SIEMENS

Basis-Datensystem 6·000

AMBOSS 1
Commercial BASIC

Programmieranleitung

Basis-Datensystem 6·000
AMBOSS 1
Commercial BASIC

Programmieranleitung

Ausgabe Juli 1979

Inhaltsverzeichnis

		Seite
TEIL I :	EINFÜHRUNG	
1	Allgemeine Einführung in BASIC	8
2	Struktur des Siemens Commercial BASIC	9
2.1	Die BASIC-Sprache	9
2.1.1	BASIC-Quellsprache	9
2.1.2	BASIC Systemkommandos	10
TEIL II :	SPRACHDEFINITION	
1	Einführung	12
2	Datenelemente	17
2.1	Konstante	17
2.1.1	Zahlen	17
2.1.2	Strings	18
2.2	Variable	19
2.2.1	Dezimalvariable	19
2.2.2	Stringvariable	19
2.2.3	Feldvariable	19
3	Ausdrücke	20
3.1	Arithmetische Ausdrücke	20
3.2	Stringausdrücke	21
4	Funktionen	22
4.1	Standardfunktionen	22
4.2	Vom Benutzer definierte Funktionen	23
5	Nicht-ausführbare Anweisungen	24
5.1	Kommentare	24
5.1.1	REM	24
5.2	Datendeklarationen	24
5.2.1	DECIMAL	25
5.2.2	STRING	27

		Seite
5.3	Funktionsdefinition	27
5.3.1	DEF	28
5.3.2	DEF$	29
6	Ausführbare Anweisungen	31
6.1	Zuweisungen	31
6.1.1	LET	32
6.2	Steueranweisungen	32
6.2.1	GOTO	33
6.2.2	IF ... THEN	33
6.2.3	ON ... GOTO	34
6.2.4	FOR	35
6.2.5	NEXT	37
6.2.6	GOSUB	38
6.2.7	RETURN	39
6.2.8	PROC	40
6.2.9	PROCEND	41
6.2.10	CALL	42
6.2.11	ENTER	43
6.2.12	STOP	47
6.2.13	EXIT	47
6.2.14	END	47
6.2.15	START	48
6.3	Ein/Ausgabe-Anweisungen	48
6.3.1	OPEN	48
6.3.2	CLOSE	49
6.3.3	INPUT	50
6.3.4	INPUTC	51
6.3.5	INCHAR	51
6.3.6	PRINT	52
6.4	Andere Anweisungen	53
6.4.1	PRECISION	53
6.4.2	DISPLAY	54
7	Standardfunktionen	56
7.1	Zahlfunktionen	56
7.1.1	ABS	56
7.1.2	INT	56
7.1.3	SGN	57
7.2	Stringverarbeitungsfunktionen	57
7.2.1	STR	57
7.2.2	SUB	58
7.2.3	JNC	58
7.2.4	MSK	59
7.2.5	LEN	60
7.2.6	VAL	60
7.2.7	IDX	60

		Seite
7.3	Ausgabefunktionen	61
7.3.1	CRLF	61
7.3.2	QUOTE	61
7.3.3	TAB	62
7.3.4	BIN	62
7.4	Funktionen für die Bildschirmausgabe	62
7.4.1	LOC	63
7.4.2	ATT	63
7.4.3	NOR	64
7.4.4	UND	64
7.4.5	CLR	64
7.5	Sonstige Funktionen	65
7.5.1	ASCII	65
7.5.2	NEW	65
8	Standardeingabefunktionen	66
8.1	SKIP	66
8.2	FIXED	66

TEIL III: BENUTZUNG DES SYSTEMS

1	Einführung	69
2	Beginn und Ende eines Dialogs mit dem BASIC-System	70
2.1	BYE	71
3	Laden eines BASIC-Programms	72
3.1	GET	72
3.2	SAVE	73
3.3	NEW	74
3.4	KILL	74
3.5	REPLACE	75
4	Auflisten eines BASIC-Quellprogramms	76
4.1	LIST	76

		Seite
5	Einstellungen des Interpreters	78
5.1	SIZE	78
5.2	SIZE$	78
6	Ausführen eines BASIC-Programms	80
6.1	RUN	81
6.2	CONTINUE	81
6.3	ABORT	82
7	Ändern eines Programms	83
7.1	Einfügen und Ändern von Quellzeilen	83
7.2	Löschen von Quellzeilen	84
8	Testen eines Programms	85
8.1	Systemkommandos	85
8.2	Quellanweisungen	85
9	Produktions-Modus	90
9.1	START	90

ANHANG

A	Liste der Schlüsselwörter	91
B	Zeichenvorrat	93
C	Fehlermeldungen	96
D	Beispielprogramm	99
E	Begriffserläuterungen	101
F	Interpreterroutinen	106
INDEX		108
NACHTRAG		114

TEIL I : EINFÜHRUNG

1 Allgemeine Einführung in BASIC

Commercial BASIC ist ein Mittel für die einfache und schnelle Entwicklung von Programmen für kommerzielle Anwendungen. Die Sprache ist interaktiv, dies ermöglicht leichtes Editieren und Modifizieren der Programme.

BASIC wurde ursprünglich am Dartmouth College um 1964 entwickelt. Seitdem wurden zahlreiche Änderungen und Verbesserungen an der Sprache vorgenommen, um sie mehr der jeweiligen Anwendung anzupassen. BASIC wurde eine der beliebtesten Programmiersprachen bei der Benutzung auf Mini- und Mikrocomputern.

Von den vielen Implementierungen, die im Moment verfügbar sind, kann eine große Anzahl als "Commercial BASIC" bezeichnet werden. In den meisten Fällen heißt das, daß Bestandteile des Original-BASIC weggelassen wurden, die in der kaufmännischen Anwendung nicht benötigt werden (z.B. Matrixoperationen, mathematische Funktionen). Im allgemeinen werden auch Eigenschaften hinzugefügt, um die Sprache für den kommerziell ausgerichteten Benutzer nützlicher zu machen. Auch das Commercial BASIC von Siemens geht diesen Weg, fügt aber auch eine Anzahl von neuen Konzepten hinzu, um eine effiziente Benutzung der Bildschirmeinheit zu gewährleisten.

Zu diesen neuen Eigenschaften gehören:

- Funktionen für die Bedienung des Bildschirms
- Formatierung und Typüberprüfung der Eingabe vom Bildschirm
- Stringfunktionen, einschließlich Druckaufbereitung und automatischer Parameteranzeige
- Dezimalarithmetik mit automatischer Behandlung von Genauigkeit und Rundung
- Felder mit Dezimal- und Stringdaten
- Mittel für modulare Programme mit Unterprogrammaufrufen und Schnittstelle zu Assemblerprogrammen

Siemens Commercial BASIC ist mittels eines Interpreters implementiert. Das bedeutet, daß jede Zeile sofort bei der Eingabe überprüft wird. Wenn ein Programm ausgeführt wird, nimmt der Interpreter jede benötigte Quellzeile und führt die angegebenen Aktionen aus. Dies ermöglicht das leichte Ändern und Testen von Programmen.

