDA
```

```
LINE # LOC      CODE        LINE

2720  FE8A  A9 08           LDA #8
2721  FE8C  20 93 FD        JSR DM          ;DISPLAY 8, INCR TMP0
2722  FE8F  F0 DD           BEQ DSP1

2724  FE91  4C 56 FD  BEQS1 JMP STRT

2726  FE94  4C F7 E7  ERRS1 JMP ERROPR

2728  FE97                  ;ALTER REGISTERS

2730  FE97  20 B6 E7  ALTR  JSR RDOB        ;SKIP 2 SPACES
2731  FE9A  20 A7 E7        JSR RDOA        ;CY=0 IF SP
2732  FE9D  90 03           BCC AL2         ;SPACE
2733  FE9F  20 88 FD        JSR PUTP        ;ALTER PC
2734  FEA2  20 CF FF  AL2   JSR $FFCF
2735  FEA5  20 A7 E7        JSR RDOA
2736  FEA8  90 0A           BCC AL3
2737  FEAA  A5 FB           LDA TMP0
2738  FEAC  8D 88 02        STA INVL
2739  FEAF  A5 FC           LDA TMP0+1
2740  FEB1  8D 87 02        STA INVH
2741  FEB4  20 8F FD  AL3   JSR SETR        ;SET TO ALTER R'S
2742  FEB7  D0 0A           BNE A4

2744  FEB9                  ;ALTER MEMORY - READ ADR AND DATA

2746  FEB9  20 B6 E7  ALTM  JSR RDOB        ;SKIP 2 SPACES
2747  FEBC  20 A7 E7        JSR RDOA        ;READ MEM ALTER ADR
2748  FEBF  90 D3           BCC ERRS1       ;CY=0, IF SPACE,ERR
2749  FEC1  A9 88           LDA #8          ;SET CNT = 8
2750  FEC3  85 B5     A4    STA TMPC
2751  FEC5  20 EB E7  A5    JSR RDOC
2752  FEC8  20 A7 FD        JSR BYTE
2753  FECB  D0 F8           BNE A5
2754  FECD  F0 C2     A9    BEQ BEQS1
2755  FECF  20 CF FF  GO    JSR $FFCF
2756  FED2  C9 8D           CMP #$8D        ;IF CR, EXIT
2757  FED4  F0 8C           BEQ G1
2758  FED6  C9 20           CMP #$20        ;IF NOT SPACE, ERR
2759  FED8  D0 BA           BNE ERRS1
2760  FEDA  20 A7 E7        JSR RDOA
2761  FEDD  90 03           BCC G1
2762  FEDF  20 88 FD        JSR PUTP
2763  FEE2  AE 86 02  G1    LDX SP
2764  FEE5  9A              TXS             ;ORIG OR NEW SP VALUE TO SP
2765  FEE6  78              SEI
2766  FEE7  AD 87 02        LDA INVH
2767  FEEA  85 91           STA CINV+1
2768  FEEC  AD 88 02        LDA INVL
2769  FEEF  85 90           STA CINV
2770  FEF1  AD 80 02        LDA PCH
2771  FEF4  48              PHA
2772  FEF5  AD 81 02        LDA PCL
2773  FEF8  48              PHA
2774  FEF9  AD 82 02        LDA FLGS
```

```
LINE # LOC      CODE        LINE

2775  FEFC  48              PHA
2776  FEFD  AD 03 02        LDA ACC
2777  FF00  AE 04 02        LDX XR
2778  FF03  AC 05 02        LDY YR
2779  FF06  40              RTI

2781  FF07  AE 06 02  EXIT  LDX SP
2782  FF0A  9A              TXS
2783  FF0B  4C 89 C3        JMP READY       ;EXIT TO BASIC WARM START

2785  FF0E  4C F7 E7  ERRL  JMP ERROPR

2787  FF11            ZZZ1  =BUF+7

2789  FF11            ;MACHINE LANGUAGE LOAD ROUTINE

2791  FF11  A0 01     LD    LDY #1
2792  FF13  84 D4           STY FA          ;DEFAULT DEVICE #1
2793  FF15  88              DEY
2794  FF16  84 D1           STY FNLEN
2795  FF18  84 9D           STY VERCK
2796  FF1A  A9 02           LDA #>ZZZ1      ;PLACE TO STORE NAME
2797  FF1C  85 DB           STA FNADR+1
2798  FF1E  A9 07           LDA #<ZZZ1
2799  FF20  85 DA           STA FNADR
2800  FF22  20 CF FF  L1    JSR $FFCF
2801  FF25  C9 20           CMP #' '
2802  FF27  F0 F9           BEQ L1          ;SPAN BLANKS
2803  FF29  C9 0D           CMP #CR
2804  FF2B  F0 1A           BEQ L5          ;DEFAULT TO LOAD
2805  FF2D  C9 22           CMP #'"
2806  FF2F  D0 DD     L2    BNE ERRL        ;FILE NAME MUST BE NEXT
2807  FF31  20 CF FF  L3    JSR $FFCF
2808  FF34  C9 22           CMP #'"
2809  FF36  F0 24           BEQ L8          ;END OF NAME
2810  FF38  C9 0D           CMP #CR         ;DEFAULT A LOAD
2811  FF3A  F0 0B           BEQ L5
2812  FF3C  91 DA           STA (FNADR)Y
2813  FF3E  E6 D1           INC FNLEN
2814  FF40  C8              INY
2815  FF41  C0 10           CPY #16
2816  FF43  F0 C9     L4    BEQ ERRL        ;FILE NAME TOO LONG
2817  FF45  D0 EA           BNE L3
2818  FF47  A5 B4     L5    LDA SAVX
2819  FF49  C9 06           CMP #6
2820  FF4B  D0 E2     L6    BNE L2          ;NOT A LOAD
2821  FF4D  20 22 F3        JSR LD15
2822  FF50  20 E6 F8        JSR TWAIT
2823  FF53  A5 96           LDA SATUS
2824  FF55  29 10           AND #SPERR
2825  FF57  D0 F2     L7    BNE L6          ;LOAD ERROR
2826  FF59  4C 56 FD        JMP STRT
2827  FF5C  20 CF FF  L8    JSR $FFCF
2828  FF5F  C9 0D           CMP #CR
2829  FF61  F0 E4           BEQ L5          ;DEFUALT LOAD
```

```
LINE #  LOC     CODE        LINE

2830  FF63  C9 2C           CMP #',
2831  FF65  D0 F0       L9  BNE L7          ;BAD SYNTAX
2832  FF67  20 B6 E7       JSR RDOB
2833  FF6A  29 0F          AND #$F
2834  FF6C  F0 D5       L10 BEQ L4          ;DEVICE 0
2835  FF6E  C9 03           CMP #3
2836  FF70  F0 FA       L11 BEQ L10         ;DEVICE 3
2837  FF72  85 D4           STA FA
2838  FF74  20 CF FF        JSR $FFCF
2839  FF77  C9 0D           CMP #CR
2840  FF79  F0 CC           BEQ L5          ;DEFAULT LOAD
2841  FF7B  C9 2C           CMP #',
2842  FF7D  D0 E6       L12 BNE L9          ;BAD SYNTAX
2843  FF7F  20 A7 E7        JSR RDOA
2844  FF82  20 97 E7        JSR T2T2
2845  FF85  20 CF FF        JSR $FFCF
2846  FF88  C9 2C           CMP #',
2847  FF8A  D0 F1       L13 BNE L12         ;MISSING END ADDR
2848  FF8C  20 A7 E7        JSR RDOA
2849  FF8F  A5 FB           LDA TMP0
2850  FF91  85 C9           STA EAL
2851  FF93  A5 FC           LDA TMP0+1
2852  FF95  85 CA           STA EAH
2853  FF97  20 97 E7        JSR T2T2
2854  FF9A  20 CF FF    L20 JSR $FFCF
2855  FF9D  C9 20           CMP #$20
2856  FF9F  F0 F9           BEQ L20
2857  FFA1  C9 0D           CMP #CR
2858  FFA3  D0 E5       L14 BNE L13         ;MISSING CR AT END
2859  FFA5  A5 B4           LDA SAVX
2860  FFA7  C9 07           CMP #7
2861  FFA9  D0 F8           BNE L14
2862  FFAB  20 A4 F6        JSR SV5
2863  FFAE  4C 56 FD        JMP STRT

2865  FFB1                ZZ1=ALTM-1
2866  FFB1                ZZ2=ALTR-1
2867  FFB1                ZZ3=DSPLYR-1
2868  FFB1                ZZ4=DSPLYM-1
2869  FFB1                ZZ5=GO-1
2870  FFB1                ZZ6=EXIT-1
2871  FFB1                ZZ7=LD-1
2872  FFB1                ZZ8=LD-1
```

NOTIZEN

NOTIZEN

NOTIZEN

NOTIZEN

NOTIZEN

5. Programme, die Ihnen den Computer erklären

5.1. Übungs- und Beispielprogramme zu den Punkten 1.2.1–1.2.16

Der REM-Befehl

```
100  REM PROGRAMMANFANG
110  REM EINGABE VON 2 WERTEN
120  INPUT "GEBEN SIE WERT 1 EIN" ; A
130  INPUT "GEBEN SIE WERT 2 EIN" ; B
140  REM ADDITION DER BEIDEN WERTE
150  C = A + B
160  REM AUSGABE DES ERGEBNISSES AUF DEM BILDSCHIRM
170  PRINT "DAS ERGEBNIS IST" ; C: REM PROGRAMMENDE
```

Die hinter dem REM-Befehl stehenden Zeichen haben keinen Einfluß auf das Programm. Sie dienen lediglich der besseren Übersicht beim Erstellen der Programme oder bei der Fehlersuche in Programmen; siehe Zeile 100, 110, 140, 160. Wird ein REM-Befehl mit anderen Befehlen in eine Zeile geschrieben, so muß er immer als letzter stehen, (siehe Zeile 170) da die Befehle, die hinter dem REM-Befehl stehen, nicht ausgeführt werden.

Die END-Anweisung

```
10  REM PROGRAMMANFANG
11  INPUT "ZAHL 1" ; A
12  INPUT "ZAHL 2" ; B
13  IF A > B THEN PRINT A "IST GRÖSSER ALS" ; B
14  IF A < B THEN PRINT A "IST KLEINER ALS" ; B
15  IF A = B THEN PRINT A "IST GLEICH" ; B
16  INPUT "WOLLEN SIE NOCH EINEN VERGLEICH DURCHFÜHREN"
    ; A$ : IFA$= "JA" THEN 10
17  END
```

Die END-Anweisung in Zeile 17 beendet das Programm. Sie kann aber auch fortgelassen werden. Eine END-Anweisung darf auch öfter im Programm verwendet werden. Sie beendet den Programmablauf ohne Meldung "BREAK IN . . ." wie bei der STOP-Anweisung. Stattdessen erscheint die Meldung READY im Bildschirm.

Der STOP-Befehl

```
110  PRINT "VON WELCHER ZAHL WOLLEN SIE DIE WURZEL ZIEHEN?"
120  INPUT A
130  B = SQR (A)
140  PRINT "DIE WURZEL DER ZAHL "A" IST "B"
150  STOP
160  GOTO 110
```

Der STOP-Befehl in Zeile 150 bewirkt das Anhalten des Programmes und die Meldung "BREAK IN 110". Es können auch mehrere STOP-Befehle in einem Programm verwendet werden. Nach einem Programmhalt durch einen STOP-Befehl kann das Programm mittels "CONT" fortgesetzt werden.

Der LET-Befehl

```
11  INPUT "ZAHL 1" ; A
12  INPUT "ZAHL 2" ; B
13  INPUT "ZAHL 3" ; C
14  LET D = A + B + C
15  PRINT "A" PLUS "B" PLUS "C" IST GLEICH "D"
```

Der LET-Befehl in Zeile 14 war bei früheren BASIC-Versionen als Zuordnung nötig. Er kann bei Ihrem Computer weggelassen werden. Zeile 14 sähe dann so aus:

14 D = A + B + C. Ältere Programme, in denen noch ein LET-Befehl steht, brauchen nicht modifiziert werden, da er keine Fehlermeldung hervorruft.

PRINT

```
100 REM PROGRAMMANFANG
110 FOR I = 1 TO 10 : INPUT"TEXT";A$(I) : NEXT
120 PRINT : PRINT"WELCHE AUSGABE (0 BIS 4; 0=PROGRAMMENDE)"
130 INPUT A
135 PRINT
140 ON A+1 GOTO 300, 220, 240, 260, 280
200 PRINT "EINGABE UNZULAESSIG"
210 GOTO 130
220 FOR I = 1 TO 10 : PRINT A$(I) : NEXT
230 GOTO 120
240 FOR I = 1 TO 10 : PRINT A$(I),: NEXT
250 GOTO 120
260 FOR I = 1 TO 10 : PRINT A$(I);: NEXT
270 GOTO 120
280 FOR I =1 TO 10 : PRINT A$(I)" ";:NEXT
290 GOTO 120
300 PRINT "ENDE" : END
```

Die PRINT-Befehle in den Zeilen 220, 240, 260, 280 bewirken die Darstellung von Zeichen auf dem Bildschirm. In Zeile 220 werden die eingegebenen Werte untereinander geschrieben. Setzt man ein Komma oder ein Semikolon hinter den PRINT-Befehl (wie in den Zeilen 240, 260), so werden die Werte nebeneinander geschrieben. Nur die Abstände zwischen den einzelnen Werten ändern sich durch das Komma oder Semikolon. Man kann nun noch den Abstand vergrößern, indem man Leerstellen mit ausdrucken läßt (Zeile 280).

INPUT

```
100 INPUT A
110 INPUT "ZAHL 2";B
120 INPUT A$
130 FOR I = 1 TO 10 : INPUT "GEBEN SIE NAECHSTEN WERT EIN";A$(I) : NEXT
140 PRINT A
150 PRINT B
160 PRINTA$
170 FORI=1TO10:PRINTA$(I):NEXT
```

Der INPUT-Befehl ermöglicht die Eingabe von Werten über die Tastatur. Durch den INPUT-Befehl in Zeile 100 können Sie eine Zahl eingeben. Der Rechner meldet sich nur mit einem Fragezeichen. Für die Zeile 110 gilt das Gleiche wie für Zeile 100, außer daß der Rechner den Text innerhalb der Anführungszeichen am Bildschirm darstellt. Es können neben Zahlen auch Buchstaben und graphische Zeichen eingegeben werden, wenn diese in einer Stringvariablen gespeichert werden (Zeile 120, 130). Für die Stringvariablen gilt die gleiche Vorschrift wie für einfache Variablen.

GET

```
10 PRINT"B LOESCHT DEN BILDSCHIRM,"
20 PRINT"V LOESCHT ALLE VARIABLEN,"
30 PRINT"N LOESCHT DAS PROGRAMM."
40 PRINT:PRINT"EINGABE?"
50 GET C$ : IF C$=" " THEN 50
60 IF C$ = "B" THEN PRINT CHR$ (147) : END
70 IF C$ = "V" THEN CLR : END
80 IF C$ = "N" THEN NEW
90 GOTO 50
```

Durch einen GET-Befehl wird ein einzelnes Zeichen vom Tastenfeld eingelesen, jedoch ohne das Programm anzuhalten, wie es beim INPUT-Befehl ist. Der GET-Befehl in Zeile 50 bewirkt das Einlesen des Zeichens und speichert dieses in C$. Im zweiten Teil der Zeile 50 wird abgefragt, ob überhaupt ein Zeichen eingelesen wurde. Ist dies der Fall, so wird das Programm mit der Zeile 60 fortgesetzt, ansonsten wird Zeile 50 wiederholt. Dies ist also eine Schleife, die das Programm solange anhält, bis ein Zeichen eingelesen wurde.

DATA, READ und RESTORE

```
100 DATA ICH, DU, ER, SIE, ES, WIR, IHR, SIE, 1,2,3,4,5,6,7,8,9,10
110 PRINT "DRÜCKEN SIE JETZT 1 ODER 2"
115 PRINT "(1 = READ, 2 = RESTORE UND READ)"
120 GETA$ : IFA$ = "1" OR A$ = "2" THEN 140
130 GOTO 120
140 IFA$ = "1" THEN 170
150 IFA$ = "2" THEN 160
160 RESTORE
170 READ A$
180 PRINT A$
190 IF A$ = "10" THEN 210
200 GO TO 120
210 RESTORE
220 GOTO 120
```

Die Daten, die hinter der DATA-Anweisung im Programm stehen (Zeile 100) können mittels eines READ-Befehls (Zeile 120) nacheinander gelesen werden. Versucht man, mehr Daten, als unter DATA stehen, zu lesen, so erfolgt eine Fehlermeldung. Nach einem RESTORE-Befehl (Zeile 160) wird mit dem nächsten READ-Befehl wieder der erste Wert der DATA-Anweisung gelesen. So ist es möglich, die Werte der Anweisung öfter zu benutzen.

GOTO, ON .. GOTO, IF ... THEN

```
10 REM TEXTAUSWAHL
15 PRINT
20 INPUT"WELCHES DER 5 TEXTMODULN";X
30 IF X<1 OR X>5 THEN 160
40 ON X GOTO 60,80,100,120,140
50 GOTO 20
60 PRINT"FUER JEDES 'MODUL' KOENNTE BELIEBIGER TEXT STEHEN"
70 GOTO 15
80 PRINT "MODUL2"
90 GOTO 15
100 PRINT "MODUL3"
110 GOTO 15
120 PRINT "MODUL4"
130 GOTO 15
140 PRINT "MODUL5"
150 GOTO 15
160 PRINT"BITTE GEBEN SIE EINE ZAHL ZWISCHEN 1 UND 5 EIN!"
170 GOTO 15
```

Die Variable (oder Funktion) hinter ON .. gibt die Position in der nach GOTO folgenden Adressenliste an. Die gefundene Adresse wird dann für den Sprungbefehl GOTO in der gleichen Zeile (40) benutzt.

Ist bei ON ... GOTO die Variable 0, verzweigt der Programmablauf direkt zur nächsthöheren Programmzeile.

GOSUB, ON ... GOSUB

```
10 INPUT"WUENSCHEN SIE EINE ERKLAERUNG";X$
20 IF X$="JA"THEN GOSUB160
30 INPUT"IHRE ZAHL";A
40 INPUT"IHREN CODE";B:IFB<0ORB>3THEN10
50 ON B GOSUB70,100,130
60 GOTO30
70 REM UNTERPROGRAMM 1
80 PRINT"DIE WURZEL AUS IHRER ZAHL LAUTET";SQR(A)
90 RETURN
100 REM UNTERPROGRAMM 2
110 PRINT"IHRE ZAHL ZUM QUADRAT LAUTET";A↑2
```

```
120 RETURN
130 REM UNTERPROGRAMM 3
140 PRINT"DIE ZUFALLSZAHL AUS IHRER ZAHL LAUTET";RND(A)
150 RETURN
160 REM ERKLAERUNG
170 PRINT"GEBEN SIE EINE BELIEBIGE ZAHL EIN"
180 PRINT"MIT EINER CODE-ZAHL BESTIMMEN SIE EINE VORPROGRAMMIERTE
    RECHENOP."
190 PRINT TAB(10)"1=QUADRATWURZEL"
200 PRINT TAB(10)"2=POTENZIERUNG MIT 2"
210 PRINT TAB(10)"3=ZUFALLSZAHL"
220 RETURN
```

Mit GOSUB 160 in Zeile 20 wird ein Unterprogramm angesprungen und automatisch ausgeführt. Hier wird nur eine Erklärung zum Programm ausgedruckt.

Mit ON .. GOSUB wird das gleiche Prinzip wie bei ON .. GOTO erfüllt. In Abhängigkeit von einer Variablen wird eine bestimmte Adresse für einen Unterprogrammsprung verwendet.

Durch RETURN in den Unterprogrammen verzweigt das Programm wieder zurück zu dem Befehl, der **nach** dem GOSUB-Befehl steht. Hier ist das immer der Befehl in der nächsten Zeile. Der nachfolgende Befehl kann jedoch in der gleichen Zeile stehen. Ist bei ON .. GOSUB die Variable=0, so verzweigt das Programm direkt zur nächsthöheren Programmzeile.

5.2. Beispielprogramm zu 1.2.17

DIE BEFEHLSGRUPPE FOR, NEXT, STEP.