2 Struktur des Siemens Commercial BASIC

Um das BASIC-System ablaufen zu lassen, werden die folgenden Hard- und Softwarekomponenten benötigt:

- Hardware: sie besteht aus einem Bildschirm-Computer 6.610 mit 48 KBytes Arbeitsspeicher und einem Floppy-Disk-Laufwerk.

- Betriebssystem: der BASIC-Interpreter läuft unter dem Betriebssystem BS 610 (siehe BS 610 Anwender-Handbuch). Es wird benutzt, um den BASIC-Interpreter zu laden, ebenso übernimmt es auch die Dateiverwaltung.

- BASIC-Interpreter: er prüft BASIC-Programme, die vom Benutzer erstellt wurden und führt sie aus. Er führt auch die Aktionen durch, die durch BASIC-Systemkommandos spezifiziert werden.

2.1 Die BASIC-Sprache

Die BASIC-Sprache umfaßt

- die Quellsprache, die benutzt wird, um Programme zu schreiben

- Systemkommandos, die benutzt werden, um das Laden und Ausführen von Programmen zu steuern.

2.1.1 BASIC-Quellsprache

Ein BASIC-Programm besteht aus einer oder mehreren BASIC-Quellzeilen. Jede Zeile besteht aus einer Zeilennummer und einer Quellanweisung. Anweisungen haben das allgemeine Format eines Anweisungsnamens (z.B. LET) i.a. gefolgt durch ein oder mehrere Operanden wie Variablennamen, Funktionsnamen, Ausdrücke usw. Anweisungen werden in der Reihenfolge ihrer Nummern ausgeführt, solange nicht ein expliziter Wechsel in der Folge angegeben ist (z.B. durch eine GOTO-Anweisung).
Über die üblichen, in vielen anderen BASIC-Interpretern vorhandenen Anweisungen hinaus bietet Siemens Commercial BASIC

- umfangreiche Möglichkeiten zur Bildschirmbehandlung (DISPLAY, NEW)

- Formatierung und Typüberprüfung der Eingabe vom Bildschirm (NEW, FIXED)

- Stringfunktionen, einschließlich Druckaufbereitung (MSK) und automatischer Parameteranzeige (DISPLAY)

- Dezimalarithmetik mit automatischer Behandlung von Genauigkeit und Rundung

- Mittel für modulare Programmierung mit Unterprogrammen (PROC, PROCEND, CALL) und Schnittstelle zu Assemblerprogrammen (ENTER).

Die BASIC-Quellsprache erlaubt die Benutzung von Funktionen, die standardmäßig vorhanden, also vordefiniert sind ebenso wie solche, die vom Benutzer innerhalb eines BASIC-Programms definiert werden können. Den Funktionen werden im allgemeinen Parameter übergeben, die in vordefinierter Weise verarbeitet werden. Danach gibt die Funktion ein Resultat (Zahl,String) zurück.

In wissenschaftlich orientierten Implementierungen von BASIC bestehen diese vordefinierten Funktionen fast ausschließlich aus mathematischen Funktionen (z.B. SIN, COS, LOG, usw.). Diese wurden im Commercial BASIC weggelassen; an ihrer Stelle stehen Textverarbeitungs- (SUB), Zahl- (INT) und Spezialfunktionen (NEW), die ein einfaches Mittel zur Benutzung der Möglichkeiten des Bildschirms darstellen.

2.1.2 BASIC-Systemkommandos

Die BASIC-Systemkommandos dienen zur Steuerung der Programmausführung. Sie sind in erster Linie für die Benutzung während der Programmentwicklung gedacht. Sie werden sofort nach der Eingabe über die Tastatur ausgeführt. Ein Systemkommando besteht aus dem Kommandonamen (z.B. SAVE), der, falls nötig, von einem oder mehreren Operanden gefolgt sein kann.

Einige Anweisungen der BASIC-Quellsprache können im STOP-Modus benutzt werden, um das Testen von Programmen zu unterstützen (z.B. PRINT zur Ausgabe des Wertes einer Variablen). Wenn Anweisungen auf diese Weise benutzt werden, sind keine Zeilennummer spezifiziert und die Anweisung sofort ausgeführt.

TEIL II : SPRACHDEFINITION

1 Einführung

Die in diesem Handbuch beschriebene BASIC-Quellsprache besteht aus Anweisungen und vordefinierten Funktionen/ Routinen, die vom BASIC-Interpreter erkannt und interpretiert werden. Teil II dieses Handbuchs beschreibt die Quellsprache im Detail und liefert einige allgemeine Informationen zum System.

Ein BASIC-Programm besteht aus einer Folge von Zeilen, von denen jede eine Zeilennummer gefolgt von einer Quellsprachanweisung enthält. Die Anweisungen werden in der Reihenfolge der Zeilennummern ausgeführt, solange nicht ein Wechsel in der Folge genau festgelegt ist (z.B. durch eine GOTO-Anweisung).

Ein Beispiel eines BASIC-Programms, das den Durchschnitt von n Zahlen berechnet, ist unten gegeben.

Die benutzte Formel ist

$$\text{Durchschnitt} = \sum_{i=1}^{n} \frac{x(i)}{n}$$

d.h. die Summe von n Werten geteilt durch n.

Die in der Rechnung benutzten Daten bestehen aus

- dem Wert n, der die Anzahl der Größen angibt, deren Durchschnitt berechnet werden soll
- n Größen (x(i), i=1,...,n), die in der Rechnung benutzt werden.

Diese Werte werden über die Tastatur eingegeben, jeder abgeschlossen durch einen Wagenrücklauf (CR-Taste).

Das im Beispiel gezeigte Programm besteht aus Kommentarzeilen, einer Deklaration und ausführbaren Anweisungen, die die nötigen Berechnungen vornehmen.