```
 5 PRINT CHR$ (147)
10 FORI=1TO3
20 PRINT"VERSUCH";J
30 NEXTI
40 PRINT
50 A=0:B=20:C=2
60 FORI=A TO B STEP C
70 PRINT"SCHRITT";I
80 NEXT
85 PRINT"      WARTESCHLEIFE":FORI=1TO2000:NEXT
90 DIMA(3,8):PRINT"   RANDOM GENERATOR      "
100 FORI=1TO3
110 FORJ=1TO8
120 A(I,J)=RND(π)
130 NEXTJ
140 NEXTI
150 END
```

In Zeile 10 und 30 wird ein einfacher Zähler gebildet, der von 1 bis 3 läuft, wobei die Schrittweite 1 ist, da diese (mit STEP) nicht angegeben ist.

In Zeile 60 werden die Grenzwerte und die Schrittweite der Schleife in Form von einfachen Variablen (Zeile 50) vorgegeben. Hier wird eine Schrittweite von 2 vorgegeben.

In den Zeilen 100, 110, 130, 140 wird eine „verschachtelte" FOR-NEXT-Schleife angegeben. Hier ist darauf zu achten, daß die Verschachtelung symmetrisch ist, d.h. die Schleife muß mit der Variablen beendet werden, mit der sie begonnen hat (Zeile 140 und 100).

Eine Besonderheit des Commodore-BASIC ist in Zeile 80 deutlich zu erkennen. In der NEXT-Anweisung muß nicht zwingend die Laufvariable angegeben werden.

5.3. Beispielprogramm zu 1.2.18

```
10 DIMA(100),B(10,10),C(3,3,3),D%(100)
20 FORI=0TO100
30 A(I)=I:D%(I)=I↑2
40 NEXTI
45 PRINT CHR$ (147)
```

```
50 FORJ=0TO10
60 FORK=0TO10
70 B(J,K)=RND(π)
80 NEXTK
90 NEXTJ
100 FORI=0TO3
110 FORJ=0TO3
120 FORK=0TO3
130 C(I,J,K)=I*J*K
135 PRINT"ELEMENTC("I;J;K")="C(I,J,K)
140 NEXTK
150 NEXTJ
160 NEXTI
```

Der DIM-Befehl in Zeile 10 reserviert innerhalb des Speichers den notwendigen Platz für ein-, zwei- oder mehrdimensionale Felder (Arrays). Wichtig ist, daß dieser Befehl vor Benutzung der Arrays mindestens einmal durch Starten des Programms mit RUN durchlaufen wird. RUN löscht ja bekanntlich alle Variablen und somit auch Arrays, die vorher vorhanden waren. Das Array D% (100) ist ein INTEGER-Array; d.h. die Zahlenwerte dürfen nicht größer als 32768 und nicht kleiner als -32767 sein.

Es ist darauf zu achten, daß Ihr Rechner mit der Basis-Null-Indizierung arbeitet; d.h. das erste Element besitzt die Indizierung 0. Diese Besonderheit ist in den FOR Statements in Zeile 20, 50, 60, 100, 110 und 120 Rechnung getragen. Die Laufvariablen beginnen dort mit 0. Um die Darstellung auf dem Bildschirm zu verlangsamen, kann die Taste RVS betätigt werden.

5.4. Beispielprogramme zu 1.2.19–1.2.24

DEFFN ... und FN ...

```
20 DEFFNY(X)=A+A1*X+A2*X↑2+A3*X↑3
30 INPUT"A,A1,A2,A3,X";A,A1,A2,A3,X
40 PRINT:PRINT"FNY("X")="FNY(X)
50 PRINT:GOTO20
```

In der Zeile 20 wird eine mathematische Funktion definiert. Daher auch DEFINE FUNCTION als Bezeichnung für dieses Statement.
Die einmal mit RUN durchlaufene Formel kann innerhalb eines Programmes beliebig oft mit FNY aufgerufen werden.

Als Name für die Funktion (hier Y) können maximal 2 Zeichen (einfache Variablen, ausgenommen INTEGER-Variablen) Verwendung finden.

OPEN, PRINT # , CLOSE und INPUT #

```
100 REM DATENEINGABE
110 FORI=1TO10
112 INPUT" ARTIKEL";A$(I)
114 INPUT"     PREIS";B$(I)
116 NEXT
120 PRINT"BITTE DATENKASSETTE RUECKSPULEN, DANN 'S' DRUECKEN!"
130 GETZ$:IFZ$<>"S"THEN130
140 OPEN1,1,1
150 FORI=1TO10:PRINT#1,A$(I):NEXT
160 FORI=1TO10:PRINT#1,B$(I):NEXT
170 CLOSE1
180 CLR
190 PRINT"BITTE DATENKASSETTE RUECKSPULEN, DANN 'L' DRUECKEN!"
200 GETZ$:IFZ$<>"L"THEN200
210 OPEN3
220 FORI=1TO10:INPUT#3,A$(I):NEXT
230 FORI=1TO10:INPUT#3,B$(I):NEXT
240 CLOSE3
250 FORI=1TO10:PRINTA$(I),,B$(I):NEXT
260 END
```

Zur Ausgabe von Daten auf das Kassettengerät oder ein anderes Peripheriegerät muß zunächst ein logisches File eröffnet werden. Für das Kassettengerät steht dieser Befehl in Zeile 140. Danach könnten Daten mit dem Befehl (PRINT # 1, Variable) auf Band geschrieben werden (Zeile 150, 160). Sind alle Daten auf Band gespeichert, so muß das eröffnete File geschlossen werden. Dieses geschieht mit dem Befehl in Zeile 170 (CLOSE 1).

Wollen Sie nun Daten von der Kassette lesen, so müssen Sie wieder ein logisches File eröffnen, beim Lesen der Kassette lautet der Befehl OPEN 3 (Zeile 210). Das eigentliche Einlesen geschieht durch den Befehl "INPUT # 3, Variable" (Zeile 220, 230). Genau wie beim Schreiben, so muß auch nach dem Lesen der Daten das File wieder geschlossen werden. Dies geschieht durch CLOSE 3 (Zeile 240).

5.5. Beispielprogramme zu 1.2.25–1.2.36

DIE LEN-FUNKTION

```
10 INPUT"GEBEN SIE EINEN BELIEBIGEN TEXT EIN";A$
20 PRINT"DIESER TEXT IST"LEN(A$)"ZEICHEN LANG"
30 FORI=1TOLEN(A$)
40 PRINTMID$(A$,I,1)
45 FORU=1TO555:NEXT:REM * WARTE *
50 NEXT
60 PRINT"TEXTLAENGE HOCH ZWEI IST";LEN(A$)↑2
```

Die Funktion LEN ( ) ermittelt die Länge (Anzahl der Zeichen) einer Stringvariablen (Zeile 20). Sie kann darüber hinaus die Funktion einer normalen Variablen übernehmen (FOR-Statement in Zeile 30). Mit ihr kann aber auch gerechnet werden (Zeile 60).

STR $ und VAL

```
10 INPUT"IRGENDEINE ZAHL";A
20 A$=STR$(A)
30 PRINT"A="A,"A$="A$
40 B=VAL(A$)
50 PRINT"B="B,"A$="A$
60 PRINT:GOTO10
```

Mit STR$ ( ) läßt sich ein numerischer Wert in eine Stringvariable transferieren (Zeile 20).

Die Umkehrfunktion dazu, die VAL ( ) Funktion, ermöglicht es, aus einer Stringvariablen den numerischen Teil (incl. Vorzeichen) zu lesen. Der Inhalt der Stringvariablen bleibt bei dieser Operation unverändert.

Befindet sich im String hinter dem numerischen Wert ein E, so wird dies als Exponent verstanden, und die nachfolgenden Zahlen werden als Exponent gelesen.

ASC und CHR$

```
10 PRINT"  ASCII-TABELLE FUER A-P":PRINT:PRINT
20 A$="ABCDEFGHIJKLMNOP"
30 FORI=1TO16
40 M$=MID$(A$,I,1)
50 PRINTM$" HAT CODE "ASC(M$)
60 NEXT
70 PRINT"IRGENDEINE TASTE DRUECKEN–"
80 GETR$:IFR$="" "THEN80
90 INPUT"WELCHER CODE";C
100 PRINT"ASC VON"C"= "CHR$(C)
110 PRINT:GOTO90
```

Die ASC ( )-Funktion wandelt das in der Klammer angegebene Zeichen in den zugehörigen ASCII* um (Zeile 50), während die CHR$ ( )-Funktion aus einem ASCII-Zeichen wiederum das alphanumerische Zeichen bildet. In Zeile 100 wird aus dem Code das zugehörige Zeichen gebildet.

---

*ASCII = American Standard Code for Information Interchange
(standardisierter Code für Informationsaustausch)

RIGHT$, MID$, LEFT$

```
100 INPUT"WIE SPAET IST ES JETZT – HHMMSS";Z$
110 PRINT"   ":REM *BILDSCHIRM LOESCHEN*
115 IFVAL(Z$)<0ORVAL(Z$)>240000THEN100
116 TI$=Z$
120 A$=RIGHT$(TI$,2):REM *DIE BEIDEN RECHTEN ZEICHEN *
130 B$=MID$(TI$, 3,2):REM *DIE BEIDEN MITTLEREN ZEICHEN AB DEM DRITTEN*
140 C$=LEFT$(TI$,2):REM *DIE BEIDEN LINKEN ZEICHEN *
150 PRINT" QQQQQQQQ ES IST JETZT":PRINTC$" UHR, "B$" MINUTEN, "A$" SEKUNDEN
160 GOTO 120
```

Durch den Befehl A$ = RIGHT$ (TI$,2) in Zeile 120 werden von dem String TI$, das die Uhrzeit HHMMSS enthält, von der rechten Seite an 2 Zeichen in A$ gespeichert. Durch den Befehl in Zeile 130 werden aus der Mitte des Strings TI$, vom 3. Zeichen beginnend, 2 Zeichen in B$ gespeichert. Der Befehl in Zeile 140 bewirkt das Einlesen von 2 Zeichen von der linken Seite des Strings. Sollen mehr Zeichen als das String hat eingelesen werden, so wird der gesamte String dargestellt.

Durch die Printanweisung in Zeile 150 wird erst „Cursor home" programmiert, dann 10 mal „Cursor nach unten". (Programmierte Cursorbewegungen erscheinen als Sonderzeichen auf dem Bildschirm.)

POKE, PEEK, SYS

Nachfolgendes Assemblerprogramm addiert die Inhalte zweier Speicher und legt sie in einem dritten Speicherplatz ab.

```
CLC                 Clear Carry-Flag
LDA     $0347       Lade-Akku mit Inhalt von 839
ADC     $0348       Addiere zum Akku den Inhalt von 840
STA     $0349       Speichere das Ergebnis in 841
RTS                 Return (Rückkehr aus der Subroutine)
```

Umgesetzt in die Maschinensprache erhält man

```
18
AD 47 03
6D 47 03        Hexadezimalcode
8D 47 03
60
```

Zunächst werden die Befehle nach Dezimal umgewandelt:

```
 24
173 71 3
109 72 3
141 73 3
 96
```

Mit einem BASIC-Programm wird dieses Programm in den Speicher geschrieben.

```
10 DATA 24, 173, 71, 3, 109, 72, 3, 141, 73, 3, 96
20 FOR M = 826 TO 836
30 READ A
40 POKE M, A
50 NEXT
```

Der Befehl SYS (826) kann dieses Maschinenprogramm aufrufen.

Zuvor kann mit NEW der Hauptspeicher gelöscht werden, da ab Adresse 826 der Pufferspeicher der Kassette # 2 beginnt, der mit NEW nicht gelöscht wird.

```
10 INPUT"1.ZAHL";Z1:POKE839,Z1
20 INPUT"2.ZAHL";Z2:POKE840,Z2
30 SYS(826)
40 PRINTZ1"+"="PEEK(841)
```

Mit POKE (Zeile 40) wurde der Code unseres Maschinenprogramms direkt in die Speicherstellen 826 bis 836 geschrieben.

Bei der Abarbeitung des Unterprogramms (Zeile 30) wurde das Ergebnis in den Speicher 841 gebracht. PEEK (841) in Zeile 40 liest diesen Wert und druckt ihn aus.

6. Was tue ich, wenn der Computer mich nicht versteht?

Im allgemeinen wird es nötig sein, Ihr Gerät zum Service zu bringen. Ihr Computer enthält hochkomplexe Schaltungen, und wir empfehlen keinerlei Eingriffe, außer der unten ausdrücklich beschriebenen.

Beachten Sie besonders, daß der Bildschirmteil mit Hochspannung (mehrere Tausend Volt) arbeitet.

Trotzdem einige Fehlerhinweise für Hardware-Interessierte:

6.1. Der Bildschirm spinnt.

a) Der Bildschirmhintergrund ist zu hell.

Diagnose: Versuchen Sie, am Potentiometer (Regler) auf der Bildschirmrückseite das gesamte Bild dunkler zu regeln. Sollte das keinen Erfolg haben, so ist der HF-Trafo im Bildschirmteil defekt.

b) Der Bildschirm zeigt nach dem Einschalten ein festes Bild mit diversen Zeichen und reagiert nicht auf die Tastatur.

Diagnose: Der Einschaltzyklus des Rechners ist nicht einwandfrei (power-on-reset).

c) Der Bildschirm zeigt nach dem Einschalten weniger als 31743 bzw. 15359 bzw. 7167 Bytes an.

Diagnose: Eines der RAM's ist defekt, sofern dieser Fehler wiederholt auftritt. Sie können trotzdem noch mit dem verbliebenen Speicherrest arbeiten, sollten aber das defekte RAM irgendwann einmal austauschen lassen.

d) Im Bildschirm befindet sich eine Stelle, die nach "Clear Home" noch immer ein Zeichen beinhaltet.

Diagnose: Eines der RAM für den Bildschirm ist defekt. Der Fehler ist nicht tragisch, sollte jedoch behoben werden.

e) Gewisse Zeichen werden auf dem Bildschirm falsch angezeigt (z.B.: 7 anstatt A)

Diagnose: Der Zeichengenerator ist defekt.

f) Der Bildschirm ist vertikal gedrückt oder gedehnt.

Diagnose: Lassen Sie die Einstellung der Vertikalablenkung in einer Service-Werkstatt durchführen.

g) Das Bild steht schräg.

Diagnose: Lassen Sie in Ihrer Werkstatt die Ablenkspulen justieren.

h) Die Zeichen sind verschwommen.

Diagnose: Lassen Sie die Fokussierung einstellen.

6.2. Die Tastatur reagiert nicht mehr.

a) Eine einzelne Taste funktioniert nicht.

Diagnose: Die Tastatur muß gereinigt werden. Durch normale Benutzung verunreinigen die Tastaturkontakte mit der Zeit. Die meisten Probleme treten bei den Tasten RETURN / SHIFT / HOME auf, da diese Tasten auch am meisten benutzt werden.

Hinweis: Es ist möglich, einzelne Tasten vorsichtig mit einem Schraubenzieher abzuheben.

b) Eine Gruppe von Tasten reagiert nicht.

Diagnose: Eine Leitung an der Tastatur ist locker.

6.3. Die Kassette will nicht so wie ich.

a) Beim Einladen eines Programmes von Kassette wird nichts gefunden, obwohl ein Programm vorhanden ist.

Diagnose: Haben Sie die Kassette vor Programmbeginn zurückgespult?
Wenn ja, so kann der Tonkopf am Rekorder falsch stehen. Lassen Sie das überprüfen.

b) Sie speichern ein Programm auf Kassette. Anschließend stellen Sie fest, daß die Kassette fehlerhaft beschrieben wurde.

Diagnose: Reinigen Sie den Tonkopf des Rekorders mit einem handelsüblichen Tonkopfreiniger. Sollte der Fehler danach wieder auftreten, so ist die Qualität der benutzten Kassette schlecht.
Übrigens: Sie sollten den Tonkopf ca. alle 10 Spielstunden reinigen.

c) Der Rechner reagiert nicht auf Tastenbenutzung am Kassettenrekorder, wenn Sie darauf lesen oder schreiben wollen.

Diagnose: Es kann sich hierbei um einen Kontaktfehler im Rekorder handeln. Suchen Sie die Werkstatt auf.

d) Der Kassettenrekorder leiert.

Diagnose: Lassen Sie die Zugrollen im Rekorder reinigen.

6.4. Es tut sich gar nichts.

Diagnose: Überprüfen Sie die Sicherung auf der Rechnerrückseite **(ziehen Sie unbedingt vorher den Netzstecker).** Sollte diese in Ordnung sein, so kann der Bildschirmstecker auf der Hauptplatine lose sein, oder der Einschaltzyklus des Rechners ist nicht einwandfrei. Ihre Werkstatt wird in diesem Falle helfen können.

6.5. Der Computer meldet einen Fehler.

Grundsätzlich ist Ihr Computer so ausgelegt, daß er keinerlei Bauteildefekte lokalisieren kann. Fehlbedienungen durch Ihre Programmierung kann der Rechner allerdings in weitem Rahmen feststellen, und meldet diese auch. Im Anhang finden Sie eine Anzahl von Meldungen, die auf dem Bildschirm angezeigt werden, wenn ein Programmierfehler entdeckt wurde.

An dieser Stelle ist jedoch zu erwähnen, daß ein Fehler erst in dem Augenblick gemeldet wird, wenn das Programm im Ablauf die Fehlerstelle erreicht hat. Sollte Ihr Programm umfangreicher sein, so müssen Sie alle Eventualitäten beim Testen des Programmes berücksichtigen, um auch tatsächlich alle Fehler im Programm zu entdecken. So werden Sie bei dem Programm:

```
10 INPUT "GIB WERT A UND B" ; A,B
20 PRINT "ERGEBNIS"; A/B
```

nur dann Fehler entdecken, wenn Sie folgendes eingeben:

a) für B den Wert 0. Der Rechner meldet dann DIVISION BY ZERO ERROR IN 20.

b) für A = 1E3Ø und B = E–25. Der Rechner stellt dann ? OVERFLOW ERROR IN 2Ø fest, da das Ergebnis 1E55 größer als die größte abzuspeichernde Zahl 1.5E38 ist.

c) Geben Sie durch Kommata getrennt 3 anstatt der verlangten 2 Variablen ein.

Der Rechner bemerkt dazu

? EXTRA IGNORED.

Da dieser Fehler aber nicht tragisch ist, setzt der Rechner anschließend seine Arbeit fort und vergißt die 3. Zahl.

6.6. Was sonst noch so passieren kann.

Wir können Ihnen im Rahmen dieses Handbuches verständlicherweise nicht alle möglicherweise auftretenden Fehler diagnostizieren.

Für diese Fälle haben wir ein Servicenetz in Deutschland aufgebaut, das sich sehen lassen kann. Bei jedem COMMODORE-Händler befindet sich gleichzeitig eine Reparatur-Werkstatt, die durch uns speziell auch für Ihren Rechnertyp geschulte Techniker hat. Dort wird man Ihre Probleme mit dem Rechner schnell und sicher beseitigen können. Während der Garantiezeit sind die Reparatur- und evtl. Ersatzteile natürlich kostenlos. (Es sei denn, der Techniker kommt zu Ihnen ins Haus.) Aber auch danach wird für eine schnelle Instandsetzung gesorgt sein. Ein Verzeichnis unserer Händler und Service-Werkstätten liegt diesem Handbuch bei.

7. Wie schnell ist eigentlich mein Rechner?

7.1. Die Geschwindigkeit des BASIC-Interpreters.

Ihnen eine Tabelle mit Zeiten für jeden Befehl zu geben, halten wir für unsinnig. Durch den jeweiligen Aufbau des Programmes können sich nämlich erhebliche Geschwindigkeitsdifferenzen ergeben. Wir wollen Ihnen lieber ein Programm zeigen, mit dem Sie die Geschwindigkeit eines einzelnen Befehls genauso messen können wie die einer kompletten Programmroutine oder auch eines ganzen Programmes.

Von Zeile 1 bis Zeile 63995 kann Ihr zu testendes Programm stehen. In diesem dürfen TU, TV und TT nicht als Variablen vorkommen. Starten Sie das komplette Programm mit RUN und warten Sie mindestens 20 Sekunden.

```
Ø TT=TI:FORTU=ØTOTV
1 REM      *** TESTANFANG    ***
2 A=A
63996 REM*** TESTENDE        ***
63997 NEXT:TT=TI-TT:IFTT(1ØØØTHENTV=TU*1ØØØ/TT:GOTOØ
63998 PRINTINT(1ØØØØ*(TT/(TU*6Ø)))/1Ø"MILLISEKUNDEN PRO DURCHLAUF"
63999 PRINT"GETESTET IN"TU"DURCHLAEUFEN"
```

7.2. Die Geschwindigkeit der Maschinensprache.

Die Taktfrequenz Ihres Rechners beträgt genau 1 MHz, die eigentlichen Maschinensprachenbefehle brauchen zwischen 2 und 7 Taktzyklen. Das bedeutet, daß 1 Maschinensprachenbefehl 2–7 μsec braucht. Wenn man davon ausgeht, daß eine Addition zweier Integerzahlen mit ca. 20 Befehlen zu bewältigen ist, so dauert solch eine Addition im ungünstigsten Falle 140 μsec.

7.3. Wie kommt es zu solchen Geschwindigkeitsdifferenzen?

Wenn wir eine Integer-Addition in Basic programmieren und diese mit unserem Testprogramm zeitlich erfassen, so erhalten wir eine Zeit von ca. 3 Millisekunden.