Beispiel:

```
0010 REM Programm zum Berechnen des Durchschnitts von
0015 REM n Zahlen
0020 DECIMAL a(1:100)
0030 REM fuer die Speicherung von n Zahlen
0040 LET sum = 0
0050 REM sum : Summe der Zahlen bis jetzt
0060 LET x = 3
0070 LET y = 2
0080 REM (x,y) ist ein Eingabezeiger
0090 PRINT ,CLR,LOC(2,1), "Wie viele Zahlen"
0100 INPUT ,n
0110 REM n : Anzahl der Zahlen
0120 DISPLAY i$,24,10,4,8
0130 LET i$ = "0"
0140 PRINT ,LOC(24,15),"Zahlen bis jetzt",LOC(x,1),"?"
0150 FOR i = 1 TO n
0160 REM lies naechste Zahl
0170 INPUT ,a(i)
0180 LET i$ = STR(i)
0190 PRINT ,LOC(x,y-1),a(i),"?"
0200 LET sum = sum + a(i)
0210 REM die naechsten 6 Zeilen stellen den Eingabezeiger
0220 LET y = y + LEN(STR(a(i)))  + 1
0230 IF y  = 60 THEN 280
0240 PRINT ,LOC(x,y-1)," "
0250 LET y = 2
0260 LET x = x + 1
0270 PRINT ,LOC(x,1),"?"
0280 NEXT i
0285 PRINT ,LOC(24,1),"
0290 PRINT ,LOC(24,1),"Der Durchschnitt ist ",sum/n,LOC(25,1)
0300 END
```

Kommentarzeilen wie

```
0030 REM für die Speicherung von n Zahlen
```

und

```
0160 REM lies naechste Zahl
```

haben keine Wirkung auf die Ausführung und werden vom Interpreter ignoriert.

Die Zeile

```
OOO2O DECIMAL a(1:100)
```

ist eine Deklaration und legt ein Feld, genannt 'a', von 100 Dezimalwerten fest.

Die Zeilen

```
0040 LET sum = 0
0060 LET x = 3
0070 LET y = 2
```

deklarieren die Variablen 'sum', 'x' und 'y' implizit und weisen ihnen die Anfangswerte 0, 3 bzw. 2 zu. Zu beachten ist, daß implizite Deklarationen beim ersten Auftreten einer Variablen in einer Anweisung, die keine Deklaration ist, ausgeführt werden. In den obigen Beispielen sind die deklarierten Variablen dezimale Größen, da die Bezeichnungen kein $-Zeichen enthalten.

Die Zeile

```
0120 DISPLAY i$, 24, 10, 4, 8
```

ordnet der Variablen i$ ein Bildschirmfeld in der unteren linken Ecke des Schirms zu (Zeile 24, Stellen 10, 11, 12, 13). Da dies das erste Auftreten von i$ ist, ist es zugleich eine implizite Deklaration, und da der Name ein $-Zeichen enthält, ist die Variable vom Typ String. Die Variable i$ hat jetzt noch keinen Wert und deshalb wird kein Wert im Bildschirmfeld angezeigt.

Die Zeile

```
0130 LET i$ = "0"
```

besetzt i$ mit "0" und bewirkt, daß dieser Wert im zugeordneten Bildschirmfeld angezeigt wird.

Die Zeile

```
0140 PRINT ,LOC(24,15), "Zahlen bis jetzt", LOC (x, 1), "?"
```

bewirkt eine Ausgabe auf den Bildschirm. Sie zeigt den Text "Zahlen bis jetzt" ab Position 15 der Zeile 24. Zeile 24 lautet also:

```
          0   Zahlen bis jetzt
```

Diese Anweisung bewirkt auch, daß ein "?" am Beginn der Zeile 3 ausgegeben wird. Das ist das Zeichen, das dem Benutzer anzeigt, daß er die nächste Zahl eingeben soll.

Die Zeilen

```
0150 FOR i = 1 TO n
0170 INPUT ,a(i)
0200 LET sum = sum + a(i)
0280 NEXT i
```

bilden eine FOR-Schleife. Die zwei Zeilen innerhalb der FOR-Schleife (170, 200) lesen eine Zahl von der Tastatur ein und addieren den Wert zur bis jetzt aufgelaufenen Summe. Die anderen 2 Zeilen (150, 280) steuern die Ausführung der FOR-Schleife. Die FOR-Schleife wird n-mal ausgeführt, wobei i, beginnend mit dem Wert 1, jeweils um 1 erhöht wird.

Die Zeile

```
0220 LET y = y + LEN(STR(a(i)))+1
```

ist eine Zuweisung. Sie weist der Variablen y einen neuen Wert zu, nämlich den Wert des arithmetischen Ausdrucks

```
          y + LEN(STR(a(i)))+1
```

Die Zeile

```
0300 END
```

bewirkt die Beendigung der Ausführung. Jedes Programm muß eine END-Anweisung haben, und sie muß immer die letzte Zeile des Programms sein.

Syntax

Die ganze in diesem Handbuch verwendete Syntax ist in einer als Backus-Naur-Form oder BNF bekannten Standardform dargestellt.
Begriffe, die definiert werden, sind in spitzen Klammern <> eingeschlossen. Nur ein solcher Begriff darf auf der linken Seite einer Definition auftauchen. Der Operator

::= bedeutet "ist definiert als" und ein senkrechter Strich | zeigt Alternativen in einer Definition an.

Eine Bezeichnung hat also die Definition

<Bezeichnung>::=<Buchstabe>|<Bezeichnung><Buchstabe>|
<Bezeichnung><Ziffer>|<Bezeichnung>$

Dies ist zu lesen als:

"Eine Bezeichnung ist entweder definiert als ein Buchstabe, oder als eine Bezeichnung gefolgt von einem Buchstaben, oder als Bezeichnung gefolgt von einer Ziffer, oder als eine Bezeichnung gefolgt von einem $-Zeichen".

Dies könnte umschrieben werden als:

"Eine Bezeichnung ist ein Buchstabe, dem eventuell eine Reihe von Buchstaben, Ziffern und $-Zeichen folgt".

Also ist A eine Bezeichnung, weil es der Buchstabe A ist. AB ist eine Bezeichnung, weil es die Bezeichnung A ist, der der Buchstabe B folgt. NAME$ ist eine Bezeichnung, weil es die Bezeichnung NAME ist, der ein $-Zeichen folgt.

2. Datenelemente

Siemens Commercial BASIC kennt zwei Typen von Daten, Strings und Dezimalzahlen. Sie können konstant sein oder als variable Größen vom Interpreter gespeichert werden.

2.1 Konstante

Alle Zahlen und Strings, die als Literale in einem Programm spezifiziert werden, sind Konstanten und können durch das Programm nicht geändert werden.

Beispiele:

```
LET x = 3
```

3 ist eine Konstante und repräsentiert die Zahl 3.

```
LET i$ = "O"
```

"O" ist eine Konstante und repräsentiert den String O.

2.1.1 Zahlen

Zahlen können in der üblichen Form als ganze oder Dezimalzahlen ausgedrückt werden, d. h. sie bestehen aus einer oder mehreren Ziffern, denen eventuell ein Vorzeichen (+ oder -) vorangestellt ist, und falls nicht ganzzahlig, von einem Dezimalpunkt und eventuell einer oder mehreren Ziffern gefolgt werden. Falls die Zahl nicht ganzzahlig ist, muß mindestens eine Stelle vor dem Dezimalpunkt sein.

Beispiele:

```
10
-2
+5
+0.5
-2.459
12.0
10.
```

ungültige Zahlen:

```
.5
-.2
```

2.1.2 Strings

Strings ermöglichen es einem BASIC-Programm, Zeichenfolgen zu bearbeiten. Eine Stringkonstante ist von Apostrophen (") begrenzt und kann jedes Zeichen aus dem verfügbaren Zeichenvorrat (siehe Anhang B) mit Ausnahme von Apostrophen enthalten. Die Länge einer Stringkonstante ist definiert als die Anzahl der Zeichen in diesem String ausschließlich der Apostrophe. Auch Nullstrings (d. h. Strings der Länge Null) können spezifiziert werden.

Beispiele:

```
"JA"                        (Stringlänge = 2)
""                          (Nullstring)
"Das ist ein String"        (Stringlänge = 18)
```

Ungültiger String:

```
"Das ist kein "String""
```

2.2 Variable

Eine variable Größe, die in einem BASIC-Programm erscheint, wird durch einen Variablennamen oder eine Bezeichnung repräsentiert. Eine Bezeichnung ist eine Folge von Buchstaben, Ziffern und $-Zeichen, die mit einem Buchstaben beginnt und bis zu 6 Zeichen lang ist. Jede in einem Programm spezifizierte Bezeichnung korrespondiert zu einer Größe (oder einem Feld von Größen) eines bestimmten Typs (dezimal oder String).

Der Typ einer mit einer Bezeichnung verbundenen Größe ist entweder explizit durch eine Deklarationsanweisung oder implizit spezifiziert, falls die Bezeichnung in keiner Deklarationsanweisung auftritt. Im letzten Fall wird der Typ als dezimal angenommen, es sei denn, die Bezeichnung enthält ein oder mehrere $-Zeichen, im diesem Fall ist sie vom Typ String.

Es wird darauf hingewiesen, daß eine Variable nicht angesprochen werden darf (z. B. auf der rechten Seite einer Zuweisung), bevor ihr nicht ein Wert zugewiesen wurde. Jeder solche Versuch führt während der Ausführung zu einem Fehler.

Beispiele:

```
A
INHALT
S$
B5
```

2.2.1 Dezimalvariable

Dezimalvariable können nur für die Speicherung von Dezimalwerten benutzt werden. Ein Dezimalwert kann ganzzahlig sein oder eine Dezimalzahl mit Nachkommastellen. Die maximale Größe einer Dezimalvariablen wird durch das SIZE-Kommando bestimmt.

Die Anzahl der Nachkommastellen einer variablen Größe wird ihre Genauigkeit genannt. Im Siemens Commercial BASIC kann die Genauigkeit einer Variablen durch eine PRECISION-Anweisung angegeben werden (s. Abschnitt 6.4). Die festgelegte Genauigkeit darf die maximale Länge der Variablen nicht überschreiten. Falls für eine Variable keine Genauigkeit spezifiziert ist, wird sie als Null angenommen, d. h. die Variable ist ganzzahlig.

2.2.2 Stringvariable

Stringvariable werden benutzt, um Folgen von ASCII-Zeichen abzuspeichern.

Die Länge einer Stringvariablen wird entweder explizit in einer Datendeklarationsanweisung (STRING) bestimmt oder durch den voreingestellten Wert bei impliziten Deklarationen. Im letzten Fall wird die voreingestellte Stringlänge durch das SIZE$-Kommando bestimmt (s. Teil III Abschnitt 5.1). Implizit deklarierte Variable werden als vom Typ String angenommen, wenn ein oder mehrere $-Zeichen in der Bezeichnung auftauchen.

2.2.3 Feldvariable

Ein Feld ist eine geordnete Menge von Größen desselben Typs (entweder dezimal oder String). Jedes Element des Feldes wird durch eine indizierte Variable angesprochen. Diese besteht aus einer Bezeichnung gefolgt von einem Index in Klammern. Der Index gibt an, welches Element benötigt wird.

Felder werden durch die Datendeklarationsanweisungen DECIMAL und STRING (s. Abschnitt 5.2) deklariert. Die Deklaration beinhaltet die Festlegung der Feldgröße und Feldgrenzen. Es sind ein- oder zweidimensionale Felder erlaubt.

Die implizite Deklaration von Feldern ist nicht möglich.

3 Ausdrücke

Ein Ausdruck dient zum Festlegen der Berechnung eines Dezimal- oder Textwertes. Der Ausdruck enthält Operanden (z. B. Konstante, Variable, Funktionsaufrufe, usw.), Operatoren (z. B. +, -, usw.) und Klammern.

3.1 Arithmetische Ausdrücke

Ein arithmetischer Ausdruck legt die Berechnung eines Dezimalwertes fest. Den Wert erhält man durch Ausführen der angegebenen arithmetischen Operationen auf den aufgeführten dezimalen Größen entsprechend den folgenden Regeln:

- Ausdrücke in Klammern werden zuerst ausgewertet.
- Operatoren höherer Präzedenz werden vor Operatoren niederer Präzedenz abgearbeitet (d. h. Multiplikation und Division werden vor Addition und Subtraktion ausgeführt).
- Operatoren gleicher Präzedenz (z. B. * und /) werden von links nach rechts ausgeführt.
- Wenn ein Funktionsaufruf auftritt, wird er ausgewertet und der sich ergebende Dezimalwert im Ausdruck eingesetzt.

Beispiele:

```
X(J)
-5
X(J)  - 5

A + B x 5                    diese beiden sind identisch
A + (B x 5)

A / B x C                    diese beiden sind identisch
(A / B) x C

A / (B x C)
VAL(A$)
VAL(A$) - A

-A + B                       diese beiden sind nicht
-(A + B)                     identisch
```

3.2 Stringausdrücke

Ein Stringausdruck legt die Berechnung eines Stringwertes fest. Da es in BASIC keine Stringoperatoren gibt, besteht ein Stringausdruck nur aus Stringvariablen (die indiziert sein können), Konstanten oder Aufrufen von Stringfunktionen.

Beispiele:

A$	Variable
A$(J)	indizierte Variable
"NEIN"	Konstante
STR(65)	Funktionsaufruf

4 Funktionen

Eine Funktion führt eine festgelegte Berechnung aus und gibt ein Ergebnis zurück, wobei die an sie übergebenen Parameterwerte benutzt werden. Das Ergebnis kann, je nach Definition der Funktion, eine Dezimalzahl oder ein String sein.