```
1Ø A% = A%+A%
```

Das vergleichbare Maschinenprogramm ist in diesem Falle rund 20 mal schneller.

Generell kann man sagen, daß Maschinenprogramme etwa zwanzig bis tausend mal schneller sind als in Basic geschriebene Programme. Die Geschwindigkeitsdifferenz erklärt sich dadurch, daß die Basicbefehle vom Rechner zusätzlich zur Ausführung noch interpretiert werden müssen. Und das dauert halt seine Zeit. Für diejenigen unter Ihnen, die jetzt sagen: „Warum soll ich dann nicht generell in der Maschinensprache programmieren?" sei erwähnt, daß Programme in Maschinensprache sich wesentlich schwerer programmieren lassen als in Basic und Ihnen keinerlei Möglichkeiten geben, Befehle einzufügen oder zu listen.

8. Wie schreibe ich ein Programm?

8.1 Am Anfang steht die Idee

Bevor Sie die Idee in die Tat umsetzen, sollten Sie sich natürlich einige Gedanken machen, ob der Computer überhaupt in der Lage ist, Ihr Problem zu lösen.

Zunächst gilt es, den erforderlichen Speicherbedarf zumindest überschlägig zu ermitteln, denn sehr oft ist dieses die kritischste Größe überhaupt. Die zu speichernden Daten werden meistens in Tabellen (Arrays) innerhalb von DATA-Statements Ihres Programms abgelegt.

Hier kann sehr einfach auf exakte Art der erforderliche Platzbedarf ermittelt werden.

Beim Anlegen von Arrays gelten nachfolgende Speicherbedarf-Formeln:

Array mit voller Genauigkeit:

$$x = 5 + 2 * Z + 5 * \pi\ (Z_y + 1)$$
$$y = 1 \ldots Z$$

Array mit Integer Genauigkeit:

$$x = 5 + 2 * Z + 2 * \pi\ (Z_y + 1)$$
$$y = 1 \ldots Z$$

Arrays mit String-Variablen:

$$x = 5 + 2 * Z + 3 * \pi\ (Z_y + 1)$$
$$y = 1 \ldots Z$$

dabei bedeuten:

x	= gesamter Platzbedarf in Bytes
Z	= Anzahl der Dimensionen
Z_1 bis Z_u	= Größe der einzelnen Dimensionen, wie in DIM-Befehl angegeben
π	= Alle Summen werden miteinander multipliziert.

Beispiel:

DIM A (5,6): REM hier ist Z = 2
X = 5 + 2 *2 + 5 * (5 + 1) * (6 + 1) = 219 Bytes

DIM A % (3,4)
x = 5 + 2 * 2 + 2 * (3 + 1) * (4 + 1) = 49 Bytes

DIM A (10, 10, 10)
x = 5 + 2 * 3 + 5 * (10 + 1) * (10 + 1) * (10 + 1) = 6666 Bytes

DIM A$ (5,6)
x = 5 + 2 * 2 + 3 * (5 + 1) * (6 + 1) = 135 Bytes

Sollen eventuelle Daten oder Texte innerhalb eines Programms in DATA Statements abgelegt werden, muß man wohl oder übel Zeichen für Zeichen zählen und so den erforderlichen Platzbedarf ermitteln. Ein Zeichen entspricht dann einem Byte.

Ferner könnte man die Ein- und Ausgabe-Intensität des zukünftigen Programms berücksichtigen. Ihr Rechner bietet sehr viel Komfort für den Dialog mit dem Benutzer, den man selbstverständlich auch ausnutzen möchte. Dieser Komfort kostet andererseits auch viel Platz. Wenn man davon ausgeht, daß im allgemeinen ca. 30 % eines vollbeschriebenen Bildschirms für eine gut gestaltete Ein-/Ausgabemaske benötigt werden, so erzielt man auch hier sehr schnell, je nach Anzahl der Masken, den erforderlichen Platzbedarf.

Sind die Speicherbedarf-Kriterien geklärt, kann man sich mit dem Ablaufschema befassen.

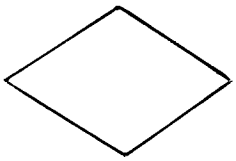
8.2 Ein Schema wird erdacht (Flußdiagramm)

Um ein Flußdiagramm erstellen zu können, ist es natürlich notwendig, die dazu verwendeten Symbole kennenzulernen.

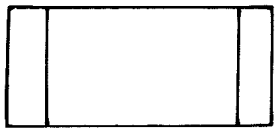
Auch hier gibt es eine Norm. Die nachfolgende Liste zeigt die nach DIN 66001 zulässigen Symbole.

Symbole nach DIN 66001 für Programmablaufpläne

Eine allgemeine Operation, die etwas ausführt (Handlung)



Eine Verzweigung, wahlweise in eine von zwei Richtungen (identisch mit unserer Fragestellung)



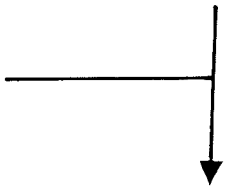
Ein Unterprogramm
(Befehlsempfänger zur Durchführung einer bestimmten Aufgabe)

Programmodifikation

Operation von Hand
(Operationen, die der Rechner nicht ausführen kann, z. B. drücken einer Taste auf dem Kassettenrecorder)

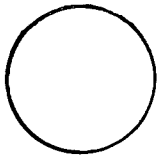
Eingabe, Ausgabe
(wenn z. B. am Anfang eines Programms Werte für eine Berechnung vorgegeben werden müssen)

Ablauflinie

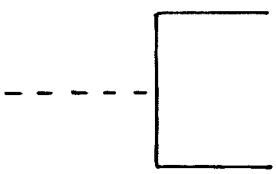


Zusammenführung von Ablauflinien

Grenzstelle
(z. B. Beginn oder Ende eines Programms)

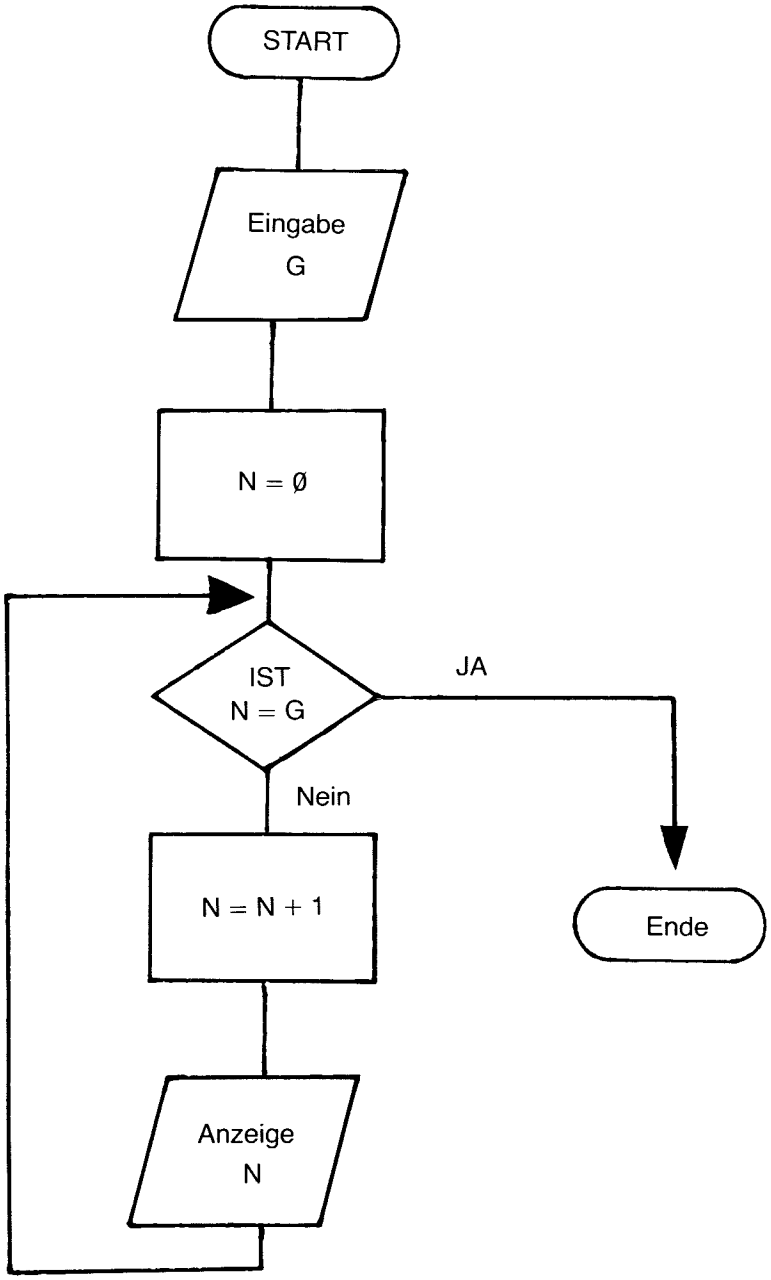


Übergangsstelle oder Schnittstelle



Bemerkungen zum Ablaufplan können neben diesem Symbol vorgenommen werden

Im nachfolgenden Beispiel eines Ablaufplans wird ein einfacher Zähler, der bis zu einem bestimmten Grenzwert G laufen soll, dargestellt.



Dieses Flußdiagramm in die Sprache BASIC umzusetzen, ist nicht weiter schwierig:

```
20  INPUT "EINGABE G"; G
30  N = Ø
40  IF N = G THEN END
50  N = N + 1
60  ? N
70  GOTO 40
```

Die Schwierigkeit bei der Umsetzung eines Flußdiagramms in die jeweilige Programmiersprache besteht darin, daß Programm möglichst effektiv aufzubauen. Und da hilft Ihnen – leider – ein Flußdiagramm kaum.

Das gleiche Programm sieht optimiert so aus:

```
20  INPUT "EINGABE G"; G
30  FOR N = Ø TO G: ? N: NEXT: END
```

Bei solchen Umwandlungen ist nur noch Ihre Erfahrung maßgebend, und die kommt erst mit der Zeit. Seien Sie nicht ungeduldig; irgendwann ist auch Ihre Programmierung schnell und kurz. Für den Anfang jedoch reicht auch die 1. Lösung, denn Sie wollen schließlich die ersten Ergebnisse sehen.

Und die Wirkung ist bei beiden Programmversionen gleich.

Doch kommen wir zum Thema zurück.

Flußdiagramme sind heute bei der Programmierung kaum wegzudenken. In der Groß-EDV dienen sie zur Dokumentation und Übersichtlichkeit der Programme, und Sie als Anfänger werden rasch die Vorteile erkennen. Auf einem Blatt Papier kann man grafisch (und natürlich mit Bleistift geschrieben, um Korrekturen zu vereinfachen) wesentlich mehr übersehen, als in einer Programmliste.

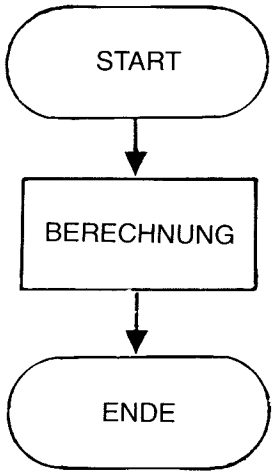
Gleichzeitig ist ein Ablaufdiagramm sprachenunabhängig. Ob Sie es in Maschinensprache oder BASIC oder sonst einer Programmiersprache ausführen, bleibt gleich.

Übrigens: Ob Sie sich an die genormten Symbole halten oder nicht:

Wichtig ist, daß Sie die Kästchen groß genug zeichnen, um alle Ausführungsbefehle einzuschreiben.

Zeichnen Sie ebenso möglichst viele Richtungspfeile an die Verbindungslinien. Sie geraten sonst leicht bei kompliziertem Aufbau auf den falschen „Pfad". Lassen Sie genug Platz zwischen den einzelnen Kästchen um gegebenenfalls später ein oder zwei Symbole nachtragen zu können.

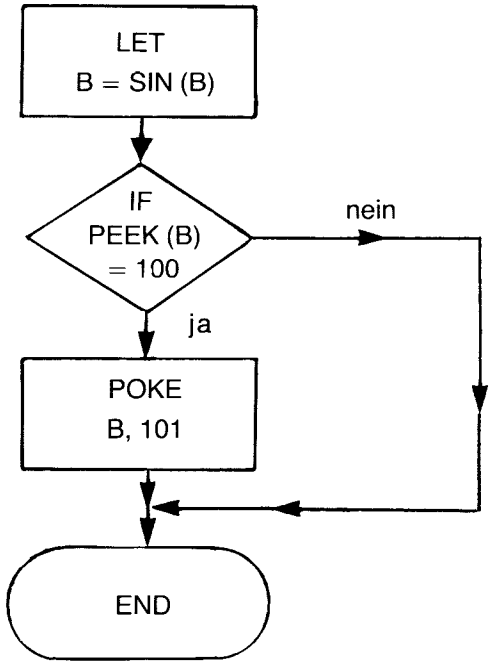
Gehen Sie genug ins Detail bei jedem Diagrammschritt. Es reicht nicht, wenn Sie nur 3 Symbole benutzen, wie z. B.:



FALSCH

Verhaspeln Sie sich aber auch nicht, indem Sie versuchen, die Symbole den BASIC-Befehlen gleichzustellen.

Beispiel:



FALSCH

Versuchen Sie in **Ihrer** Sprache, das Problem zu unterteilen. Benutzen Sie dabei ruhig Begriffe, die eine für Sie geschlossene Einheit bilden, wie z. B.: EINGABE DER WERTE A UND B UND PRÜFEN AUF RICHTIGKEIT.

Schreiben Sie Formeln so, wie sie im Mathematikbuch stehen:

$$X_{1/2} = -\frac{P}{2} \pm \frac{P^2}{4} - 9$$

8.3 Das Programm entsteht

Nachdem Sie nun die Vorarbeit geleistet haben und wissen, was Sie programmieren wollen, beginnt die eigentliche Arbeit am Rechner.

An dieser Stelle ein Tip, der Ihnen viel Ärger ersparen kann:

1. Geben Sie zu Beginn dem Programm einen Namen. Dieser Name sollte möglichst aus einem Wort mit nicht mehr als 12 Buchstaben bestehen.
2. Nehmen Sie sich 3 leere Kassetten, die Sie mit diesem Namen beschriften (Bleistift).
3. Speichern Sie auch schon während der Programmierung Ihr bisher erstelltes Programmteil auf jeweils die älteste Kassette ab (Reihenfolge: Kass 1, Kass 2, Kass 3, Kass 1, Kass 2 . . .), so daß Sie zumindest die letzten drei Arbeitsschritte nachvollziehen können, wenn Ihr Rechner durch Stromausfall o.ä. ausfällt.
4. Speichern Sie das Programm(-teil) immer mit seinem Namen ab.
   Schreiben Sie hinter diesem Namen ein Kennzeichen, daß Ihnen sagt, um welche Version Ihres Programmes es sich handelt.

   Wir empfehlen Ihnen folgende Kennzeichnung:

```
X        YY
|        |
Art      Lfd. Nr. (ØØ–99)
```

P = Programmteil
T = Testphase
R = Fertiges Programm

Beispiel:
```
SAVE "HAUSHALT PØ1"   (Programmteil 1)
SAVE "HAUSHALT PØ2"   (Programmteil 1 + 2)
SAVE "HAUSHALT PØ3"   (Programmteil 1 + 2 + 3)
SAVE "HAUSHALT TØ1"   (Fertiges Programm Test 1)
SAVE "HAUSHALT TØ2"   (Fertiges Programm Test 2)
SAVE "HAUSHALT RØ1"   (Fertiges Programm getestet und o.k.)
```

Diese oben angegebene Vorgehensweise hat sich als sehr vorteilhaft erwiesen. Zum Einen sind Sie vor allen Defekten des Rechners gefeit, da Sie eine Kopie des letzten Programmschrittes immer auf Kassette haben. Zum Anderen ist durch die Kennzeichnung des Programmnamens gewährleistet, daß Sie mit den verschiedenen Kassetten (und das werden immer mehr) nicht durcheinanderkommen.
Sobald Ihr Programm fehlerfrei läuft, speichern Sie es mindestens 2 Mal hintereinander auf der gleichen Kassette ab. Benutzen Sie dazu folgenden Befehl:

```
FORI = ØTO1: SAVE "PROGRAMMNAME RØ1": NEXT
```

Überprüfen Sie die korrekte Abspeicherung durch

```
VERIFY
```

Ist Ihr Programm fehlerfrei gespeichert, so löschen Sie alle vorherigen Versionen mit Pxx und Txx.

Sollten Sie zu einem späteren Zeitpunkt Ihr Programm noch ändern und sei es auch nur die kleinste Modifikation, so erhöhen Sie in jedem Fall die laufende Nummer um 1.

Wenn Sie alle Punkte gewissenhaft beachten, wird Ihr Archiv stets auf dem neuesten Stand sein und Sie werden wesentlich mehr Freude mit Ihrem Rechner haben.

8.4 Da stimmt doch was nicht

(Fehlersuche und Beseitigung)

Sehr viel Zeit muß im allgemeinen in einem Testlauf investiert werden, nachdem das Programm, oder auch das Teilprogramm erstellt wurde.

Ein auf Anhieb fehlerfrei funktionierendes Programm gibt es wohl nicht, sobald es auch nur etwas komplizierter ist.

Bei der Testphase ist es dann angebracht, das Programm in Module (Teilprogramme) aufzuteilen und diese separat zu testen. Durch ein vorübergehendes STOP oder END am Ende eines Modules oder Unterprogramms läßt sich der Programmablauf teilen und die Ergebnisse können somit leichter kontrolliert werden.
Bei rechenintensiven Programmen besteht oft Unsicherheit über die Richtigkeit der Ergebnisse.

Hier empfiehlt sich, auch wieder vorübergehend, einen PRINT-Befehl an der entscheidenden Stelle des Programms einzuflechten und damit die sonst im verborgenen ablaufenden Rechenwerte sichtbar zu machen und zu kontrollieren.

Das gleiche gilt natürlich auch bei komplexeren Dateienbehandlungen, wenn mit vielen Indizes gearbeitet wird.

Ein PRINT-Befehl zur Ausgabe der behandelten Elemente erhöht oft in starkem Maße die Übersichtlichkeit über den Programmablauf und dessen Ergebnisse.

8.5 Wie mache ich es schneller

Allgemein gibt es drei Verfahren, ein Programm schneller zu machen:

1. Verbessern des Ablaufgeschehens und der Berechnungsart.
   Hierbei können wir Ihnen keine Tips geben, da wir nicht wissen, welches Problem Sie lösen wollen. Solche Verfahren können jedoch die meiste Zeit einsparen.

2. Vermeiden von Unterprogrammen, Schleifen und Sprüngen.
   Grundsätzlich ist dieses Verfahren schneller. Seine Nachteile sind jedoch beträchtlich. Zum Einen wird Ihr Programm unverhältnismäßig groß und sprengt leicht die Grenzen Ihres Rechners. Zum Anderen verlieren Sie leicht die Übersicht über solche Programme. Auch sind solche Programme nur durch immensen Zeitaufwand zu verändern.

3. Syntaktische und Speicherungstechnische, rechnerbezogene Verbesserungen.
   Hierzu gehören eine Reihe von Punkten, die einzeln nicht sehr viel Zeit einsparen; summiert aber doch eine Menge ausmachen

a) entfernen Sie alle unnötigen Leerstellen und REM-Befehle aus Ihrem Programm;
b) Verwenden Sie Variablen anstelle von Konstanten. Die Umwandlung einer Konstanten in das rechnerinterne Format benötigt wesentlich mehr Zeit als der Zugriff auf den Wert einer Variablen.
c) Dimensionieren Sie Ihre Variablen und Felder stets als erstes im Programm. Gehen Sie dabei in der Reihenfolge der Benutzungshäufigkeit vor.
d) Schreiben Sie NEXT ohne die Indexvariable.
e) Schreiben Sie möglichst viele Statements in einer Zeile.

Beispiel:

```
  5 RESTORE
  6 DATA 9,10
 10 REM ungünstiges Beispiel
 20 REM Berechnung der Kombination
 30 GOTO 100
 40 P = 1
 50 FORI = 1 TOX
 60 P = P * I
 70 NEXT I
 80 RETURN
100 READ X
110 GOSUB 40
120 A = P
130 READ X
```

```
140  GOSUB 40
150  B = P
160  PRINT B/A
```

Zeit ungefähr 170 m.sec.