Funktionen zerfallen in 2 Klassen, systemdefinierte (oder Standardfunktionen), die als Teil der BASIC-Sprache bereitgestellt werden, und benutzerdefinierte Funktionen, die in einem BASIC-Programm zur Benutzung nur innerhalb dieses Programms spezifiziert werden.

4.1 Standardfunktionen

Eine Anzahl von Standardfunktionen werden im Siemens Commercial BASIC bereitgestellt. Sie beinhalten:

- Zahlfunktionen

ABS(A)	Absolutbetrag von A
INT(A)	größte ganze Zahl, die nicht größer als A ist
SGN(A)	Vorzeichenwert von A

- Stringverarbeitungsfunktionen

IDX(S1,S2)	bestimme die Position von S2 in S1
LEN(S)	Länge des Strings S
MSK(S,A)	bereite A zum Druck auf und benutze dabei die Maske S
STR(A)	bilde den zur Zahl A äquivalenten String
VAL(S)	bilde die zum String S äquivalente Zahl
SUB(S,A1,A2)	bilde den Unterstring von S, der bei Position A1 beginnt und die Länge A2 hat

- Ausgabefunktionen

CRLF	Zeichen für Wagenrücklauf/Zei-

	lenvorschub (carriage return/ line-feed)
QUOTE	Apostroph (")
TAB	ASCII-Tabulatorsymbol
BIN(A)	ASCII-Zeichen, das dem dezimalen Parameter entspricht
CLR	Zeichen, das den Bildschirm löscht
NOR	Zeichen, das den Bildschirm in Normalmodus schaltet
UND	Zeichen, das den Bildschirm in Unterstreichungsmodus schaltet.
ATT(A)	Zeichen, das den Bildschirm in den angegebenen Modus schaltet.
LOC(A1,A2)	Zeichenfolge, die den Bildschirmcursor positioniert.

- Andere

NEW(S)	wird für die Eingabe von DISPLAY-Variablen benutzt.

4.2 Vom Benutzer definierte Funktionen

Falls ein Ausdruck mehr als einmal in einem Programm auszuwerten ist, aber mit unterschiedlichen Parameterwerten, kann eine dafür definierte Funktion benutzt werden. Die Funktion wird durch eine Funktionsdeklarationsanweisung, DEF oder DEF$ (s. Abschnitt 5.2), definiert und kann überall benutzt werden, wo ein Funktionsaufruf erlaubt ist (z. B. in einem Ausdruck).

5 Nicht-ausführbare Anweisungen

Ein BASIC-Programm besteht aus einer oder mehreren Anweisungen. Anweisungen können ausführbar sein (d. h. beschreiben die auszuführenden Berechnungen) oder nicht-ausführbar.

Es gibt zwei Arten von nicht-ausführbaren Anweisungen, Kommentare und Deklarationen. Kommentare dienen als Dokumentationshilfe und machen ein Programm leichter lesbar. Sie werden vom Interpreter ignoriert.

Durch Deklarationen werden Bezeichnungen und ihre Kennwerte für den Interpreter definiert. Bezeichnungen können sich auf Daten (Variable oder Felder) oder Funktionen beziehen, jeweils vom Typ String oder Dezimalzahl.

5.1 Kommentare

Kommentare bestehen aus textueller Information und können überall in einem Programm eingefügt werden, wo eine Anweisung erlaubt ist. Sie können dazu benutzt werden, um ein Programm lesbarer zu machen und stellen Dokumentation über das Programm dar.

5.1.1 REM

Die REM-Anweisung wird benutzt, um Kommentare zu kennzeichnen.

Format:

REM<beliebige Zeichenfolge>

Die REM-Anweisung wird vom Interpreter ignoriert und dient nur für Dokumentationszwecke.

Beispiele:

```
0001 REM Dies ist ein Kommentar
0010 REM Programm zur Berechnung des Durchscnitts
0015 REM von n Zahlen
```

5.2 Datendeklarationen

Datendeklarationen werden benutzt, um die Eigenschaften von im Programm verwendeten Variablen zu definieren,

und um diesen Variablen Bezeichnungen zuzuordnen. Variable können vom Typ Dezimalzahl oder String, eine einfache Variable oder Felder mit ein oder zwei Dimensionen sein.

Bezeichnungen sind von der Form "Buchstabe gefolgt von bis zu 5 Zeichen, die Buchstaben, Ziffern oder $-Zeichen sein können",

z. B. a
A
ZAHL
STR$
A$$1
B$X
SUM

Eine Bezeichnung kann nur in einer Deklaration auftauchen und behält die mit ihr zu diesem Zeitpunkt verbundenen Eigenschaften während des ganzen Programms.

Im Siemens Commercial BASIC müssen nicht alle Variablen deklariert werden. Implizite Deklarationen erfolgen, wenn eine Bezeichnung das erste Mal in einer Anweisung auftaucht, die keine Deklarationsanweisung ist. In diesem Fall wird die deklarierte Variable als vom Typ Dezimalzahl angenommen, es sei denn, die Bezeichnung enthält ein oder mehrere $-Zeichen; in diesem Fall ist sie vom Typ String. Nur einfache Variable können implizit deklariert werden - Feldvariable müssen durch eine Deklarationsanweisung erklärt werden.

Auf keine Variable (ob deklariert oder nicht) darf Bezug genommen werden bevor ihr nicht ein Wert zugewiesen worden ist. Jeder solche Versuch führt während der Ausführung zu einem Fehler.

Die in BASIC bereitgestellten Datendeklarationsanweisungen sind DECIMAL und STRING.

5.2.1 DECIMAL

Die DECIMAL-Anweisung wird benutzt, um einfache oder indizierte Variable vom Typ Dezimalzahl zu deklarieren. Nur Dezimaldaten (d. h. Zahlen) dürfen in Dezimalvariablen gespeichert werden.

Format:

DECIMAL Liste mit Bezeichnungen

wobei

<Liste mit Bezeichnungen>::=
<Bezeichnung>|<Feldbezeichnung>|

<Liste mit Bezeichnungen>,<Bezeichnung>|
<Liste mit Bezeichnungen>,<Feldbezeichnung>

d. h. <Liste mit Bezeichnungen> besteht aus einer oder mehreren durch Komma getrennten Bezeichnungen, die die zu deklarierenden Variablen spezifizieren. Die aufgeführten Bezeichnungen können sich auf einfache oder Feldvariable beziehen.

Eine einfache Variable wird durch Angabe ihres Namens in der Liste deklariert. Der Variablen wird Speicherplatz zugewiesen, der groß genug ist für eine Dezimalzahl der durch das SIZE-Kommando (s. Systemkommandos) festgelegten Größe. Jeder folgende Bezug auf diese Bezeichnung führt dazu, daß auf die in der Variablen gehaltenen Daten als dezimale Größe zugegriffen wird.

Falls eine Feldvariable deklariert wird, muß eine Feldbezeichnung in der Deklarationsanweisung angegeben sein. Eine Feldbezeichnung ist wie folgt definiert:

<Feldbezeichnung>::=
<Bezeichnung>(<Untergrenze>:<Obergrenze>)|
<Bezeichnung>(<Untergrenze>:<Obergrenze>,
<Untergrenze>:<Obergrenze>)

wobei

<Untergrenze>::= 0|1
<Obergrenze> ::= <ganzzahlige Konstante>

Eine Felddeklaration muß die Spezifikation der Dimensionen und Grenzen des Feldes beinhalten. Für eindimensionale Felder wird die erste Form der Feldbezeichnung benutzt, und Untergrenze und Obergrenze legen die Unter- und Obergrenze des Feldes fest. Zweidimensionale Felder werden durch die zweite Form der Feldbezeichnung festgelegt. In beiden Fällen kann die Untergrenze nur den Wert 0 oder 1 annehmen. Die Obergrenze muß eine ganzzahlige Konstante sein, die nicht kleiner als die Untergrenze ist.
Der Zugriff auf Feldelemente erfolgt durch indizierte Variable, die aus der Feldbezeichnung gefolgt von einem oder zwei in Klammern eingeschlossenen Indizes bestehen (z. B. A(2) oder B(I,J)). Die Indexwerte dürfen nicht außerhalb der in der Felddeklaration festgelegten Grenzen liegen.
Durch die Deklaration eines Feldes wird Speicherplatz für jedes Element des Feldes zugewiesen. Jedes Element eines Feldes wird genauso behandelt wie eine einfache Variable.

Beispiele:

```
0001 DECIMAL a
```

```
0002 DECIMAL x(1:10)
0003 DECIMAL I,J,K
0004 DECIMAL B1(1:5),B2(1:5,1:4),N
```

5.2.2 STRING

Die STRING-Anweisung wird benutzt, um einfache oder indizierte Variable vom Typ String zu deklarieren. Jede Folge von ASCII-Zeichen kann in einer Stringvariablen gespeichert werden.

Format:

```
STRING (<Länge )<Liste mit Bezeichnungen>
STRING <Liste mit Bezeichnungen>
```

wobei

```
<Liste mit Bezeichnungen>::=
<Bezeichnung><Feldbezeichnung>|
<Liste mit Bezeichnungen>,<Bezeichnung>|
<Liste mit Bezeichnungen>,<Feldbezeichnung>
```

und

```
<Länge>::=<ganzzahlige Konstante>
```

Die Deklaration von Stringvariablen ist in der Wirkung sehr ähnlich zur Deklaration von Dezimalvariablen, außer daß die Größe des zugewiesenen Speicherplatzes von der angegebenen Stringlänge abhängt. Jedes Element des Stringfeldes belegt die in Länge angegebene Anzahl von Bytes (ein Byte = ein ASCII-Zeichen). Die Länge muß eine Zahl aus dem Bereich von 1 bis 255 sein. Falls keine Länge angegeben ist, wird der Wert von SIZE$ angenommen.

Stringfelder sind ebenfalls erlaubt und haben die gleichen Eigenschaften wie Dezimalfelder.

Beispiele:

```
0010 STRING (8) STR1,A$
0015 STRING (5) X$(1:10)
0020 STRING a$1 (1:9),B$$B(0:100)
```

5.3 Funktionsdefinition

Um häufiger benutzte Ausdrücke auszuwerten, kann der Benutzer seine eigenen Funktionen definieren. Eine Funktion gibt als Wert, durch den sie dann ersetzt wird, einen Wert des Typs zurück, der in der Funktionsdeklaration festgelegt wurde.

Der auszuwertende Ausdruck kann Parameter enthalten,

deren Werte von den aktuellen, im Funktionsaufruf spezifizierten Werten abhängen.

Es gibt 2 Anweisungen, um Funktionen zu definieren, DEF für Dezimalfunktionen und DEF$ für Stringfunktionen.

5.3.1 DEF

Mit der DEF-Anweisung kann der Benutzer Dezimalfunktionen definieren, d. h. Funktionen, die eine Dezimalzahl als Ergebnis haben.

Format:

```
DEF <Bezeichnung> =<Ausdruck>
DEF <Bezeichnung> (<Liste mit formalen Parametern>)
                  =<Ausdruck>
```

Die angegebene <Bezeichnung> ist der Name der Funktion, durch den sie in der Folge aufgerufen wird. Der Ausdruck legt die auszuführenden Berechnungen fest, die erforderlich sind, um den benötigten Funktionswert zu erhalten.

Die erste Form der DEF-Anweisung, die oben gezeigt ist, bezieht sich auf eine parameterlose Funktion, d.h. alle im Ausdruck benutzten Bezeichnungen beziehen sich auf anderswo deklarierte Variable.

Die zweite Form der DEF-Anweisung beinhaltet eine Liste mit formalen Parametern, die die im Ausdruck benutzten Parameter bezeichnet.

```
<Liste mit formalen Parametern>::=
     <formaler Parameter>|
     <Liste mit formalen Parametern>,<formaler
      Parameter>
```

Eine Liste mit formalen Parametern besteht also aus einer oder mehreren durch Komma getrennten Bezeichnungen. Beim Aufruf einer Funktion werden die formalen Parameter im auszuwertenden Ausdruck durch die aktuellen, angegebenen Parameterwerte ersetzt. Formale Parameter sind implizit vom Typ String, falls sie ein oder mehrere $-Zeichen enthalten; sonst sind sie vom dezimalen Typ.

Die aktuellen Parameter müssen in Anzahl und Typ mit den in der DEF-Anweisung angegebenen formalen Parametern Übereinstimmen.

Beispiele:

Falls eine Funktion wie folgt definiert ist:

```
0950 DEF PROD=X*Y
```

dann ergibt die Benutzung der Bezeichnung PROD in einer Anweisung einen Wert, der gleich dem Produkt der aktuellen Werte der Variablen X und Y ist (z. B. falls X = 3 und Y = 4 ist, ist das Ergebnis 12).

Falls jedoch die Funktion als

```
0950 DEF PROD(X,Y)=X*Y
```

definiert ist, dann wäre der Funktionsaufruf von der Form PROD(I,J) und der Wert, den man erhält, wäre abhängig vom Inhalt der Variablen I und J (z. B. falls I=2 und J=5 wäre, dann ergäbe PROD (I,J)=10, wenn aber I=4 und J=1 wäre, dann ergäbe PROD(I,J)=4, usw.).