```
  5 RESTORE: DATA 9, 10
 40 READX: GOSUB 100: A = P: READX: GOSUB 100: PRINT P/A: GOTO 63997
100 P = 1: FORI = 1 TOX: P = P * I: NEXT: RETURN
```

Zeit ungefähr 150 m.sec.

ANHANG I

LISTE der Befehle des COMMODORE-BASIC

Befehl	Beispiel	Erklärung Seite
ABS	B = ABS (A)	8
AND	A = B AND C	10
	IF A = B AND C = 3 THEN . .	10
ASC	B = ASC("Q")	32
ATN	A = ATN (B)	9
CHR$	B$ = CHR$ (49)	32
CLOSE	CLOSE 1	28
CMD	CMD 3	46
COS	B = COS (A)	9
DATA	DATA 9, "PETER"	20
DEFFN	DEFFNA(X) = SIN (X)/X	27
DIM	DIM B%(9), C2$(3,3)	25
END	END	16
EXP	B = EXP(A)	9
FN	B = FNC/(Z)	27
FOR . . . NEXT	FOR I = TO 100 STEP 2	24
FRE	?FRE(0)	
GET	GET IN$	19
GET#	GET # 3, IN$	45
GOSUB	GOSUB 1000	22
GOTO	GOTO 170	21
IF . . . THEN	IF A = 5 THEN A = 6	10
INPUT	INPUT "WERT"; A	19
INPUT #	INPUT# 1, A	45
INT	B = INT (A)	8
LEFT$	A$ = LEFT$ (B$,3)	32
LEN	A = LEN (A$)	31
LET	LET A = 5 * 6	17
LOAD	LOAD "KURVE"	29
LOG	B = LOG (A)	9
MID$	A$ = MID$ (B$,2,2)	33
NEXT	siehe FOR . . . NEXT	
NOT	B = NOT B	10
ON . . . GOSUB	ON A–5 GOSUB 10, 20, 30	23
ON . . . GOTO	ON A–6 GOTO 20, 30	22
OR	A = A OR C	10
	IF A = B OR C = 3 THEN . . .	23
OPEN	OPEN 1,1,1, "DATEN"	27
PEEK	A = PEEK (B)	35
POKE	POKE B, A	35
PRINT	PRINT A$; "="; B$	17
PRINT #	PRINT # 5, A$; "="; B$	19
READ	READ A, A$	20
REM	REM ERKLAERUNG	16
RESTORE	RESTORE	20
RETURN	RETURN	24
RIGHT$	B$ = RIGHT$(A$,5)	33
RND	B = RND (TI)	9
SAVE	SAVE "VERSUCH", 2	28

Befehl	Beispiel	Erklärung Seite
SGN	B = SGN (A)	7
SIN	B = SIN (A)	8
SQR	B = SQR (A)	8
STEP	siehe FOR . . . NEXT	
STOP	STOP	16
STR$	A$ = STR$ (A)	31
SYS	SYS (64824)	34
TAN	A = TAN (B)	9
THEN	siehe IF . . . THEN	10
USR	A = USR (B)	34
VAL	A = VAL (A$)	31
VERIFY	VERIFY	30
WAIT	WAIT A, B, C	133
+	A = A + B	14
−	A = A − B	14
*	A = A * B	14
/	A = A / B	14
π	A = π	14
?	? A	17
↑	A = A ↑ B	14

RESERVIERTE VARIABLEN

TI	interne Zeit in Sechzigstelsekunden	39
TI$	interne Zeit in Stunden, Minuten, Sekunden	39
ST	Status	122

Umwandlungstabelle der Dezimalzahlen 1 - 127 in Hexadezimal; ASCII und Bildschirmcode

DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM	DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM	DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM	DEZIMAL	HEXA-DEZIMAL	ASCII	BILDSCHIRM
0	00	NULL	@	33	21	!	!	66	42	B	B	99	63	c	c
1	01	SOH	A	34	22	"	"	67	43	C	C	100	64	d	d
2	02	STX	B	35	23	≠	≠	68	44	D	D	101	65	e	e
3	03	ETX	C	36	24	$	$	69	45	E	E	102	66	f	f
4	04	EOT	D	37	25	%	%	70	46	F	F	103	67	g	g
5	05	ENQ	E	38	26	&	&	71	47	G	G	104	68	h	h
6	06	ACK	F	39	27	'	'	72	48	H	H	105	69	i	i
7	07	BL	G	40	28	(	(	73	49	I	I	106	6A	j	j
8	08	BS	H	41	29	)	)	74	4A	J	J	107	6B	k	k
9	09	HT	I	42	2A	*	*	75	4B	K	K	108	6C	l	l
10	0A	LF	J	43	2B	+	+	76	4C	L	L	109	6D	m	m
11	0B	VT	K	44	2C	,	,	77	4D	M	M	110	6E	n	n
12	0C	FF	L	45	2D	–	–	78	4E	N	N	111	6F	o	o
13	0D	CR	M	46	2E	.	.	79	4F	O	O	112	70	p	p
14	0E	LF	N	47	2F	/	/	80	50	P	P	113	71	q	q
15	0F	SI	O	48	30	0	0	81	51	Q	Q	114	72	r	r
16	10	DLE	P	49	31	1	1	82	52	R	R	115	73	s	s
17	11	DC_1	Q	50	32	2	2	83	53	S	S	116	74	t	t
18	12	DC_2	R	51	33	3	3	84	54	T	T	117	75	u	u
19	13	DC_3	S	52	34	4	4	85	55	U	U	118	76	v	v
20	14	DC_4	T	53	35	5	5	86	56	V	V	119	77	w	w
21	15	NAK	U	54	36	6	6	87	57	W	W	120	78	x	x
22	16	SYC	V	55	37	7	7	88	58	X	X	121	79	y	y
23	17	ETB	W	56	38	8	8	89	59	Y	Y	122	7A	z	z
24	18	CAN	X	57	39	9	9	90	5A	Z	Z	123	7B	{	♦
25	19	EM	Y	58	3A	:	:	91	5B	[	[	124	7C	:	:
26	1A	SUB	Z	59	3B	;	;	92	5C	/	/	125	7D	}	}
27	1B	ESC	[	60	3C	<	<	93	5D	]	]	126	7E	~	~
28	1C	FS	/	61	3D	=	=	94	5E	↑	π	127	7F	DEL	♠
29	1D	GS	]	62	3E	>	>	95	5F	←	♠				
30	1E	RS	↑	63	3F	?	?	96	60	'	'				
31	1F	US	←	64	40	@	–	97	61	a	A				
32	20	␣	␣	65	41	A	♠	98	62	b	B				

ANHANG III. LISTE DER FEHLERMELDUNGEN

?BAD SUBSCRIPT	Ansprache eines Matrix-Elementes, welches außerhalb der Dimensionierung der Matrix liegt, z.B. LET A (1,1,1,) = Z wenn A dimensioniert wurde: DIM A (2,29).
?BAD DATA	Versuch, Strings in normale Variable aus einem Peripherie-Gerät einzulesen.
?CAN'T CONTINUE	Nach einer Fehlermeldung oder einer Programmänderung ist CONTINUE nicht möglich.
?DIVISION BY ZERO	Division durch Ø unzulässig.
?ILLEGAL DIRECT	INPUT, GET und DEFFN kann nicht direkt, sondern nur im Programm ausgeführt werden.
?NEXT WITHOUT FOR	Bei der Programmschleife fehlt das entscheidende FOR-STATEMENT.
?OUT OF DATA	Mehr READ-Anweisungen als DATA-Anweisungen sind vorhanden.
?OUT OF MEMORY	Vorhandener Speicherplatz reicht nicht aus.
?OVERFLOW	Ergebnis oder Zwischenergebnis einer Rechnung war größer als 1Ø↑38. Bei Underflow erscheint "Ø" ohne einen Fehlerhinweis. Das Programm wird hierbei fortgesetzt.
?RETURN WITHOUT GOSUB	Subroutine ohne GOSUB.
?REDIM'D ARRAY	Ein Feld kann nicht zweimal dimensioniert werden.
?STRING TO LONG	Zeichenkette länger als 255 Zeichen.
?SYNTAX	SYNTAX-Fehler (der Befehl ist fehlerhaft oder existiert nicht). Klammer, Buchstaben bzw. Zeichen fehlen in einer Programmzeile.
?TYPE MISMATCH	Behandlung normaler Variablen als String oder umgekehrt.
?UNDEF'D STATEMENT	GOTO, GOSUB oder THEN zu einer nicht existenten Zeilennummer.
?UNDEF'D FUNCTION	Nicht definierte Funktion wurde aufgerufen.

Fehlermeldung ohne Programmunterbrechung

?REDO FROM START	Versuch, nichtnumerische Daten bei einer INPUT-Operation vom Tastenfeld in eine normale Variable einzulesen. Diese Fehlermeldung fordert Sie auf, die Eingabe zu wiederholen.

IV. Zeichenvorrat der Tastatur

V. Technische Daten des cbm

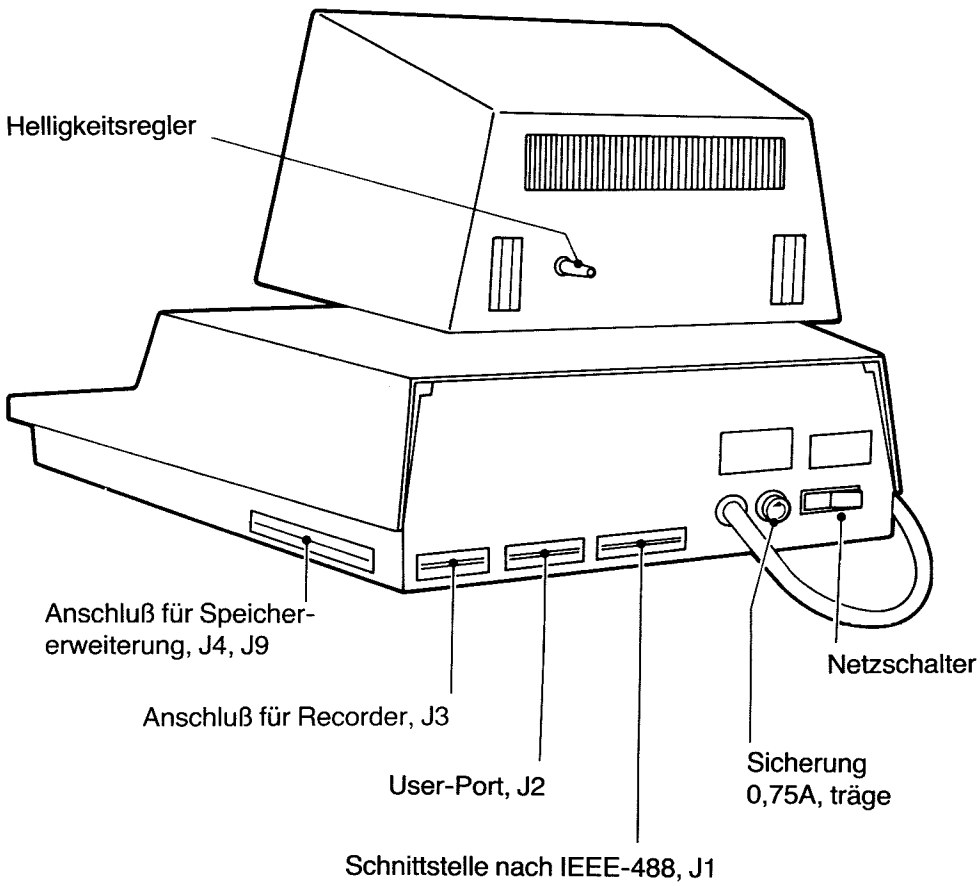
ANHANG V

Höhe	: 36 cm
Breite	: 42 cm
Tiefe	: 47 cm
Gewicht	: 20 kg
Spannung	: 220 V ± 10% / 50 Hz
Verbrauch	: 250 W
Bildschirm	
Diagonale	: 22 cm
Zeilen	: 25
Spalten	: 40
Zeichenauflösung	: 8 * 8 Punktmatrix vorprogrammiert
Darstellungsart	: Hell auf dunklem Grund (programmierbar dunkel auf hellem Grund)
Sonderzeichen	: 66 grafische Zeichen
Programmierung	
Sprache	: BASIC festgespeichert in 8 kB ROM
Anzahl Befehle	: 64, davon 19 mathematisch, 8 Textverarbeitung, 11 Peripheriebefehle
Kapazität	: 16 bzw. 32 kB
Kassette	: Rekorder für Standard Musikkassetten
Aufzeichnungsformat	: Einspur-Frequenzmodulation nach eigenem Standard
Anschlüsse	: (1 * IEC-Bus) 5 (1 * 8bit Parallel mit 2-Draht Handshake) (1 * 1. Kassettenrekorder) (1 * Speichererweiterungsausgang) (1 * 2. Kassettenrekorder)
Prozessor	: 6502
Taktfrequenz	: 1 MHz

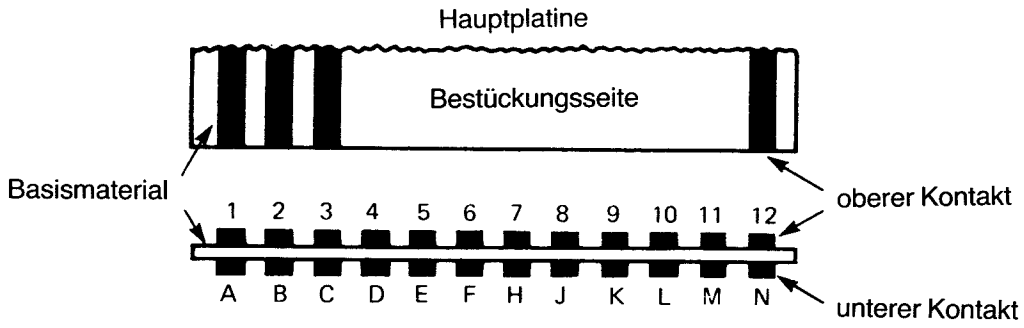
VI. SCHNITTSTELLENBESCHREIBUNG

Tabellen

1. LAGE DER SCHNITTSTELLEN UND BEZEICHNUNG DER KONTAKTE



J1 bis J4 sind Anschlüsse für Direktstecker auf der Hauptplatine. Die Bezeichnung soll durch untenstehendes Beispiel (J1 und J2) verdeutlicht werden:



2. IEEE-488 INTERFACE (STECKVERBINDER J1)

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). Codierschlitze befinden sich zwischen den Kontakten 2–3 und 9–10.

Fan-out und Impedanzanpassung entsprechen IEEE-488, (eine TTL-Last, 130 pF).

(Tabelle 1)

Kontakt-benennung	Standard IEEE	IEEE-Bezeichnung
Oben:		
1	1	DI01
2	2	DI02
3	3	DI03
4	4	DI04
5	5	EOI
6	6	DAV
7	7	NRFD
8	8	NDAC
9	9	IFC
10	10	SRQ
11	11	ATN
12	12	GND (Chassis und Kabelabschirmung)
Unten:		
A	13	DI05
B	14	DI06
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)
L	22	GND (SRQ)
M	23	GND (ATN)
N	24	GND (DI01–8)

(Tabelle 1)

3. USER-PORT (STECKVERBINDER 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.

Pinbelegung siehe Tabelle 2.

3.1. Interface-Baustein

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-I/0 Leitungen (PA0 bis PA7), einer Eingabe-Handshake Leitung (CA1) für diese acht Leitungen, die auch für andere flankengetriggerte Eingaben verwendet werden kann, und einer sehr starken Verbindung CB2. Diese hat die meisten Fähigkeiten von CA1, kann aber 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 siehe Datenblatt des 6522.

Adressierung des VIA siehe Tabelle 3.

Kontakt	Signal	Beschreibung
1	GND	Masse (Digital)
2	TV-Video-drive	Video-output für externen Bildschirm. Wird mit Testroutine zur Überprüfung des Videoteils verwendet.
3	IEEE-SRQ	Dieser Anschluß wird von der Testroutine zur Überprüfung von SRQ benötigt.
4	IEEE-EOI	Dieser Anschluß wird von der Testroutine zur Überprüfung von EOI benötigt.
5	Abfrage TEST	Ist dieser Anschluß während Power-on auf LOW, dann erfolgt ein Sprung in die Diagnose-(Test)-routine anstatt zu BASIC.
6	# 1 READ	Dient der Testroutine zur Überprüfung der Lesefunktion für Recorder #1.
7	# 2 READ	Dient der Testroutine zur Überprüfung der Lesefunktion für Recorder #2.
8	# 1, # 2 WRITE	Dient der Testroutine zur Überprüfung der Schreibfunktion beider Recorderanschlüsse.
9	TV-VERT	Vertikalsynchronsignal (60 Hz). Wird von Testroutine überprüft.
10	TV-HOR	Horizontalsignal. Wird von Testroutine überprüft.
11	GND	Masse (Digital)
12	GND	Masse (Digital)
A	GND	Masse (Digital)
B	CA1	Flankengetriggerter Anschluß des VIA 6522.
C	PAØ	PAØ bis PA7 sind einzeln als Input- oder Outputleitung programmierbar. Sie sind direkt mit dem VIA 6522 verbunden.
D	PA1	
E	PA2	
F	PA3	
H	PA4	
J	PA5	
K	PA6	
L	PA7	
M	CB2	Anschluß CB2 des VIA 6522
N	GND	Masse (Digital)

(Tabelle 2) User-Port

Dez.	Hex.	$E840+	Adressierte Speicherstelle/Tätigkeit
59456	E840	0000	Output Register für I/0-Port B (ORB)
59457	E841	0001	Output Register für I/0-Port A (ORA) mit Handshake
59458	E842	0010	I/0-Port B Datenrichtungsregister (DDRB)
59459	E843	0011	I/0-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 reset Interrupt-Flag. 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/0-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/0-Port A ohne Handshake.

(Tabelle 3) Adressen des VIA

3.2. Programmierung des User-Ports

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

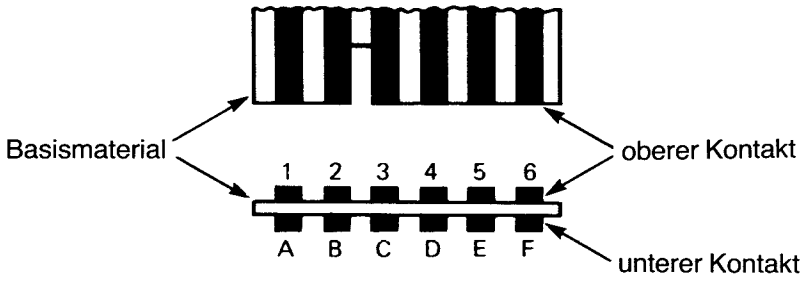
(Tabelle 4)

4. INTERFACE FÜR EXTERNEN RECORDER (STECKVERBINDER J3)

Dieses Interface ist 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.

Dieser Port sollte ausschließlich für den zweiten Kassettenrecorder von Commodore benutzt werden. Verbindungen anderer Art sind für den Benutzer zu riskant. Bitte beachten Sie, daß die Spannung von +5 Volt nicht als Spannungsquelle für andere externe Geräte vorgesehen ist.



Bemerkung: Bei manchen früheren Geräten sind die Anschlüsse 1 bis 6 nicht mit A und F verbunden.

Anschluß	Bez.	Beschreibung
A–1	GND	Masse (Digital)
B–2	+5 V	Spannungsversorgung für die Recorder
C–3	MOTOR	+6 V (vom Computer gesteuert) für Motor des Recorders
D–4	READ	Lesen von Kassette
E–5	WRITE	Schreiben auf Kassette
F–6	SENSE	Leitung stellt fest, ob eine der Tasten REW, FFWD oder PLAY gedrückt ist (Rückmeldeleitung).

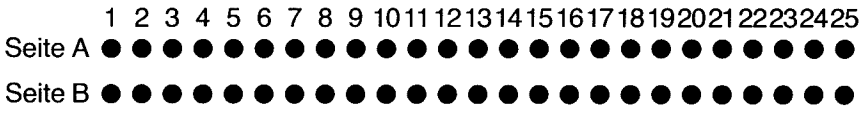
(Tabelle 5)

5. ANSCHLÜSSE FÜR SPEICHERERWEITERUNG (J4, J9) UND FÜR EXTERNE PLATINEN (J1Ø, J11)

J4 und J9 für Stecker mit 2x25 Kontakten, J1Ø und J11 mit 2x7 Kontakten, 0,1" (2,54 mm Rastermaß).

J1Ø		J11	
●●●●●●● 1 2 3 4 5 6 7		●●●●●●● 1 2 3 4 5 6 7	
1	–5V unreg.	1	+9V unreg.
2	–5V unreg.	2	Codierung
3	Codierung	3	Codierung
4	+12V unreg.	4	+9V unreg.
5	+12V unreg.	5	Masse
6	Masse	6	+9V unreg.
7	Masse	7	Masse

(Tabelle 6a) Stecker für externe Platinen



J4 und J9 (Seite B sind die Masseanschlüsse, Seite A siehe Tabelle 6b)

Anschl.	Bez.	Beschreibung
J9 – 1	GND	Masse (digital)
J9 – 2	BAØ	
J9 – 3	BA1	
J9 – 4	BA2	
J9 – 5	BA3	
J9 – 6	BA4	
J9 – 7	BA5	Gepufferte Adreßbits AØ bis A15
J9 – 8	BA6	für Speichererweiterung
J9 – 9	BA7	
J9 – 10	BA8	
J9 – 11	BA9	
J9 – 12	BA1Ø	
J9 – 13	BA11	
J9 – 14	BA12	
J9 – 15	BA13	
J9 – 16	BA14	
J9 – 17	BA15	
J4 – 10	$\overline{\text{SEL2}}$	2ØØØ – 2FFF
J4 – 11	$\overline{\text{SEL3}}$	3ØØØ – 3FFF
J4 – 12	$\overline{\text{SEL4}}$	4ØØØ – 4FFF wählen je einen
J4 – 13	$\overline{\text{SEL5}}$	5ØØØ – 5FFF 4 kByte-Block an.
J4 – 14	$\overline{\text{SEL6}}$	6ØØØ – 6FFF
J4 – 15	$\overline{\text{SEL7}}$	7ØØØ – 7FFF
J4 – 16	$\overline{\text{SEL8}}$	8ØØØ – 8FFF
J4 – 17	$\overline{\text{SEL9}}$	9ØØØ – 9FFF
J4 – 18	$\overline{\text{SELA}}$	AØØØ – AFFF
J4 – 19	$\overline{\text{SELB}}$	BØØØ – BFFF
J4 – 25	GND	Masse (digital)
J4 – 22	$\overline{\text{RES}}$	Resetleitung (Power-on-reset)
J9 – 19	$\overline{\text{IRQ}}$	Interrupt-Request-Leitung des 6502
J9 – 21	$\overline{\text{B 2}}$	Takt 2.
J9 – 22	R/W	Lese-Schreib-Leitung des 6502, gepuffert
J4 – 20	$\overline{\text{RAS}}$	RAS für die RAM's
J4 – 21	$\overline{\text{CAS}}$	CAS für die RAM's
J4 – 2	BDØ	
J4 – 3	BD1	
J4 – 4	BD2	
J4 – 5	BD3	Gepufferte Datenbits DØ bis D7
J4 – 6	BD4	
J4 – 7	BD5	
J4 – 8	BD6	
J4 – 9	BD7	
J9 – 18	SYNC	–
J9 – 20	–	Für RAM-Verwaltung
J9 – 23	$\overline{\text{R/W}}$	Lese-Schreibleitung des 6502, gepuffert
J9 – 24	$\overline{\text{DMA}}$	Leitung für direkten Speicherzugriff
J4 – 1	GND	Masse (digital)
J4 – 23	$\overline{\text{RDY}}$	Ready-Leitung des 6502
J4 – 24	$\overline{\text{NMI}}$	Nicht maskierbarer Interrupt
J9 – 25	GND	Masse (digital)

(Tabelle 6b) Anschluß für Speichererweiterung

INPUT/OUTPUT

1. BEFEHLE DES ERWEITERTEN BASIC

An diesem Punkt ist der Benutzer sicher bereits mit dem Gebrauch der Befehle INPUT und PRINT vertraut. INPUT erlaubt die Dateneingabe über die Tastatur und PRINT erlaubt die Ausgabe oder die Anzeige von Daten. Diese Befehle sind Teil des Standard-BASIC.

Um die Flexibilität des Computersystems zu erhöhen, wurden zu den klassischen BASIC-Befehlen einige Befehle hinzugefügt, und zukünftige Geräte von Commodore werden die sich daraus ergebenden Möglichkeiten weitgehend ausschöpfen. Im allgemeinen wird die Flexibilität des Datenaustauschs zwischen dem Computer und den Peripheriegeräten durch den Gebrauch zusätzlicher Befehle erhöht.

Folgende Befehle des erweiterten BASIC dienen der Kommunikation mit der Peripherie:

a) OPEN Eröffnen bzw. Schließen eines logischen Files
CLOSE

b) PRINT # Daten vom Computer auf Ausgabegerät schreiben

c) CMD Wie PRINT #, CMD hinterläßt aber das IEEE-Gerät nach Ausführung des Befehls als aktiven Hörer am Bus.

d) INPUT # liest Daten von Peripheriegerät in den Rechner ein

e) GET # Der Computer holt ein einzelnes Zeichen vom Peripheriegerät.

2. PARAMETER VON I/0-BEFEHLEN

Um die zusätzlichen Befehle, wie sie in Punkt 1 erwähnt werden, durchzuführen, müssen bis zu vier Parameter beachtet werden.

a) # des logischen Files (LF)

b) Gerätenummer (D)

c) Sekundäradresse (SA)

d) File-Name (FN)

Diese Parameter können beispielsweise im OPEN-Befehl erscheinen:

OPEN # LF,D,SA,FN

Diese Parameter werden in 2.1. bis 2.4. definiert und erläutert.

2.1. Logische Files

Files werden benötigt, um Daten zu speichern oder zu holen, wie etwa im Falle eines Magnetband- oder eines Plattenfiles.

Dieser Gedanke kann erweitert werden, indem man jedes Gerät, das Daten empfangen und/oder senden kann, als logisches File ansieht. Für das Betriebssystem des Rechners können Daten von/zu einem beliebigen Speichersystem kommen/gehen.

Ist etwa ein externes Digitalvoltmeter so angeschlossen, daß es nach Bedarf Meßwerte über den IEEE-Bus senden kann, dann wird der Programmierer während des Voltmeter-Programms manchmal ein File öffnen und dem Voltmeter eine NF zuteilen müssen. Sobald dies erfolgt ist, kann der Rechner den Lesebefehl (INPUT#) benutzen, welcher die File-Nummer verwendet, um sich auf das Voltmeter zu beziehen. Werden keine weiteren Daten vom Voltmeter verlangt, kann das File geschlossen werden.

Ganz allgemein sind die Vorteile logischer Files die folgenden:

a) Jede aus Gerätenummer und Sekundäradresse bestehende Kombination kann innerhalb eines Programms eindeutig einer logischen Filenummer zugeordnet werden.

b) Auf Mehrfachfiles innerhalb eines einzigen Gerätes kann über verschiedene logische FN zugegriffen werden.

c) Sobald eine logische Filenummer in einer OPEN-Aussage innerhalb eines Programms angegeben wurde, braucht in den folgenden Input/Output Aussagen nur noch diese eine Nummer angegeben werden. Dadurch erübrigt sich die Notwendigkeit für eine wiederholte Angabe der Gerätenummer, der Sekundäradresse (wo erforderlich) und des Filenamens (wo erforderlich).

Obgleich es zulässig ist, viele logische Files in einem Programm zuzuweisen und zu gebrauchen, so muß das Betriebssystem des Rechners doch diejenigen Files, die augenblicklich im Programm gebraucht werden, auf dem laufenden halten. Die höchste Anzahl von Files, die vom Rechner zu einem beliebigen Zeitpunkt kontrolliert werden können, beträgt zehn. Bei mehr als 10 gleichzeitig offenen Files wird die Fehlermeldung ?TOO MANY FILES ERROR ausgegeben. Eine logische File-Nummer kann jede ganze Zahl im Bereich von 1 bis 255 sein.

2.2. Geräte-Nummern

Allen Geräten, mit denen der Rechner über den IEEE-Bus Verbindung hat, sind Gerätenummern zugeteilt. Die ersten vier davon beziehen sich auf die folgenden Peripheriegeräte:

Geräte #	Gerät
Ø	Tastatur
1	eingebauter Recorder
2	externer Recorder
3	Bildschirm

Alle anderen Geräte betrachtet der Computer automatisch als IEEE-Geräte, und die Kontrolle wird dem Gerät übertragen, dem zuvor eine Nummer zwischen 4 und 3Ø zugeteilt worden ist. Von speziellen Fällen abgesehen, wird zur Kommunikation mit jedem Gerät diesem eine bestimmte Nummer zugeteilt werden. Bei vielen IEEE-Geräten erfolgt die Zuteilung der Gerätenummer durch einen Schalter, oder, weniger aufwendig, Kurzschlußbrücken am IEEE-Gerät selbst.

2.3. Sekundäradressen

Die Benutzung einer Sekundäradresse ermöglicht es einem intelligenten Peripheriegerät, in jeder möglichen Betriebsart zu arbeiten. Im cbm 3022 Drucker gibt es beispielsweise sechs Sekundäradressen:

Sekundäradresse	Betriebsart
Ø	normal drucken
1	formatiert drucken
2	Daten dienen als Formatanweisung
3	Daten legen Anzahl der Zeilen/Seite fest
4	schriftliche Fehlermeldungen ein/aus
5	Daten definieren programmierbares Sonderzeichen

Kurz gesagt, können über die Veränderung der Sekundäradresse die Tätigkeitsmerkmale des Peripheriegeräts bei Bedarf vollkommen verändert werden. Viele IEEE-Geräte haben ihre eigenen besonderen Sekundäradressen-Vereinbarungen, die befolgt werden müssen. Besondere Vereinbarungen dieser Art ersehen sie aus dem Handbuch des betreffenden Geräts.

Beispielsweise gelten für die Recorder folgende Regeln für Sekundäradressen:

Sekundäradresse	Operation
Ø	Band wird eröffnet zum Lesen
1	Band wird eröffnet zum Schreiben
2	Band wird eröffnet zum Schreiben mit Schreiben eines EOT-Zeichens ("end of tape" – Bandende), sobald das File geschlossen wird.

Die Sekundäradresse kann zwischen Ø und 31 liegen.

2.4. Filenamen

Bei Geräten mit freiem Speicherzugriff, die für mehr als ein File zugänglich sind, ist es unbedingt erforderlich, Namen zu vergeben, um die Files unterscheiden zu können. Im Falle von Bändern sollte man ebenfalls Filenamen vergeben, auch wenn es nur ein File auf dem Band gibt. Dies erleichtert die Identifizierung der Bänder.

Für die beiden Recorder kann der Filename aus einer Kombination von bis zu 128 Zeichen bestehen.

Bei der Suche nach einem bestimmten Filenamen wird nur auf Übereinstimmung mit dem Suchbegriff geachtet.

Nehmen wir als Beispiel, daß beim Schreiben ein aus sechs Buchstaben bestehender Filename, nämlich "GEWINN", festgelegt wurde. Dieser Begriff kann auch durch eine Abkürzung, etwa "GE", gesucht werden. Der erste File-Header (Titel) mit diesen zwei hintereinander folgenden Zeichen wird dann als der gesuchte eröffnet. Dies träfe natürlich auch auf nicht gesuchte Files, wie beispielsweise die Files "GELD" oder "GEWERBESTEUER" zu.

Zur Vermeidung von Irrtümern sollte daher der vollständige Filename aufgeführt werden.

Bei anderen Geräten, die Filenamen verwenden, sollte die Gebrauchsanweisung für das Gerät beachtet werden, um zu wissen, was im einzelnen für den Gebrauch von Filenamen erforderlich ist.

3. KASSETTEN-FILES

Besondere Aufmerksamkeit widmet der Computer den beiden Recordern. Die Bandeinheiten sind dahingehend modifiziert, daß der Rechner die Motorbewegungen kontrollieren kann.

Er kann auch feststellen, ob eine der Tasten PLAY, REW oder FFWD (abspielen, schnelles Rückspulen, schneller Vorlauf) gedrückt wurde; dies geschieht mit Hilfe eines einzigen Schalters, der sich im Recorder befindet.

Beachten Sie bitte, daß der gleiche Schalter gebraucht wird, um alle drei Tasten abzutasten: wenn einer von ihnen heruntergedrückt ist, ist der Computer der Meinung, daß Sie die Taste PLAY gedrückt haben und reagiert entsprechend.

Bedingt durch den Mechanismus in der Bandeinheit müssen die Bänder von Hand zurückgespult, vorwärtsgespult, gestoppt, geladen und ausgeworfen werden. Die Bandeinheit muß gleichfalls von Hand in Schreibstellung gebracht werden, indem man die Aufnahmetaste (REC) entweder gleichzeitig mit PLAY oder zuvor drückt.

Der Computer übernimmt die Kontrolle über die Bewegungen des Bandes, sobald die entsprechenden Tasten des Recorders gedrückt worden sind. Meldungen des Betriebssystems teilen dem Benutzer mit, wann die Taste PLAY und wann die Tasten PLAY und REC gedrückt werden müssen.