5.3.2 DEF$

Mit der DEF$-Anweisung kann der Benutzer Stringfunktionen definieren, d. h. Funktionen, die einen String als Ergebnis haben.

Format:

```
DEF$ <Bezeichnung>=<Stringausdruck>
DEF$ <Bezeichnung>(<Liste mit formalen Parametern>) =
                        <Stringausdruck>
```

wobei <Bezeichnung> den Namen der Funktion festlegt, unter dem sie im folgenden aufgerufen wird.

<Stringausdruck> legt die Berechnung fest, die ausgeführt werden muß, um den gewünschten Funktionswert zu erhalten.

Die erste Form der DEF$-Anweisung spezifiziert eine parameterlose Funktion. Das bedeutet, daß sich alle im Stringausdruck benutzten Variablen auf anderswo deklarierte Variable beziehen.

Die zweite Form enthält eine Liste mit formalen Parametern, die die im Ausdruck zu benutzenden Parameter bezeichnet. Die Liste mit den formalen Parametern ist eine Menge von durch Komma getrennten Bezeichnungen.

Wenn die Funktion aufgerufen wird, müssen die formalen Parameter durch die aktuellen Parameter ersetzt werden.

Der Typ jedes formalen Parameters ist implizit bestimmt. Wenn die Bezeichnung ein oder mehrere $-Zeichen enthält, dann ist sie vom Typ String, sonst vom dezimalen Typ. Die aktuellen, im Funktionsaufruf angegebenen Parameter müssen in Anzahl und Typ mit den in der DEF$ Anweisung angegebenen Parametern übereinstimmen.

Beispiele:

```
1010 DEF$ blink = ATT(48)
1020 DEF$ NUM$(A,B) = JNC(STR(A),STR(B))
```

6 Ausführbare Anweisungen

Ausführbare Anweisungen beschreiben die durch ein Programm auszuführenden Berechnungen. BASIC besitzt vier Arten von ausführbaren Anweisungen:

- Zuweisungen
  Sie bewirken die Besetzung von Variablen mit neuen Werten.

- Steueranweisungen
  Steueranweisungen werden benutzt, um die Ausführungsreihenfolge der Zeilen eines Programms zu ändern. Sie ermöglichen Sprünge auf andere Teile des Programms, Ausführung von Schleifen, Unterprogramme und Prozeduren und die Beendigung des Programms.

- Ein/Ausgabeanweisungen
  Ein/Ausgabeanweisungen dienen dazu, daß ein Programm Daten von einer externen Quelle empfangen und Ergebnisse zu einem externen Ziel senden kann. Diese Quellen und Ziele können sein:

  o Dateien auf Floppy-Disk/Kassette
  o Bildschirmeinheit
  o Ein anderer Rechner
  o Ein Drucker.

- Andere
  Es gibt zwei andere ausführbare Anweisungen,die nicht eindeutig in eine der obigen Kategorien fallen. Dies sind:

  o PRECISION: wird benutzt, um die Anzahl der Nachkommastellen festzulegen.

  o DISPLAY: wird benutzt,um eine Stringvariable mit einem Feld auf dem Bildschirm zu verbinden.

6.1 Zuweisungen

Eine Zuweisung wertet einen Ausdruck aus und weist den sich ergebenden Wert einer angegebene Variable zu. Der Ausdruck und die Variable müssen vom selben Typ sein (String oder Dezimalzahl). Es gibt nur eine Anweisung für die Zuweisung in BASIC. Dies ist die LET-Anweisung.

6.1.1 LET

Die LET-Anweisung wird benutzt, um Stringausdrücke an Stringvariable und arithmetische Ausdrücke an Dezimalvariable zuzuweisen.

Format:

```
LET<Variable>=<Ausdruck>
LET<Stringvariable>=<Stringausdruck>
```

wobei <Variable> und <Stringvariable> folgendermaßen definiert sind:

```
<Bezeichnung>|<Bezeichnung>(<Indexliste>)
```

In der ersten Version der LET-Anweisung wird also das Ergebnis eines arithmetischen Ausdrucks an eine Dezimalvariable zugewiesen, deren Bezeichnung auf der linken Seite des Gleichheitszeichens (=) erscheint.

In der zweiten Version wird das Ergebnis eines Stringausdrucks an eine Stringvariable zugewiesen, deren Bezeichnung auf der linken Seite des Gleichheitszeichens erscheint.

Beispiele:

```
0040 LET sum = 0
0130 LET i$ ="0"
0180 LET i$ = STR(i)
0220 LET y = y + LEN(STR(a(i))) + 1
```

6.2 Steueranweisungen

Die Anweisungen eines BASIC-Programms werden in der Reihenfolge der Zeilennummern ausgeführt. Steueranweisungen ermöglichen es, die Reihenfolge, in der die Programmbefehle ausgeführt werden, abzuändern.
Es sind Anweisungen verfügbar, um:

- in einen neuen Bereich des Programms zu springen, z. B. GOTO.
- einen Teil des Programms wiederholt in einer Schleife auszuführen, (FOR ... NEXT).
- eine gemeinsame Folge von Anweisungen durch GOSUB oder CALL auszuführen und dann zur Anweisung hinter dem GOSUB- oder CALL-Aufruf zurückzukehren.
- die Programmausführung anzuhalten, entweder zeitweilig (STOP) oder endgültig (EXIT, END).

6.2.1 GOTO

Die GOTO-Anweisung bewirkt einen unbedingten Sprung zu einem anderen Teil des Programms.

Format:

GOTO <Zeilennummer>

wobei <Zeilennummer> eine Dezimalzahl im Bereich 1 bis 9999 sein und einer existierenden Zeilennummer des gleichen Programms entsprechen muß.

Als nächstes wird die Anweisung mit der angegebenen Zeilennummer ausgeführt und von dort geht die Programmausführung in der normalen Reihenfolge weiter.

Beispiele:

```
0100 GOTO 200
1500 GOTO 10
```

6.2.2 IF ... THEN

Die IF ... THEN-Anweisung ermöglicht es dem Programm, die aktuellen Werte von Variablen abzufragen und die Ausführung in Abhängigkeit vom Ergebnis an zwei verschiedenen Stellen fortzusetzten. Die zwei Stellen sind:

- die nächste Zeile
- die in der IF ... THEN-Anweisung angegebene Zeile.

Format:

IF <Vergleich> THEN <Zeilennummer>

wobei <Vergleich> definiert ist als:
<Ausdruck><Vergleichsoperator><Ausdruck>|
<Stringausdruck><Vergleichsoperator><Stringausdruck>

Ein <Vergleich> besteht also aus zwei Ausdrücken desselben Typs, die durch einen <Vergleichsoperator> getrennt sind.