3.1. File-Wiedergabetechniken

Die Datenstruktur auf den Magnetbändern gibt die Gewähr für größtmögliche Speicherdichte und Zuverlässigkeit der Aufnahme.

Daten werden als zwei verschiedene Frequenzen in zwei aufeinander folgenden Datenblocks aufgezeichnet. Alle Daten werden insgesamt wiederholt, und mit Hilfe von Software-Fehlersuchtechniken ist es möglich, die meisten Dropouts (kurze Unterbrechungen) durch das Betriebssystem korrigieren zu lassen, indem die überzähligen, im zweiten Datenblock gespeicherten Daten nutzbar gemacht werden.

Eine Geschwindigkeitskorrelationstechnik während des Lesens wird benutzt, um (a) zu korrigieren, daß die Recorder verschiedene Geschwindigkeiten haben, und (b) um die Dehnbarkeit der Bänder auszugleichen. Um die individuelle Start-Stop-Charakteristik zu kompensieren und richtig zwischen jedem Bandblock zu synchronisieren, wird ein Dauerton zwischen die Blocks geschrieben.

Sowohl Position als auch Geschwindigkeit des Bandes werden damit synchronisiert. Unterschiedliche Tonlängen werden zu Beginn und zwischen den Datenblocks des Bandes benutzt. Durch einen ungefähr 10 Sekunden dauernden Ton bei jeder Eröffnung eines Files gibt der Computer automatisch einen Leervorspann ein. Die einzelnen Bandblöcke sind durch kürzere Töne voneinander getrennt.

3.2. Fileüberschriften (Headers)

Beim Entwurf des Bandsystems wurde von der Annahme ausgegangen, daß der Teilnehmer oft mehr als ein Datenfile auf einem Band wiedergeben möchte. Um dies zu erleichtern und um die Filenamen deutlich zu kennzeichnen, werden sogenannte "headers" geschrieben. Die Fileüberschrift sieht genauso aus wie der normale Datenblock, außer daß das erste Symbol auf jedem Block des Bandes ein Erkennungssymbol enthält, durch das der Computer zwischen Programmblocks, Datenblocks, Fileüberschriften und Bandendeüberschriften unterscheiden kann. Der Rechner kann bis zu 128 Symbole für einen Filenamen in einer Fileüberschrift aufnehmen. Dies ist der Name nach dem gesucht wird und der den verschiedenen OPEN/CLOSE Wahlmöglichkeiten angepaßt wird.

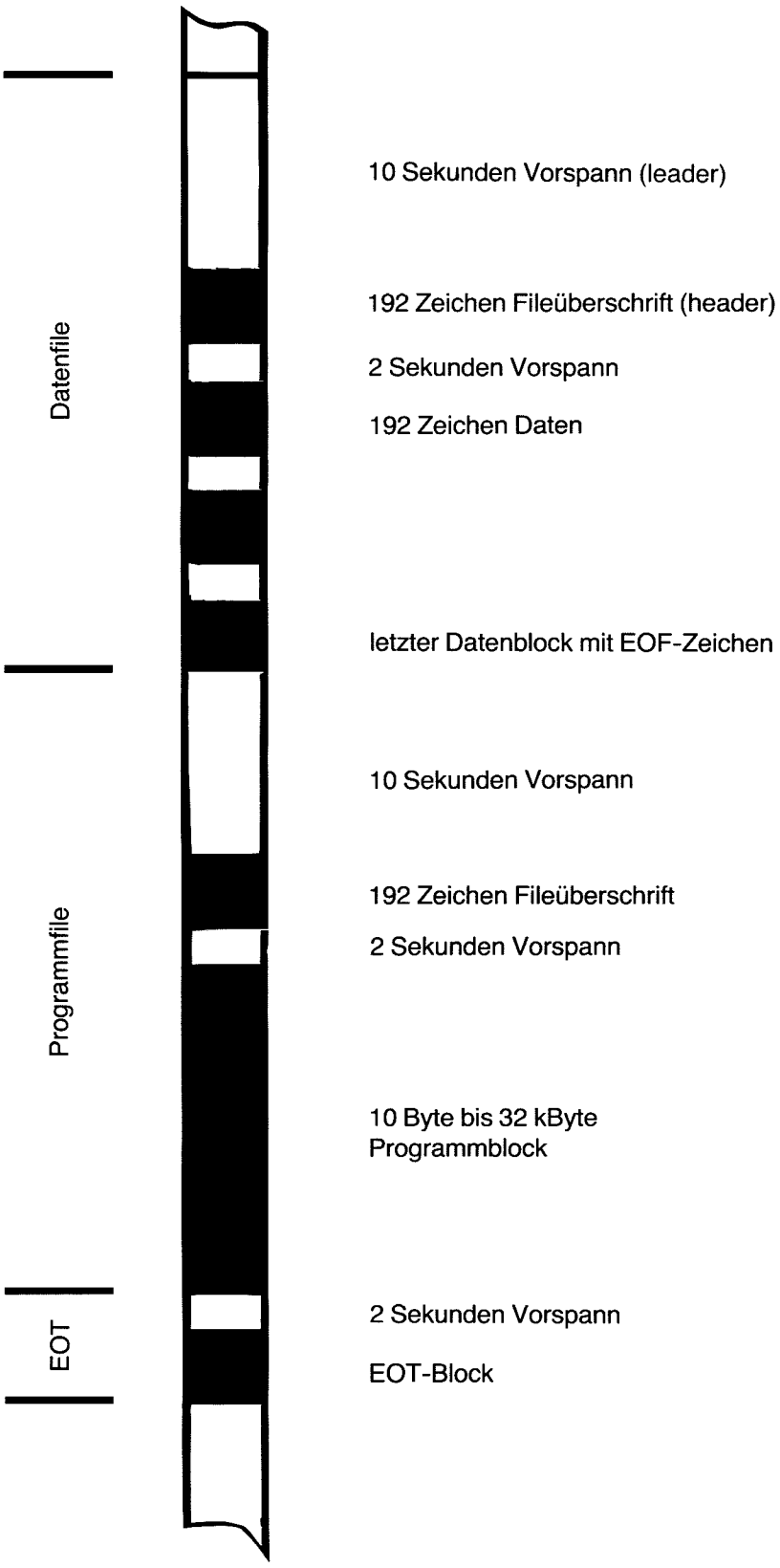
3.3 Puffer für den Recorder

Das Betriebssystem weist jedem der beiden Recorder einen Speicherblock als Datenpuffer zu. Ein aus 192 Speicherplätzen bestehender Puffer beginnt bei der Dezimaladresse 634 für Recorder # 1, gefolgt von einem Zwischenspeicher für 192 Zeichen ab Dezimaladresse 826 für Recorder # 2. Die Fileüberschrift für das Band wird zuerst in den Zwischenspeicher geholt. Bandfileüberschriften und Datenblocks sind daher 192 Zeichen lang.

Für Programme werden die Pufferspeicher nicht benutzt, sie werden direkt vom Hauptspeicher auf das Band geschrieben. Die Fileüberschrift für ein Programmfile enthält dazu Anfangs- und Endadresse des Programms. Das gesamte Programm wird wie üblich in zwei aufeinanderfolgenden redundanten Blocks auf das Band geschrieben.

3.4. Mehrfache Files

Will man mehrere Files auf einem Band haben, muß der Benutzer Files auf dem Band hinzufügen können und wissen, wann das Band zu Ende ist. Daher ist es wichtig, daß "Fileende" (EOF) und "Bandende" (EOT) vom Betriebssystem erkannt wird. Dem Datenfile wird ein EOF-Zeichen automatisch angehängt, um das Ende des jeweiligen Datenfiles zu kennzeichnen.



3 Beispiel für Datenaufzeichnung

Der Benutzer kann durch Überprüfung des Statusworts ST feststellen, ob ein EOF oder ein EOT vorliegt.

Im Fall von Programmfiles zeigt das Blockende gleichzeitig das Ende des Programmes an, da alle Daten immer in einem einzigen Block enthalten sind.

Um anzuzeigen, daß das Bandende erreicht ist, wird eine besondere getrennte Fileüberschrift eingeschrieben. Kommt man während der Suche nach Files zu diesem Punkt, so hält der Computer automatisch das Band an und zeigt dem Benutzer an: ?FILE NOT FOUND ERROR. Ein typisches Band mit mehreren Files könnte zuerst ein Datenfile, dann ein Programmfile enthalten, gefolgt von einer "end of tape"-Überschrift, wie am Beispiel 3 gezeigt wird.

4. File – Input-Output: Allgemeines

Diese Operationen können in drei Schritte unterteilt werden:

a) Eröffnung des Files – gibt dem Computer alle Informationen, die er über das File benötigt,

b) Lesen der Daten aus dem logischen File oder Schreiben der Daten,

c) Schließen des Files – der Computer beendet das logische File und desaktiviert das zugehörige Peripheriegerät.

Diese Schritte werden in den Abschnitten 5 bis 9 im einzelnen besprochen.

5. OPEN

Um BASIC Anweisungen über das zu bearbeitende File zu geben, ist es als erstes notwendig, das File zu eröffnen. Dies geschieht mit der folgenden Anweisung:

OPEN logische Filenummer, Gerät, Sekundäradresse, Filename

Die Anweisung besteht im einzelnen aus dem Befehl OPEN, gefolgt von der logischen Filenummer, dann der Nummer des Gerätes, dem das File zugeordnet ist. Es folgen die Sekundäradressendaten (falls vorhanden), die von BASIC dem File übertragen werden, und zuletzt der Name des physischen Files (falls vorhanden).

Diese Anweisung wird von BASIC interpretiert. Es können daher berechnete logische Filenummern, Gerätenummern oder Sekundäradressendaten benutzt werden. Diese Fähigkeit ist bei der Verwendung von Geräten, die Mehrfachfiles gestatten, wie z.B. Platten, äußerst nützlich.

Das Schlüsselwort OPEN und die logischen Filenummern sind unerläßlich, um das File zu eröffnen, also ein Gerät zu adressieren zur Vorbereitung auf Lesen (INPUT#) oder Schreiben (PRINT#). Die Gerätenummer ist frei wählbar; wird keine Gerätenummer angegeben, so wird von BASIC der Ersatzwert 1 eingesetzt. (Siehe Abschnitt 2.3.)

Der Filename für die Recorder ist nicht erforderlich, obgleich es vorzuziehen ist, wenn ein Name vorhanden ist. Für eine Floppy-Disk ist jedoch ein Name notwendig.

5.1. Beispiele für OPEN-Anweisungen

Die Anweisung OPEN 1,2,1 wird durch das Betriebssystem wie folgt interpretiert:

Parameter

LF = 1 Das logische File #1 wird eröffnet

D = 2 Das logische File #1 wird dem Recorder zugeordnet.

SA = 1 Recorder # 2 wird angewiesen, auf Band zu schreiben.

FN – Ein Name wurde dem File nicht zugeordnet.

Und so wird OPEN 3 interpretiert:

Parameter

LF = 3 Das logische File #3 wird eröffnet.

D – Das logische File # 1 wird Recorder # 2 zugeordnet.

SA – Recorder #1 wird angewiesen, vom Band zu lesen (Ersatzwert ist Ø).

FN – Kein Filename angegeben.

Wenn dem Drucker 3022 eine "4" als Gerätenummer zugeordnet ist, dann wird OPEN 12,4,1 wie folgt interpretiert:

Parameter

LF = 12 Das logische File #12 wird eröffnet.

D = 4 Das logische File #12 wird dem Gerät # 4 zugeordnet.

SA = 1 Der Drucker wird angewiesen, formatiert zu drucken.

FN – Filename nicht anwendbar.

5.2. LOAD

Einen Sonderfall des OPEN-Befehls stellt das Laden (LOAD) eines bekannten Files dar: LOAD hat folgendes allgemeines Format:

LOAD Name, Gerätenummer

Das Betriebssystem bewirkt automatisch ein OPEN, indem es die entsprechenden Sekundäradressen für LOAD benutzt. Ausgelöst durch OPEN, sucht das Ladegerät nach dem Programmnamen.

Wenn das Programm gefunden ist, wird es automatisch vom Gerät ausgelesen und in den Speicher eingeladen, wobei mit der in der Fileüberschrift angegebenen Adresse begonnen wird. Alle Lesefehler, die beim ersten Durchlauf durch das Programm auftreten, werden automatisch, falls möglich, während des zweiten Durchlaufs behoben.

Am Ende des Ladezyklus wird eine Kontrollsumme des gesamten Programms erstellt. Falls ein Kontrollsummenfehler oder ein nicht mehr zu verbessernder Lesefehler aufgetreten sein sollte, druckt das Betriebssystem automatisch ?LOAD ERROR und stoppt das Ladeprogramm. Erfolgt das Laden auf direkte Art, so impliziert dies CLR am Ende des Ladevorgangs. Dadurch werden alle Variablen initialisiert.

Steht der LOAD-Befehl in einem Programm, so wird das neue Programm an die Stelle eingeladen, die von dem aufrufenden Programm besetzt gewesen war, die Werte aller Variablen werden vom vorherigen Programm übernommen. Dadurch besteht die Möglichkeit, daß ein Programm ein anderes Programm aufruft und Parameter in das gerufene Programm übernommen werden.

Eine Beschränkung liegt darin, daß alle aufgerufenen Programme den Platzbedarf des Aufrufprogramms nicht übersteigen dürfen. Da das aufrufende Programm durch das neue überschrieben wird, ist es nicht ohne weiteres möglich, ein Hauptprogramm und verschiedene Subroutinen als Overlays zur Verfügung zu haben. Man erzielt jedoch die gleiche Wirkung bei geringem Geschwindigkeitsverlust, wenn man in jede Subroutine das Hauptprogramm einfügt.

Indem man mit Namen versehene Files mit Overlays kombiniert, können umfangreiche strukturierte Programme geschrieben werden.

5.3. VERIFY

Diese Anweisung ist eine Sonderform von LOAD. Man sollte sie nach jedem SAVE anwenden.

VERIFY bewirkt, daß BASIC alle Schritte eines LOAD-Programms ausführt, die gelesenen Daten aber nicht in den Speicher übernimmt. Stattdessen werden die Daten mit dem Speicherinhalt verglichen. Sollten beim ersten oder zweiten Datenblock Irrtümer auftreten, wird ?VERIFY ERROR ausgedruckt. Das bedeutet, daß das Programm neu auf die Kassette aufgezeichnet werden sollte (mit SAVE). Nach VERIFY hat das Statuswort (Variable ST) die folgenden Bedeutungen (Einzelheiten siehe Abschnitt 10.1.):

ST:	Bedeutung
4	kurzer Datenblock
8	langer Datenblock
16	verschiedene Daten
32	Kontrollsummenfehler

5.4. SAVE

SAVE führt ebenfalls automatisch OPEN und CLOSE aus. Der Befehl SAVE hat das Format:

SAVE''Name'', Geräte-Nummer

Ist das angesprochene Gerät einer der beiden Recorder, so initiiert das Betriebssystem automatisch einen Vorspann und eröffnet ein File mit dem entsprechenden Namen. Die Fileüberschrift enthält Anfangs- und Endadresse.

Handelt es sich um ein IEEE-488-Gerät, wird eine spezielle Eröffnungsinformation gesandt, die darauf hinweist, daß der Computer ein Programmfile sendet.

Das Programm wird dann direkt aus den Speicherstellen auf das Band oder den IEEE-488-Bus geschickt.

Bei SAVE auf einen der Recorder wird eine Kontrollsumme errechnet und ebenfalls auf der Kassette aufgezeichnet. Dann wird das ganze Programm noch einmal geschrieben (redundante Aufzeichnung). Am Ende des Programms wird das Band automatisch gestoppt und ist positioniert für die nächsten Daten.

5.5. IEEE-488: Besondere Eigenschaften

Bei Verwendung eines der beiden Recorder wird die Anfangs- und Endadresse des Programms in der Fileüberschrift eingespeichert bzw. aus dem Header geholt.

Um die IEEE-488-Daten wirksamer zu verwenden, wird bei SAVE die Anfangsadresse des Programms mit den ersten beiden Datenbits gesandt und bei LOAD von diesen Positionen abgerufen.

5.6. Bemerkungen zu IEEE-488 OPEN

Wählt OPEN ein Gerät mit einer Gerätenummer größer 3, so nimmt das Betriebssystem an, daß es sich um ein IEEE-488-Gerät handelt.

Gibt OPEN keinen Filenamen an, dann kommt keine Übertragung auf dem IEEE-488-Bus zustande. Ist ein Filename angegeben, sendet das Betriebssystem eine listen-attention-Folge an die im OPEN angegebene Gerätenummer mit einer Sekundäradresse, die das logische ODER (OR) von $FØ mit der im OPEN-Befehl angegebenen Sekundäradresse darstellt.

Die von Commodore gelieferten Peripheriegeräte, wie beispielsweise die Floppy-Disk, verwenden diese Sekundäradresse sowie den Filenamen, der dann an den Listener übermittelt wird, um die Daten später in das offene File zu übertragen.

6. OPEN und CLOSE bei Bandfiles

Bandfiles können eröffnet werden:

a) mit dem Ziel, vom Computer auf das Band zu schreiben,

b) mit dem Ziel, vom Band in den Computer einzulesen.

6.1. OPEN zum Schreiben auf Band

Das Flußdiagramm 6.1. erläutert die Wechselwirkung zwischen Computer und Benutzer bei Eröffnung eines Files zum Schreiben auf Band. Es gibt zwei Möglichkeiten:

a) OPEN – zum Schreiben eines Datenbandes

b) SAVE – Programmband schreiben.

Beachten Sie, daß bei Eröffnung des Bandfiles direkt über die Tastatur die Mitteilung: WRITING Name erscheint. Wird das File unter Programmkontrolle eröffnet und ist eine der Tasten PLAY, FFWD, REW gedrückt, so erscheint keine Mitteilung auf dem Bildschirm. Dadurch wird die vom laufenden Programm erzeugte Bildschirmanzeige nicht gestört.

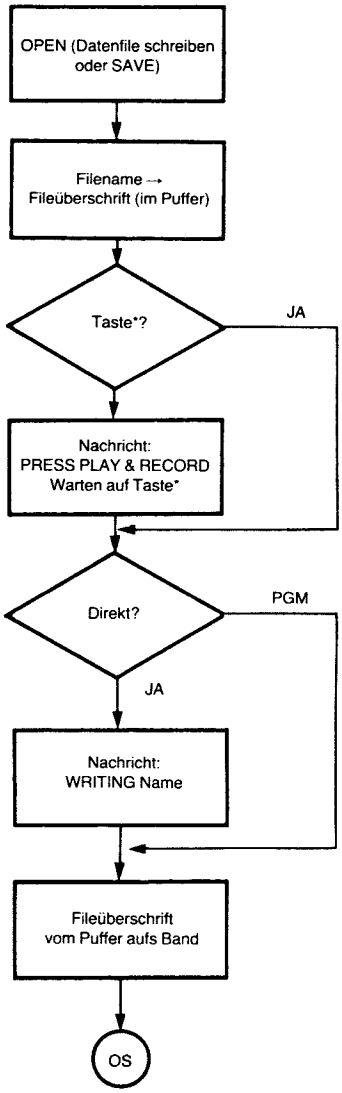
Für die drei Tasten PLAY, FFWD und REW gibt es nur eine Abfrageleitung. Das heißt, der Computer kann nicht feststellen, welche davon gedrückt ist. Die Taste REC wird überhaupt nicht abgefragt. Ein Teil der Verantwortung für das Betätigen der richtigen Tasten liegt also allein beim Benutzer.

6.2. OPEN zum Lesen vom Band

Das Flußdiagramm 6.2. erläutert diese Funktion. Der erste Block demonstriert die beiden Möglichkeiten:

a) Eröffnung zum Lesen eines Datenbandes,

b) Programm in den Speicher laden.



6.1. Schreiben eines Bandfiles (PRINT #, CMD oder SAVE)

*Mit "Taste" ist eine der drei Tasten PLAY, FFWD, REW gemeint. Es wird vom Computer nicht unterschieden, welche dieser Tasten gedrückt ist.

OS . . . Betriebssystem (operating system)

B . . . . BASIC

Beachten Sie, daß bei direkter Eröffnung des Files über die Tastatur die Mitteilungen PRESS PLAY . . ., SEARCHING FOR . . . und FOUND . . . erscheinen. Wurde LOAD befohlen, dann werden die BASIC-Variablen des geladenen Programms initialisiert.

Wird das File unter Programmkontrolle eröffnet und ist die PLAY-Taste gedrückt, so erscheinen auf dem Bildschirm keine Mitteilungen, um zu vermeiden, daß die vom laufenden Programm erzeugte Anzeige gestört wird.

Eine Initialisation der BASIC-Variablen findet nicht statt.

7. Input: Allgemeines

Das Wort Input bedeutet hier die Eingabe von Daten in den Computer.

7.1. INPUT

INPUT # ist der Befehl, um die Datenübertragung von I/0-Geräten zum Betriebssystem zu veranlassen. Format:

INPUT # logische Filenummer, Variablenliste

In der Variablenliste stehen, jeweils durch Kommata getrennt, Zahl- oder Stringvariablen, die zeichenseriell vom betreffenden File an den Computer übertragen werden.

Da die Regeln des BASIC-Interpreters auch für die Inputanweisungen gelten, werden alle CR, Kommas, Feldbegrenzer, Nullzeichen, Spaces (außer innerhalb von Zeichenketten) sowie andere Kontrollzeichen automatisch gelöscht.

Es ist nicht immer möglich, numerische und alphanumerische Daten auf dem I/0-Gerät zu mischen. Ist ein numerisches Feld spezifiziert, so werden nur numerische Daten in der von BASIC vorgesehenen Standardform akzeptiert. Ansonsten erscheint die Mitteilung ?BAD DATA ERROR.

Sollten Unklarheiten über einlaufende Daten auftauchen, so kann man diese in Strings übernehmen und dann die verschiedenen Stringmanipulationen benutzen, um die Daten in entsprechende Variablen umzuwandeln.

7.1.1. Beispiel für INPUT

Falls X eine Reihe von 50 Zahlen darstellt, die auf einem Bandfile mit der Bezeichnung Vektor gespeichert sind, und wir annehmen, daß die PLAY-Taste des Recorders #1 gerade gedrückt wurde, dann wird folgendes Programm die 50 Zahlen einzeln einlesen und sie auf dem Bildschirm erscheinen lassen:

10 OPEN 1,1,0, "VEKTOR"	Logisches File #1 eröffnen File dem Recorder #1 zuordnen Band zum Lesen eröffnen Nach physikalischem File mit der Bezeichnung "VEKTOR" suchen
20 FOR K = 1 to 50	50 mal
30 INPUT #1,X	Zahl einlesen vom Recorder #1
40 PRINT X	Zahl auf den Bildschirm schreiben
50 NEXT K	
60 CLOSE 1	Wenn die 50 Zahlen eingelesen sind, logisches File #1 schließen.

7.2 GET

Nicht alle Geräte übermitteln Daten in einer Form, die BASIC direkt aufnehmen kann. Es gibt eine Reihe binärer Daten und Codes, die von BASIC ignoriert werden. Es gibt eine ganze Reihe von IEEE-Geräten – wenn auch nicht alle – die für BASIC akzeptable Codes ausgeben.

Zusätzlich möchte der Programmierer manchmal unmittelbaren Zugang zu Symbolen haben, die auf dem Bus transportiert werden. GET # holt ein einzelnes Zeichen vom IEEE-Bus (oder vom Recorder) und legt es in der Variablen ab:

GET # logische Filenummer, Variable

7.3. Input vom Band

Das Lesen eines Bandfiles wird vom Betriebssystem gesteuert. Jedesmal, wenn BASIC von einem logischen File, das zum Lesen von Recorder #1 oder #2 eröffnet wurde, INPUT# oder GET# anordnet, wird ein Unterprogramm abgerufen, das eine Bandeingabe initiiert.

Das jeweilige Symbol wird von BASIC aus dem entsprechenden Puffer abgerufen. Wenn der Pufferspeicher leer ist, unterbricht das Betriebssystem das laufende Programm, liest den nächsten Datenblock vom Band in den Zwischenspeicher ein und überträgt das nächste Symbol an BASIC. Tritt ein Lesefehler auf, wird die Statusvariable (ST) entsprechend geändert.

Wird im Zwischenspeicher das Fileende (EOF) angezeigt, dann wird die Statusvariable dahingehend verändert und automatisch CR ausgegeben, bis der Befehl beendet ist.

Am Ende eines Befehls ruft BASIC ein anderes Programm auf, welches wieder Eingabe über die Tastatur erlaubt und der Fileende Operation anzeigt, daß der Befehl ausgeführt wurde.

7.4. IEEE-488-Input

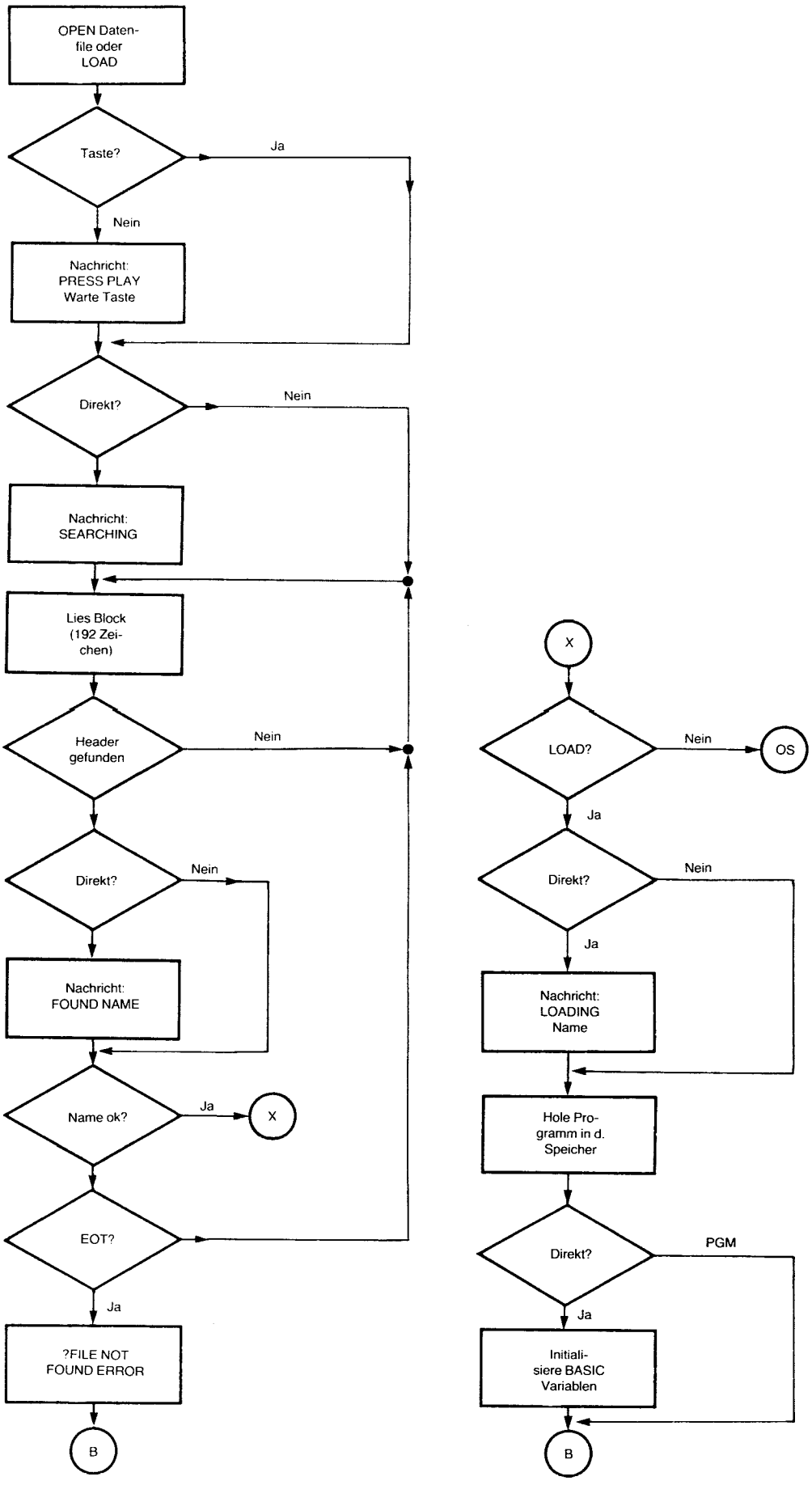
Alle INPUT# - oder GET# -Befehle durchlaufen die gleiche Sequenz. Trifft das Gerät zum ersten Mal auf einen solchen Befehl, so wird eine Routine aufgerufen, die eine Talk-Attention-Folge an das Gerät sendet sowie die Sekundäradresse, die in der OPEN-Folge für dieses logische File spezifiziert ist. Am Ende der Attention-Folge begibt sich der Computer in den Listener-Zustand und wartet auf ein DAV Signal (data valid), welches anzeigt, daß ein einzelnes Symbol empfangen worden ist. Wenn DAV innerhalb von 65 Millisekunden eingegangen ist, wird dieses Symbol an BASIC weitergeleitet und/oder an das andere Programm, welches die IEEE-488 Routine aufgerufen hat. Bei jedem Aufruf des IEEE-488-Programms geht es durch die gleiche Folge, in der es versucht, innerhalb von 65 ms (time out) ein einzelnes Symbol zu erhalten. Antwortet der Bus nicht innerhalb von 65 Millisekunden, so beendet das IEEE-488-Programm automatisch die Folge; es zeigt einen Lesefehler über die Statusvariable an, was zugleich das Ende dieser Routine bedeutet.

Empfängt der Computer im Verlauf des Zeichen-Einlesens ein EOI-Signal (End Or Identify), so zeigt er in der Statusvariablen das Ende der Information an und erzwingt CR's, bis der Befehl, unter dem er gerade arbeitet, beendet wird. Am Ende des Befehls ruft BASIC ein Beendigungsprogramm ab. Damit wird das Gerät erneut für die Tastatur initialisiert, und der Computer sendet "Untalk" an den IEEE-488-Bus, wodurch dieser für den nächsten Befehl empfangsbereit wird.

7.5. Grenzen des Input-Puffers

Obgleich das Betriebssystem die Daten zur Bearbeitung einzeln überträgt, sammelt BASIC diese Zeichen in einem 80-spaltigen Inputpuffer. Am Ende dieses Zwischenspeichers muß ein Wagenrücklauf stehen. Werden mehr als 80 Symbole mit INPUT# eingelesen, so funktioniert das Betriebssystem nicht mehr, da die Variablen des Betriebssystems überschrieben werden. Der Computer wird wieder funktionstüchtig durch Aus- und Wiedereinschlagen.

Diese Einschränkung sollte bei Benutzung des Bands und der Floppy-Disk nicht vergessen werden.



6.2. Lesen eines Bandfiles
(siehe auch Abb. 6.1.)

Dies bedeutet, daß so oft CR = CHR$(13) auf Band, Platte oder andere Peripheriegeräte geschrieben werden muß, daß nicht mehr als 80 Symbole hintereinander stehen, ohne von Wagenrückläufen unterbrochen zu sein.

Sendet ein I/0-Gerät mehr als 80 Symbole, kann der GET-Befehl zum Einlesen verwendet und mit eigener Stringverarbeitung die Zwischenspeicher-Begrenzung umgangen werden.

8. Datenausgabe

8.1. PRINT#

Auf den Befehl PRINT# muß die logische Filenummer folgen, sowie ein Komma zur Trennung von den folgenden Daten.

PRINT# logische Filenummer, Daten

Daten werden zeichenseriell an dasjenige Gerät übertragen, das mit dem logischen File, wie es in der OPEN Anweisung spezifiziert ist, übereinstimmt. Viele File-Trennsymbole, wie z.B. Kommas, werden von BASIC automatisch beseitigt; obgleich dies keinen großen Effekt beim Schreiben hat, sollte beim Zurücklesen vom Band oder einem anderen I/0-Gerät beachtet werden, daß File-Trennsymbole forciert werden müssen. Dies kann durch die Eingabe von CHR$(44) – Komma – zwischen Feldern erreicht werden, oder indem in jeder PRINT# Anweisung nur ein Feld vorkommt, wodurch Wagenrückläufe erzwungen werden.

Beispiel:

Anstatt PRINT# LF,A;B$;C$

– was als ABC ohne Trennungszeichen gesendet wird, sollte geschrieben werden:

PRINT# LF,A;CHR$(44);B$;CHR$(44);C$

oder: PRINT# LF, A", "; B$;",", C$

was als A,B$,C$,CR

erscheint (CR = Wagenrücklauf), oder:

```
PRINT# LF,A
PRINT# LF,B$
PRINT# LF,C$
```

was die Ausgabe von

A CR B$ CR C$ CR

bewirkt.

Da BASIC die Ausgabe an jedes Gerät immer so aufbaut, als wenn es an den Bildschirm ausgeben würde, bewirkt PRINT# LF,A,B immer mehrere Leerzeichen (Skips) zwischen den Werten von A und B, während A;B einen Output ohne Zwischenräume bewirkt.

Eine Ausnahme bildet die Ausgabe auf Band, wo das erste „Skip" – CHR$(29) – unterdrückt wird.

Anmerkung: Obgleich INPUT # und PRINT# fast auf die gleiche Weise funktionieren wie INPUT und PRINT, kann die Abkürzung ? zwar für PRINT aber nicht in PRINT # verwendet werden. PRINT wird im Programmspeicher mit 153 codiert, PRINT# mit 152. ?# wird wie PRINT# gelistet, führt jedoch zu ?SYNTAX ERROR bei dem Versuch, es durchzuführen.

8.1.1. Beispiele für PRINT# Anweisung

Dieses Programm druckt die Zahlenfolge 1,2,3, . . . 50; und zwar druckt sie der cbm Drucker alle einzeln nacheinander.

```
10 OPEN 5,4,0        Eröffnung des logischen Files #5
                     Zuordnung des logischen Files #5 zum Gerät #4 (Drucker) in der normalen Druckart, die der Sekundäradresse "0" entspricht.
20 FOR K=1 to 50
30 PRINT #5,K        Schreibt 50 Zahlen in den Drucker (Puffer)
40 NEXT K
50 CLOSE 5           Schließen des logischen Files #5
```

Soll die obige Zahlenfolge auf eine Kassette im Recorder #2 geschrieben werden, so muß lediglich der OPEN-Befehl modifiziert werden, wenn man die gleichen Filenummern beibehält:

```
10 OPEN 5,2,1      Eröffnung des logischen Files #5
                   Zuordnung des logischen Files #5 zum Gerät #2 (externer Recorder),
                   ohne Schreiben des "end of tape" (Bandendezeichen), also nur mit EOF
                   (end of file).
20 FOR K=1 to 50
30 PRINT #5,K      Schreibt 50 Zahlen über den Puffer für Recorder #2 auf das Band.
40 NEXT K
50 CLOSE 5         Schließen des logischen Files #5
```

Im obigen Beispiel werden die Daten in einem Zwischenspeicher für 192 Zeichen angesammelt, und zwar zeichenseriell. Wird die Zwischenspeicherkapazität überschritten, so wird das Einlesen der Daten unterbrochen, das Band gestartet und der Inhalt des Zwischenspeichers auf Band geschrieben. Der Zwischenspeicher wird dann wieder für 192 Zeichen initialisiert, dann kann das Programm weiterarbeiten.

8.2. Output auf IEEE-488

Der PRINT #-Befehl veranlaßt das BASIC-Programm, eine Ausgabe-Subroutine abzurufen, die ein IEEE-488-Gerät für Output initialisiert. Der erste Schritt hierbei besteht darin, daß der Computer seine normale Ausgabe vom Bildschirm wegnimmt und dem entsprechenden IEEE-Gerät zuordnet. Dann wird an das Gerät über den Bus "Listen" gesandt und die im OPEN angegebene Sekundäradresse. Das BASIC-Programm übergibt dann ein Zeichen nach dem anderen an ein anderes Unterprogramm.

Dieses letzte Programm überträgt die Daten via Bus an jedes Gerät, das als "Listener" angesprochen ist. Der Computer ist der einzige "Talker" am Bus, wie ein Redner vor einem oder mehreren Zuhörern.

Wenn BASIC mit PRINT# aufgehört hat, wird ein anderes Unterprogramm des Betriebssystems aufgerufen. Der Computer sendet einen "unlisten"-Befehl an den gesamten Bus und weist erneut die Primäradresse dem Bildschirm zu. Dadurch wird der Bus für die nächste Operation frei.

Diese "unlisten"-Folge sendet ebenfalls ein EOI-Signal an den Bus, zusammen mit dem letzten Zeichen aus dem BASIC. Um dies zu ermöglichen, wird jedes Zeichen in einem Zwischenspeicher gespeichert, bevor eine Übermittlung durch die IEEE-Programme stattfindet und das vorherige Zeichen gesandt wird.

8.3. CMD-Befehl

Normalerweise befaßt sich jeder Druckbefehl nur mit einem logischen Gerät, und am Ende wird der gesamte Bus (mit "unlisten") abgeschaltet. In einigen Fällen ist es ratsam, mehr als ein Gerät am Bus zu haben; um dies zu vereinfachen, gibt es den Sonderbefehl CMD. CMD ist fast identisch mit PRINT#, außer daß am Ende der Datenübermittlung das "unlisten" Programm nicht abgerufen wird und dadurch das Gerät als Hörer am Bus bleibt. Das Betriebssystem betrachtet das letzte Gerät weiterhin als von CMD befohlenes Primär-Ausgabegerät für BASIC. PRINT- oder LIST-Befehle werden dann, statt zum Bildschirm, zu diesem Primärgerät geleitet. Dies bedeutet, daß der CMD-Befehl an den Drucker, gefolgt von LIST, die Ausgabe des Progamms auf dem Drucker

bewirkt, anstatt auf den Bildschirm. Da jedoch weder CMD oder LIST Abschluß-Bus-Operationen für das Gerät bewirken können, ist ein PRINT# erforderlich, um einen CMD-Befehl abzuschließen.

8.3.1. Beispiel eines CMD-Befehls

Listen eines Programms:

OPEN 3,4 4 ist die Gerätenummer des Druckers

CMD 3 jeglicher Output geht an Gerät 3

LIST Listet das Programm auf dem Drucker

gleichzeitig drucken und auf Diskette schreiben:

*CMD 3 wobei das logische File #3 dem Drucker zugeordnet sei,

PRINT#15,A,B,C mit 15 als logische Filenummer der Floppy-Disk (vorher eröffnet)

führt dazu, daß A, B und C auf der Diskette gespeichert werden, aber auch auf dem Drucker erscheinen.

Überwacht man ein Eingabegerät:

**CMD 3 Output an Drucker

INPUT#15,A,B,C von Floppy lesen

führt dazu, daß Daten, die von der Diskette kommen, an A, B und C weitergeleitet, aber auch während der Weiterleitung gedruckt werden.

9. Schlußfiles

Alle logischen Files, die während eines Programms eröffnet wurden, sollten auch, wenn sie nicht mehr gebraucht werden, wieder geschlossen werden. Band und Platten-Files müssen sogar vor Programmende geschlossen werden. Folgendes sollte beachtet werden:

a) Überschreitet die Gesamtzahl der eröffneten Files zu einem beliebigen Zeitpunkt 10, so erscheint die Fehlermeldung ?TOO MANY FILES ERROR.

b) Wird ein logisches File nicht geschlossen, das einem Recorder zugeordnet ist, so erscheint am Ende des physikalischen Bandfiles kein End-Of-File-Zeichen.

Wird dieses Band dann in den Speicher eingeladen, so hat der Rechner keine Möglichkeit zu erfahren, wann das File beendet ist, und falls unerwünschte oder falsche Daten aus vorherigen Aufnahmen auf dem Band stehen, werden diese ebenfalls abgespeichert.

9.1. Beispiel für CLOSE

Um ein File zu schließen, genügt folgende einfache Anweisung:

CLOSE logisches File

Falls logisches File Nummer 5 geschlossen werden soll, so heißt es:

CLOSE 5

Files, die nie geöffnet wurden, können geschlossen werden, ohne daß hierdurch eine Fehlermeldung bewirkt wird.

9.2. Schließen von Bandfiles

Wenn ein File auf einem Band eröffnet wurde, so werden bei CLOSE zwei Operationen durchgeführt:

1. das nächste Zeichen wird als EOF-Zeichen markiert und
2. der Recorderpuffer wird geleert (Ausgabe aller Zeichen auf Band).

Wurde im OPEN die Möglichkeit "end of tape" gewählt, so wird zusätzlich ein EOT File-Header ausgegeben.

* Muß jedesmal angegeben werden, da PRINT# den Bus abschaltet.

** Muß nicht jedesmal angegeben werden, zusätzlicher Code kann zwischen den Anweisungen eingefügt werden.

9.3. Schließen einzelner IEEE-488 Gerätefiles

Für IEEE-488 Geräte, die mit Filenamen eröffnet wurden, wird eine besondere Hörer-Befehls-Folge übermittelt. Sie besteht aus der speziellen Sekundäradresse $EØ exclusiv oder mit der Sekundäradresse aus dem OPEN. Dadurch können Peripheriegeräte wie Plattenfiles geschlossen werden.

10. Fehlersuche: Allgemein

Zum Grundkonzept des Betriebssystems gehört, daß der Benutzer die Möglichkeit des freien Aufbaus hat; er kann auf Bändern, Platten und Druckern auf die für ihn günstigste Art lesen und schreiben.

Da die Ein- und Ausgabe einen völlig freien Aufbau hat, ist es wichtig, daß das Betriebssystem dem Benutzer eine Informationsmöglichkeit gibt, wenn Übertragungsfehler oder die Anzeige "Datenende" auftreten. Abschnitte 10.1. bis 10.4. befassen sich mit Methoden der Fehlersuche.

10.1. Die Statusvariable ST

Um die Fehlersuche bei Input/Output zu erleichtern, verwendet der Computer die Konzeption des "Statuswortes". Dabei wird durch jede I/O-Operation ein bestimmtes Byte im RAM manipuliert. Dieses Byte kann vom Programmierer jederzeit abgefragt werden (PRINT ST). Jedes Bit im Statuswort hat eine allgemeine Bedeutung für alle Operationen und eine besondere für das spezielle I/O-Gerät.

Tabelle 10.1. zeigt den Zusammenhang zwischen Wert der Statusvariablen und Fehlerarten für die Recorder und die IEEE Lese- und Schreiboperationen.

ST Bit	ST numerischer Wert	Band Lesen	IEEE R/W	Band VERIFY und LOAD
Ø	1		Time out beim Schreiben	
1	2		Time out beim Lesen	
2	4	zu kurzer Block		kurzer Block
3	8	zu langer Block		langer Block
4	16			jede Nicht-übereinstimmung
5	32	Prüfsum-menfehler		Prüfsummen-fehler
6	64	Fileende	EOI (End or Identify)	
7	-128	Bandende	Gerät nicht vorhanden	Bandende

Tabelle 10.1. Fehler und ST

10.2. IEEE Geräte-Fehler

Grundsätzlich gibt es drei Fehler, die während einer IEEE-488-Übertragung passieren können.

Als erstes ist es möglich, daß der Bus nicht auf "attention" reagiert. Wenn dies geschieht, setzt eine Subroutine "device not present" – (Bit 7 der Statusvariablen) und der Computer beendet das laufende Programm mit einem ?DEVICE NOT PRESENT ERROR. Wenn der Bus auf attention richtig reagiert, der Computer aber das erste Zeichen an den Bus ausgibt und das technische Gerät nicht funktionsfähig ist (angezeigt durch NRFD low oder NDAC low), so zeigt der Rechner ebenfalls "Gerät nicht vorhanden" an.

Der zweite Fehler tritt während der Datenübertragung an das Gerät auf. Der Bus reagiert nicht in den entsprechenden Zeiten und/oder, falls er aufhört zu reagieren, indem er NRFD und NDAC Logisch Eins setzt, wird ein Schreibfehlerhinweis in Bit Ø gegeben.

Der dritte Fehler tritt ein, wenn das Peripherigerät während des Lesens am IEEE-488 kein DAV innerhalb 65 Millisekunden gesendet hat; Bit 1 der Statusvariablen wird dann gesetzt. Wenn EOI (end or identify) eintritt, wird Bit 6 im Statuswort gesetzt und laufend CR erzwungen.

10.3. Kassettenfehler

Der Computer prüft die Daten lediglich beim Einlesen von Kassette. Er kann die folgenden Fehler finden:

1) Kurzer Datenblock: (4) Beim Ablesen eines Blocks vom Band wurde ein Trennungston erkannt, bevor die erwartete Anzahl von Bytes vom Block eingelesen war. Mögliche Ursache: Der Versuch, ein Programmfile als Datenfile zu lesen.
2) Langer Datenblock: (8) Beim Ablesen eines Blockes vom Band wurde Trennungston angetroffen, nachdem mehr als die erwartete Anzahl Bytes von diesem Block abgelesen war. Mögliche Ursache: Einlesen eines Programmfiles als Datenfile.
3) Nicht zu beseitigender Lesefehler (16). Ursache: mehr als 31 Fehler auf dem ersten Datenblock, oder ein nicht zu verbessernder Fehler, weil er an der gleichen Stelle in beiden Blocks auftrat.
4) Kontrollsummenfehler (32). Nach einem LOAD oder Dateneinlesen wird eine Kontrollsumme berechnet (über die Bytes im RAM) und mit einem vom Eingabegerät erhaltenen Byte verglichen. Wenn sie nicht übereinstimmen, wird Byte 32 gesetzt.
5) File-Ende (64). Dieses Byte wird gesetzt, wenn man beim Daten-Ende-File bei einer Bandwiedergabe angelangt ist.
6) Band-Ende (-128). Ein EOT-Zeichen wurde eingelesen.

10.4. Beispiele für ST-Benutzung

Es gibt keinen Status, mit dem der Computer das Schreiben auf Bänder überwacht, noch für Fehler, die bei der Ausgabe auf dem Bildschirm bzw. beim Lesen vom Bildschirm auftreten. Es gibt einen Fehler, der beim Schreiben von Daten auf den IEEE-488-Bus, und es gibt eine Vielzahl von Fehlern, die beim Eingeben vom IEEE-Gerät entdeckt werden.

Die normale Programmierungstechnik sieht vor, daß auf INPUT# oder GET# ein Test oder die Speicherung des Status folgt. Da dies nur ein einziges Byte ist und der Status sich bei jedem neuen I/O-Befehl verändert, hat der Status einen sehr veränderlichen Wert.

```
100 INPUT# 2,A
110 INPUT# 5,B
120 IF ST = Ø THEN 200
```

Dieser Code überprüft lediglich das Ergebnis der Datenübertragung vom logischen File 5. Die Ergebnisse des Lesens des logischen Files 2 sind für immer verloren.
Ähnlich:

```
100 INPUT# 2,A
110 PRINT A
120 IF ST = Ø THEN 200
```

In diesem Fall gibt ST den Druckstatus wieder und nicht die Ergebnisse des Lesens vom File 2.

Eine Möglichkeit, ST zu verwenden, ist folgende:

```
100 INPUT# 2, A,B,C
110 IF ST = Ø THEN 200          Vorgang normal
120 IF ST = 64 THEN 300         keine Fehler bis Datenende
130 IF ST = 2 THEN 400          Zeit vorbei ohne Fehler
```

Jeder Fehler kann jetzt folgendermaßen behandelt werden:

```
140 IF ST AND MASKE THEN. . .   MASKE steht für das zu testende Bit (1,2,4,8,16,32,
                                64 oder -128)
```

11. POLLING (Serielle Abfrage mehrerer Geräte)

Eine Technik, IEEE-488-Geräte abzufragen, die sehr langsam reagieren, besteht darin, INPUT # zu verwenden.
Wenn der Status TIME OUT anzeigt, so verarbeiten Sie andere Routinen. Oder Sie programmieren eine Schleife auf INPUT#, bis kein Fehler auftritt. Wenn es keine Fehler gibt, sind die richtigen Daten endlich eingelesen worden, und die Daten können verarbeitet werden.

Wenn man diese Sampling-Technik anwendet, kann man eine ganze Reihe von langsamen Geräten bedienen, zusammen mit einem Vordergrundprogramm. Man kann die eingebaute Uhr sowie die INPUT # -TIMEOUT-Schleife benutzen, um gelegentlich die Geräte abzufragen.

12. Ersatzparameter

Folgende Parameter werden vom Computer ausgegeben, wenn sie nicht vom Benutzer definiert wurden:

Parameter	Ersatzparameter	Operation
Geräte#	D=1	Kassette# 1 wird angesprochen.
Sekundäradresse	SA=Ø	Bei Band-Files: OPEN zum Lesen. Auf IEEE-Geräten: eine Sekundäradresse wird nicht gesandt.

Tabelle 12.1. Ersatzparameter

Anweisung	Gleichwertige (vom Computer eingesetzte) Parameterwerte	Operation
OPEN 1	OPEN 1,1,Ø	Eröffnung des logischen Files # 1 (Recorder# 1) zum Einlesen. Kein Filename.
OPEN 1,2	OPEN 1,2,Ø	Eröffnung des logischen Files# 1 (Recorder# 2) zum Einlesen. Kein Filename.
OPEN 1,2,1	OPEN 1,2,1	Eröffnung des logischen Files# 1 (Recorder# 2) zum Schreiben. Kein Filename.
OPEN 1,2,1,"DAT"	OPEN 1,2,1,"DAT"	Eröffnung des logischen Files # 1 (Recorder # 2) zum Schreiben. Filename "DAT"

Tabelle 12.2. Beispiele für Ersatzparameter

DER IEEE-488-BUS

1. Einführung

Der Bus besteht im wesentlichen aus 16 Signalleitungen, die von der Funktion her in drei Gruppen aufgeteilt sind:

a) Der Daten-Bus,
b) der Kontroll-Bus,
c) der Management-Bus.

Weiterhin kann der IEEE-Bus drei Arten von Geräten versorgen:

a) Talker (Sprecher). Zu jedem Zeitpunkt kann nur ein Gerät Daten an den Datenbus übermitteln.
b) Listener (Zuhörer). Es können mehrere Geräte zur gleichen Zeit vom Bus empfangen.
c) Controller (Kontrollgerät). Nur der Computer darf den IEEE-Bus kontrollieren. Es darf nur ein Controller am Bus angeschlossen sein.

Die Funktionen und Operationsweisen der Daten-, Kontroll- und Management-Busse werden in den Abschnitten 1.2. bis 1.4. besprochen.

1.1. Bus-Geräte-Verbindung

Die Verbindungen zu dem aus 12 Positionen und 24 Kontakten bestehenden Platinenstecker gehen von der Hauptplatine des Computers aus (siehe Tabelle 1.1.). Weitere Einzelheiten im Kapitel "Lage der Schnittstellen und Bezeichnung der Kontakte".

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 angeschlossen sein.

1.2. Datenbus

Dieser Datenbus umfaßt 8 Zweirichtungsleitungen, (aktiv low) für die Datensignale DI01 bis DI08. Das am langsamsten arbeitende Gerät kontrolliert zu jedem Zeitpunkt die Übertragungsgeschwindigkeit. Die Übertragungsart ist byteseriell und bitparallel.

Peripherieadressen und Kontrollinformationen werden ebenfalls über den Datenbus übertragen. Sie werden von Daten dadurch unterschieden, daß während der Übertragung ATN=wahr gilt.

Das höchstwertige Bit (MBS) liegt auf Leitung DI08.

1.2.1. Daten-Übertragungs-Methoden

Bei der Datenübermittlung an die Geräte können alle möglichen Bitmuster als Daten verwendet werden.

1.3. Kontrollbus (Transfer-Bus)

Diese drei Leitungen kontrollieren die Übertragung von Daten über den Datenbus. Diese Signale werden im Handshake-Verfahren (siehe Abschnitt 1.3.1.) verwendet.

Zum Transfer-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.

Eine Beschreibung der Signale finden Sie in Tabelle 2.1.

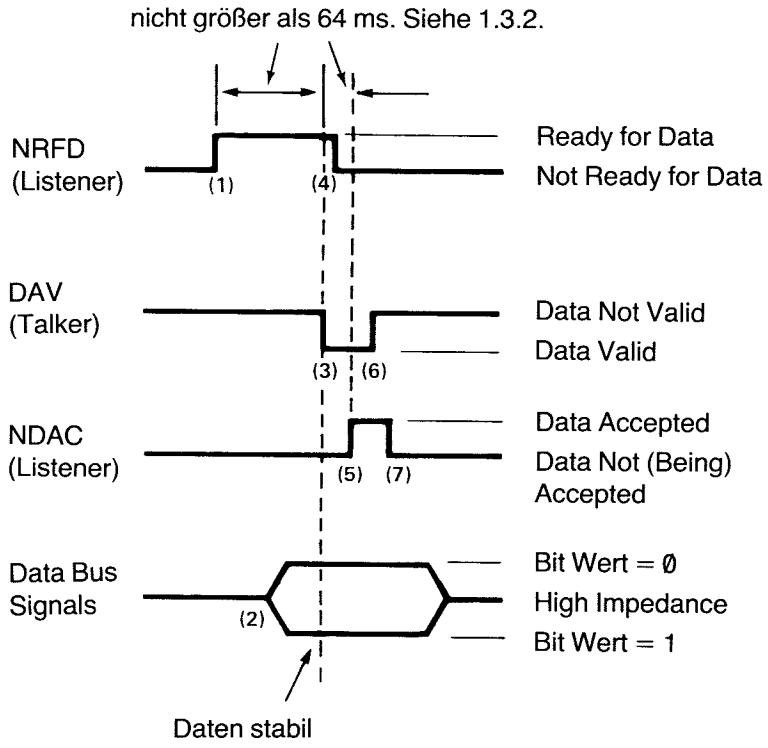
1.3.1. Das Handshake-Verfahren

Wenn ein Talker ein Datenbit an einen oder mehrere Listener übermittelt, wird dieses Kontrollverfahren angewandt, um eine erfolgreiche Operation sicherzustellen.

Das Handshake soll im wesentlichen sicherstellen, daß:

a) alle Hörer zur Datenaufnahme bereit sind
b) gültige Daten am Datenbus anliegen
c) alle gültigen Daten von allen Hörern angenommen wurden.

Die Geschwindigkeit der Datenübermittlung wird von dem langsamsten aktiven Gerät am Bus bestimmt. Dies ermöglicht die gleichzeitige Verwendung von Geräten, die Daten bei unterschiedlichen Geschwindigkeiten bearbeiten.



1.1. Transfer-Bus Handshake

Das Diagramm (Zeichnung 1.2.) demonstriert die Abläufe, die sich während der Übertragung eines Datenbytes vom Talker zu den Listenern ergeben.

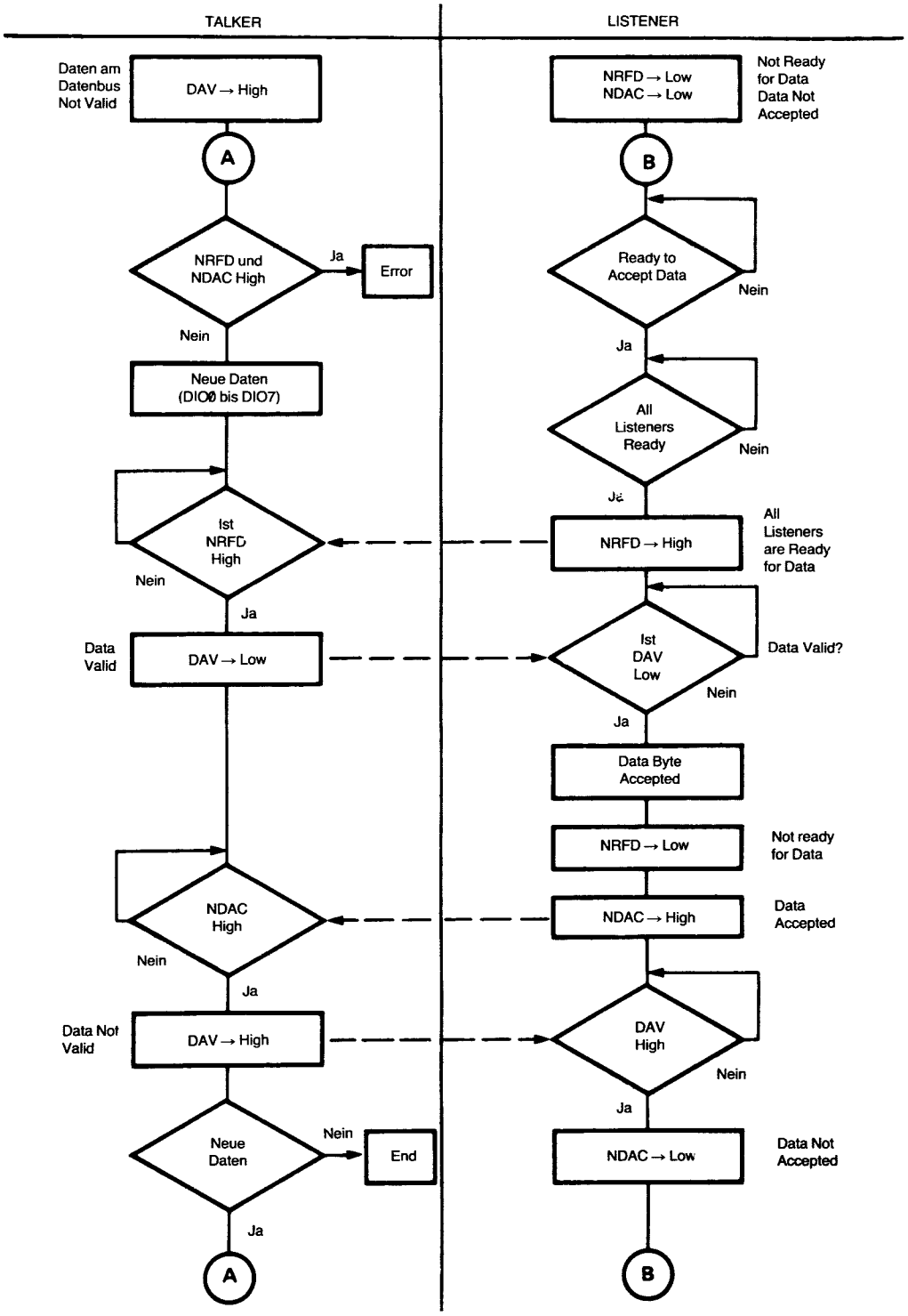
Zeichnung 1.1. demonstriert das relative Zeitverhalten der Transferbus-Signale während eines typischen Handshakes; die in Klammern gesetzten Zahlen in der untenstehenden Beschreibung beziehen sich auf die Zeichnung:

1) NRFD wird HIGH (falsch) und zeigt damit an, daß alle Hörer für das nächste Datenbyte aufnahmebereit sind.
2) Der Talker setzt das nächste Datenbyte auf den Datenbus und wartet, bis die Daten eingeschwungen sind. Dies kann vor, nach oder während (1) passieren.
3) Der Talker testet NRFD; stellt sich heraus, daß es HIGH ist, so setzt der Talker DAV=LOW, um die Hörer davon in Kenntnis zu setzen, daß die Daten am Bus jetzt gültig sind.
4) Sobald irgendein Hörer entdeckt, daß DAV=LOW gilt, setzt dieser Hörer auch NRFD=LOW.

Die Daten werden jetzt von allen Hörern jeweils zu ihrer Geschwindigkeitsrate angenommen, wobei jeder von ihnen NDAC auslöst, wenn er Daten annimmt.

5) NDAC=falsch, wenn der langsamste Listener die Daten angenommen hat.
6) Der Talker setzt DAV hoch (falsch) und zeigt damit an, daß die Daten auf dem Bus jetzt ungültig sind.
7) Die Hörer merken, daß DAV=HIGH, setzen NDAC=LOW (richtig) und beenden damit das Handshake. Sobald jeder Listener die Daten verarbeitet hat, setzt er NRFD frei.
Hiermit ist das erste Datenbyte übertragen. Die Folge wiederholt sich; sie beginnt mit (a), bis alle Daten übertragen sind.

1.2. Datenübertragung vom Talker zu den Listenern



6.2. Lesen eines Bandfiles
(siehe auch Abb. 6.1.)

1.2.3. Zeitliche Beschränkungen

Um keine 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). Siehe Tabelle 3.1.

1.4. 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:

a) ATN	Attention	Ordnet Geräten die Listener- und Talkerfunktionen zu.
b) EOI	End or identify	Zeigt an, daß das letzte Datenbyte übertragen worden ist. (Ende oder Kennzeichnung)
c) IFC	Interface clear	Initialisiert den Datenbus. Talker- und Hörerzuteilung wird aufgehoben. Das gleiche Signal wie Reset im Computer.
d) SRQ	Service request	Gerät teilt Controller mit, daß Bedienung nötig ist. Kann nicht im BASIC jedoch im Rechner durchgeführt werden (Bedienungsaufruf).
e) REN	Remote enable	Dieses Signal liegt bei Ihrem Computer permanent auf LOW.

2. Signale und Definitionen

Allen 16 Übertragungsleitungen des IEEE-488-Bus ist ein bestimmtes Signal zugeordnet. Tabelle 2.1. zeigt die Busgruppe, Namen, Abkürzung und funktionale Beschreibung jedes einzelnen dieser Signale.

2.1. Definition der Logiksignale

„Wahr", also „logisch Eins" wird durch eine Spannung nahe Null realisiert. Aufgrund der Buskonstruktion mit gemeinsamen Kollektorausgängen kann so jedes Gerät den „Wahr"-Zustand erzeugen und erhalten.

Tabelle 2.1. IEEE-BUS-SIGNALE

Signal Busgruppe	Abkürzung	Name	Funktionale Beschreibung
Manager	ATN	Attention	Der Controller setzt dieses Signal niedrig, während er Befehle an den Datenbus gibt. Ist ATN = Low, so befinden sich nur Periphere Adressen und Kontrollmitteilungen am Datenbus. Ist ATN = HIGH, so können nur vorher zugeordnete Geräte Daten übermitteln.
Transfer	DAV	Date Valid (Daten gültig)	Ist DAV = LOW, dann sind die Daten am Datenbus gültig.
Manager	EOI	End of Identify (Ende oder Kennzeichen)	Sobald das letzte Datenbyte übertragen ist, kann der Talker EOI senken. Der Computer setzt EOI immer niedrig, 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 (richtig) 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 (Bedienungs-aufruf)	Wird nicht von BASIC bedient, kann aber realisiert werden (über Software).
Manager	REN	Remote enable	REN wird vom Bus-Controller niedrig gehalten. Ihr Computer hält REN immer auf LOW.
Daten	DIO1 bis DIO8	Daten-Eingabe Ausgabe-Leitungen 1-8	Diese Signale stellen die Informationsbits am Datenbus dar. Liegt ein DIO-Signal auf LOW, so bedeutet dies, das betreffende Bit ist Null.
Allgemein	GND	Masse	Erdungsverbindungen: Es gibt sechs Masseleitungenkontroll- und Managementbytes, eines für den Datenbus und eine für Chasis.

3. Die Statusvariable ST

ST ist eine BASIC-Variable, die zur Kontrolle des Ergebnisses von Input/Output-Operationen verwendet werden kann. ST kann Werte zwischen – 128 und 127 annehmen. Tabelle 1.4. zeigt die Statusmeldungen, die sich auf den IEEE-Bus beziehen.

Tabelle 3.1. ST Status Code für IEEE-488-Bus

ST	Fehler	Erklärung
1	Zeit bei Listener überschritten	Das IEEE-Gerät hat nicht innerhalb von 65 Millisekunden geantwortet.
2	Zeit bei Talker überschritten	Das IEEE-Gerät hat kein aktives "Daten gültig" Signal (DAV low) innerhalb von 65 Millisekunden gegeben.
64	Ende oder Kennzeichnung	EOI ist LOW (richtig) mit dem letzten Datenbyte an den IEEE-Bus übermittelt worden. Beachten Sie, daß nicht alle Peripheriegeräte ein EOI-Signal hervorbringen. Informieren Sie sich in dem entsprechenden Handbuch.
–128	Geräte nicht vorhanden	Gerät reagiert nicht auf Adressierung. Dies verursacht eine Fehler-Mitteilung, und das Betriebssystem schaltet den Computer auf die BASIC-Befehls-Ebene zurück.

4. IEEE-488 Registeradressen

Tabelle 4.1. 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.

Tabelle 4.1. IEEE-Hardware-Adressen und Signal-Informationen

Sedezimale Adresse	Dezimale Adresse	Bits	IEEE	Betriebsart
E82Ø	59424	Ø-7	DIO1-8	Eingang
E822	59626	Ø-7	DIO1-8	Ausgang
E821	59425	3	NDAC	Ausgang
E823	59427	3	DAV	Ausgang
		7	SRQ	
E81Ø	594Ø8	6	EOI	Eingang
E84Ø	59456	Ø	NDAC	Eingang
		1	NRFD	Ausgang
		2	ATN	Ausgang
		6	NRFD	Eingang
		7	DAV	Eingang

Code assignments for "Command Mode" of operation.

(SENT AND RECEIVED WITH ATN TRUE)

b7 b6 b5 / Bits b4 b3 b2 b1	COLUMN→ ROW↓	0 0 0	1 MSG	0 0 1	MSG	0 1 0	MSG	0 1 1	MSG	1 0 0	MSG	1 0 1	MSG	1 1 0	MSG	1 1 1	MSG
		0		1		2		3		4		5		6		7	
0 0 0 0	0	NUL		DLE		SP	MLA ASSIGNED TO DEVICE	0	MLA ASSIGNED TO DEVICE		MTA ASSIGNED TO DEVICE	P	MTA ASSIGNED TO DEVICE		MEANING DEFINED BY PCG CODE	p	MEANING DEFINED BY PCG CODE
0 0 0 1	1	SOH	GTL	DC1	LLO	!		1		A		Q		a		q	
0 0 1 0	2	STX		DC2		"		2		B		R		b			
0 0 1 1	3	ETX		DC 3		#		3		C		S		c		s	
0 1 0 0	4	EOT	SDC	DC4	DCL	$		4		D		T		d		t	
0 1 0 1	5	ENQ	PPC3	NAK	PPU	%		5		E		U		e		u	
0 1 1 0	6	ACK		SYN		&		6		F		V		f		v	
0 1 1 1	7	BEL		ETB		'		7		G		W		g		w	
1 0 0 0	8	BS	GET	CAN	SPE	(		8		H		X		h		x	
1 0 0 1	9	HT	TCT	EM	SPD	)		9		I		Y		i		y	
1 0 1 0	10	LF		SUB		*		:		J		Z		j		z	
1 0 1 1	11	VT		ESC		+		;		K		[		k		{	
1 1 0 0	12	FF		FS		,		<		L		\		l		¦	
1 1 0 1	13	CR		GS		–		=		M		]		m		}	
1 1 1 0	14	SO		RS		.		>		N		~		n		~	
1 1 1 1	15	SI		US		/		?		O		—		o		DEL	

4

ADDRESSED COMMAND GROUP (AGG)

UNIVERSAL COMMAND GROUP (UGG)

LISTEN ADDRESS GROUP (LAG)

TALK ADDRESS GROUP (TAG)

PRIMARY COMMAND GROUP (PCG)

SECONDARY COMMAND GROUP (SCG)

NOTES:
1 MSG = INTERFACE MESSAGE
2 B_1 = DI . . . b_7 = DO07
3 REQUIRES SECONDARY COMMAND
4 DENSE SUBSECT (COLUMN 2 THROUGH 5). ALL CHARACTERS USED IN BOTH COMMAND & DATA MODES.

Tabelle 4-2
Codetabelle für CMD-Betrieb

Anhang 7: Interpretercode

ØØØ ZL.ANFG	Ø64	128 END	192 TAN
ØØ1 N.V.	Ø65 A	129 FOR	193 ATN
ØØ2 N.V.	Ø66 B	13Ø NEXT	194 PEEK
ØØ3 N.V.	Ø67 C	131 DATA	195 LEN
ØØ4 N.V.	Ø68 D	132 INPUT#	196 STR$
ØØ5 N.V.	Ø69 E	133 INPUT	197 VAL
ØØ6 N.V.	Ø7Ø F	134 DIM	198 ASC
ØØ7 N.V.	Ø71 G	135 READ	199 CHR$
ØØ8 N.V.	Ø72 H	136 LET	2ØØ LEFT$
ØØ9 N.V.	Ø73 I	137 GOTO	2Ø1 RIGHT$
Ø1Ø N.V.	Ø74 J	138 RUN	2Ø2 MID$
Ø11 N.V.	Ø75 K	139 IF	2Ø3 N.V.
Ø12 N.V.	Ø76 L	14Ø RESTORE	2Ø4 N.V.
Ø13 N.V.	Ø77 M	141 GOSUB	2Ø5 N.V.
Ø14 N.V.	Ø78 N	142 RETURN	2Ø6 N.V.
Ø15 N.V.	Ø79 O	143 REM	2Ø7 N.V.
Ø16 N.V.	Ø8Ø P	144 STOP	2Ø8 N.V.
Ø17 N.V.	Ø81 Q	145 ON	2Ø9 N.V.
Ø18 N.V.	Ø82 R	146 WAIT	21Ø N.V.
Ø19 N.V.	Ø83 S	147 LOAD	211 N.V.
Ø2Ø N.V.	Ø84 T	148 SAVE	212 N.V.
Ø21 N.V.	Ø85 U	149 VERIFY	213 N.V.
Ø22 N.V.	Ø86 V	15Ø DEF	214 N.V.
Ø23 N.V.	Ø87 W	151 POKE	215 N.V.
Ø24 N.V.	Ø88 X	152 PRINT#	216 N.V.
Ø25 N.V.	Ø89 Y	153 PRINT	217 N.V.
Ø26 N.V.	Ø9Ø Z	154 CONT	218 N.V.
Ø27 N.V.	Ø91 [	155 LIST	219 N.V.
Ø28 N.V.	Ø92 √	156 CLR	22Ø N.V.
Ø29 N.V.	Ø93]	157 CMD	221 N.V.
Ø3Ø N.V.	Ø94 ↑	158 SYS	222 N.V.
Ø31 N.V.	Ø95 ←	159 OPEN	223 N.V.
Ø32 SPC	Ø96	16Ø CLOSE	224 N.V.
Ø33 !	Ø97 !	161 GET	225 N.V.
Ø34 "	Ø98 "	162 NEW	226 N.V.
Ø35 #	Ø99 #	163 TAB(	227 N.V.
Ø36 $	1ØØ $	164 TO	228 N.V.
Ø37 %	1Ø1 %	165 FN	229 N.V.
Ø38 &	1Ø2 &	166 SPC(	23Ø N.V.
Ø39 '	1Ø3 '	167 THEN	231 N.V.
Ø4Ø (	1Ø4 (	168 NOT	232 N.V.
Ø41)	1Ø5)	169 STEP	233 N.V.
Ø42 *	1Ø6 *	17Ø +	234 N.V.
Ø43 +	1Ø7 +	171 –	235 N.V.
Ø44 ,	1Ø8 ,	172 *	236 N.V.
Ø45 –	1Ø9 –	173 /	237 N.V.
Ø46 .	11Ø .	174 ↑	238 N.V.
Ø47 /	111 /	176 OR	239 N.V.
Ø48 Ø	112 Ø	177 <	24Ø N.V.
Ø49 1	113 1	178 =	241 N.V.
Ø50 2	114 2	179 >	242 N.V.
Ø51 3	115 3	18Ø SGN	243 N.V.
Ø52 4	116 4	181 INT	244 N.V.
Ø53 5	117 5	182 ABS	245 N.V.
Ø54 6	118 6	183 USR	246 N.V.
Ø55 7	119 7	184 FRE	247 N.V.
Ø56 8	12Ø 8	185 POS	248 N.V.
Ø57 9	121 9	186 SQR	249 N.V.
Ø58 :	122 :	187 RND	25Ø N.V.
Ø59 ;	123 ;	188 LOG	251 N.V.
Ø60 <	124 <	189 EXP	252 N.V.
Ø61 =	125 =	19Ø COS	253 N.V.
Ø62 >	126 >	191 SIN	254 N.V.
Ø63 ?	127 ?		255 π

* N.V. = Nicht verwendet.

ANHANG VII. DER BEFEHL WAIT

Im folgenden ist

A die Adresse einer Speicherstelle (eine ganze Zahl zwischen Null und 65535).
B eine ganze Zahl zwischen Null und 255, und
C eine ganze Zahl zwischen Null und 255.

WAIT A, B, C, bewirkt Anhalten des Programms, bis folgender Ausdruck ungleich Null wird:
((Inhalt von A) EXCLUSIV-ODER (C)) AND (B).

WAIT A, B wartet, bis die Bedingung:
((Inhalt von A) AND (B)) ungleich Null
erfüllt ist.

Der logische Ausdruck EXCLUSIV-ODER (abgekürzt XOR – vom englischen exclusive or) kann in BASIC so gebildet werden:

A XOR C = A AND NOT C OR NOT A AND C

Hauptanwendung des WAIT-Befehls ist das Abwarten, bis ein bestimmtes Bit einer bestimmten Speicherstelle Null (oder Eins) geworden ist.

WAIT A,2↑N (N = 0,1, ... ,7)

wartet, bis Bit N der Adresse A logisch Eins ist.

WAIT A,256-2↑N, 2↑N (N = 0,1, ... ,7)

wartet, bis Bit N der Speicherstelle A logisch Eins ist.

Ein Beispiel:

Die Adressen 141 bis 143 enthalten die laufende Zeit in Sechzigstelsekunden.

Zwei Sekunden Wartezeit kann man so erzeugen:

POKE 143,0 : WAIT 143,2↑7

Der POKE-Befehl setzt alle Bits der Adresse 143 auf Null, dann wartet das Programm, bis das Bit 7 dieser Adresse logisch Eins wird. Dies ist das erste Mal beim Wert 128 (binär 10000000) der Fall, also nach etwa zwei Sekunden (128 Sechzigstelsekunden).

Anmerkung: 1. POKE 143,0 verändert die Variablen TI und TI$, also die interne Uhr.

Ein anderes Beispiel

WAIT 152,1 : WAIT 152,255,1

hält das Programm so lange an, bis die SHIFT-Taste gedrückt (152 enthält 1) und wieder losgelassen wird (152 enthält 0).

Der WAIT-Befehl ist vor allem bei Datenein- und -Ausgabe über Userport und IEEE-Bus nützlich. Hier benötigt man oft die Abfrage auf den Zustand eines bestimmten Bits.

ZU DEN CBM-COMPUTERN 3016-2 UND 3032-2

Nach dem Einschalten zeigt der Bildschirm:

\# \# \# commodore basic \# \# \#

der Computer befindet sich also im Schreibmaschinenmodus.

Die Anpassung der -2-Tastatur an eine gewöhnliche Schreibmaschine hat aber noch andere Auswirkungen:

So besteht der Ziffernblock der Tastatur nur noch aus den Tasten 0 bis 9 und dem Dezimalpunkt. Cursorsteuertasten und die Tasten für +, –, *, / und = sind in den Alphateil der Tastatur verlegt.

Neu sind drei Tasten:

TAB = CHR$ (9)
ESC = CHR$ (27)
REPEAT = CHR$ (63)

TAB und ESC sind vom Benutzer (über geeignete Software) frei programmierbare Tasten. Ohne solche Software wird die Betätigung von TAB oder ESC vom Betriebssystem ignoriert. Drücken von REPEAT hat das selbe zur Folge, wie die Betätigung der Taste ? (PRINT), stellt aber eine Vereinfachung dar, weil ? nur über SHIFT zu erreichen ist.

Die Darstellungsgeschwindigkeit auf dem Bildschirm kann mit der Taste ← verlangsamt werden. (Bei den anderen Versionen geschieht das mit RVS.)

Ferner ist zu beachten, daß ein Teil der Sonderzeichen bei den -2-Geräten auch nach dem Umschalten in den Grafikmodus (POKE 59468,12) nicht über die Tastatur zugänglich ist, sondern nur über die Funktion CHR$. Näheres siehe Tabelle auf der nächsten Seite.

GRAFIKZEICHEN AUF DEM CBM 3016-2 UND 3032-1

GRAPHIC SYMBOL	KEYBOARD ENTRY	GRAPHIC SYMBOL	KEYBOARD ENTRY	GRAPHIC SYMBOL	KEYBOARD ENTRY
♠	A		CHR$ (161)		CHR$ (187)
	B		CHR$ 162		CHR$ 188
	C		CHR$ 163		CHR$ 189
	D		CHR$ 164		CHR$ 190
	E		CHR$ 165		CHR$ 191
	F		CHR$ 166		CHR$ 192
	G		CHR$ 167		CHR$ 219
	H		CHR$ 168		CHR$ 200
	I		CHR$ 169		CHR$ 221
	J		CHR$ 170		CHR$ 223
	K		CHR$ 171	π	CHR$ 255
	L		CHR$ 172		
	M		CHR$ 173		
	N		CHR$ 174		
	O		CHR$ 175		
	P		CHR$ 176		
●	Q		CHR$ 177		
	R		CHR$ 178		
♥	S		CHR$ 179		
	T		CHR$ 180		
	U		CHR$ 181		
	V		CHR$ 182		
	W		CHR$ 183		
♣	X		CHR$ 184		
	Y		CHR$ 185		
◆	Z		CHR$ 186		

Diese Zeichen sind auf dem Bildschirm darstellbar, nachdem POKE 59468,12 ausgeführt worden ist.

Inhaltsverzeichnis

commodore

Commodore Büromaschinen GmbH
Frankfurter Straße 171-175
Postfach 426
6078 Neu-Isenburg
Telefon (06102) *8003 ·Telex 4185663 como d

15d-1079