Es gibt folgende <Vergleichsoperatoren>:

>	größer als
<	kleiner als
<=	kleiner als oder gleich
>=	größer als oder gleich
=	gleich
<>	ungleich

Der Vergleich ist entweder wahr oder falsch, je nachdem ob der Vergleich für die aktuellen Werte der Ausdrücke auf den beiden Seiten des Vergleichsoperators gilt oder nicht.

Z. B. ist a = b + 1 genau dann wahr, wenn der aktuelle Wert von a um 1 größer ist als der aktuelle Wert von b.

Falls der Vergleich wahr ist, bewirkt die IF ... THEN-Anweisung einen Sprung auf die angegebene Zeilennummer; d. h. die Anweisung mit der angegebenen Zeilennummer wird als nächste ausgeführt und die Programmausführung setzt normal von dort fort.

Im anderen Fall hat die IF ... THEN-Anweisung keine Wirkung und die Ausführung des Programms geht bei der nächsten Zeile weiter.

Wenn zwei Zahlen unterschiedlicher Genauigkeit verglichen werden, so wird die kürzere mit Nullen verlängert, so daß der Vergleich wie erwartet ausfällt, z. B. ist bei

```
IF 2.3 = 2.300 THEN 200
```

der Vergleich wahr.

Beispiele:

```
0230 IF y< 60 then 280
```

6.2.3 ON ... GOTO

Die ON ... GOTO-Anweisung ermöglicht es, auswählbare Sprünge durchzuführen, wobei das Ziel des GOTO entsprechend dem Wert des angegebenen Ausdrucks ausgewählt wird.

Format:

ON <Ausdruck> GOTO <Liste mit Zeilennummern>

wobei <Ausdruck> ein arithmetischer Ausdruck ist. Falls das Ergebnis dieses Ausdrucks keine ganze Zahl ist, wird es zur nächstgelegenen ganzen Zahl gerundet.

<Liste mit Zeilennummern> ist eine Folge von durch Kommata getrennten Zeilennummern.

Falls das Ergebnis des <Ausdrucks> 1 ist, wird ein GOTO zur ersten Zeilennummer in der Liste ausgeführt.

Falls das Ergebnis des <Ausdrucks> 2 ist, wird ein GOTO zur zweiten Zeilennummer in der Liste ausgeführt.

Falls das Ergebnis des Ausdrucks kleiner als oder gleich 0 oder größer als die Anzahl der Zeilennummern in der Liste ist, wird die Ausführung in der nächsten Zeile fortgesetzt.

Die Anzahl der in der Liste angegebenen Zeilennummern ist nur durch die Zeilenlänge des Bildschirms begrenzt, d. h. durch die Anzahl von Zeilennummern, die in einer Zeile geschrieben werden können.

Beispiel:

```
0090 REM lies Schluessel in n
0100 INPUT SI,n
0110 REM springe nach Zeile 200, falls n=1
0120 REM    "      "     "   300, falls n=2
0130 REM    "      "     "   400, falls n=3
0140 REM andernfalls springe nach Zeile 900
0150 ON n GOTO 200,300,400
0160 GOTO 900
.
.
.
```

6.2.4 FOR

Die FOR-Anweisung wird benutzt, um eine Schleife im BASIC-Programm zu kennzeichnen, die mehrfach durchlaufen werden soll.Das Ende einer solchen Schleife wird durch die zugehörige NEXT-Anweisung angezeigt. Zusammengehörige FOR- und NEXT-Anweisungen müssen die gleiche "Schleifenvariable" bezeichnen.

FOR-Schleifen können in beliebiger Tiefe geschachtelt sein, d. h. die Anweisungen zwischen einer FOR-Anweisung und der zugehörigen NEXT-Anweisung können eine beliebige Anzahl anderer FOR-Schleifen enthalten. FOR-Schleifen dürfen sich jedoch nicht überlappen, d. h. eine FOR-Anweisung und das folgende NEXT müssen dieselbe Schleifenvariable besitzen, z. B.

```
FOR i = 1 TO 5
FOR j = 1 TO 5
.
.
.
NEXT j
NEXT i
```

Format:

FOR <Bezeichnung> = <Ausdruck> TO <Ausdruck>
FOR <Bezeichnung> = <Ausdruck> TO <Ausdruck> STEP <Ausdruck>

wobei <Bezeichnung> eine Dezimalvariable bezeichnet und alle drei Audrücke arithmetische Ausdrücke sind. Falls ihr Ergebnis nicht ganzzahlig ist, wird es zur nächstgelegenen ganzen Zahl gerundet.

Die zweite Version ist die allgemeinere Form der FOR-Anweisung. Die erste Version ist die eingeschränkte Form, bei der der STEP Wert 1 ist (d. h. STEP 1). In dem Fall kann der STEP-Teil weggelassen werden.

Wenn die FOR-Anweisung ausgeführt wird, wird die Schleifenvariable am Anfang auf den Wert des ersten Ausdrucks gesetzt.

Die Schleifenvariable wird dann mit dem Wert des zweiten (oder TO) Ausdrucks verglichen und falls sie kleiner als dieser oder gleich diesem Wert ist, wird die Ausführung bei der Anweisung nach der FOR-Anweisung fortgesetzt, es sei denn, der STEP-Ausdruck ist negativ.

Falls der STEP-Ausdruck negativ ist, zählt die Schleifenvariable vom Anfangswert zum Endwert (TO-Ausdruck) herunter. Der Vergleich in der FOR-Anweisung muß also prüfen, ob die Schleifenvariable größer als oder gleich dem TO-Ausdruck ist, und die Schleife beenden, wenn dies nicht mehr zutrifft.

Wenn die NEXT-Anweisung gefunden wird, wird die Schleifenvariable um den Wert des dritten oder STEP-Ausdrucks erhöht (oder um 1, falls kein STEP-Ausdruck angegeben ist). Die Ausführung kehrt dann zu dem Punkt zurück, wo die Schleifenvariable mit dem TO-Ausdruck verglichen wird.

Die Anweisungen innerhalb der FOR-Schleife werden auf diese Weise wiederholt ausgeführt, bis die Schleifenvariable größer (bzw.kleiner) wird als der Wert des TO-Ausdrucks in der FOR-Anweisung. Wenn dies geschieht, wird die Ausführung des Programms bei der Anweisung hinter der zugehörigen NEXT-Anweisung fortgesetzt.

Es wird darauf hingewiesen, daß, falls bei positivem STEP-Ausdruck der Anfangswert der Schleifenvariable größer ist als der TO-Ausdruck (oder bei negativem STEP-Ausdruck der Anfangswert kleiner ist als der TO-Ausdruck) die Schleife überhaupt nicht ausgeführt wird.

6.2.5 NEXT

Die NEXT-Anweisung zeigt das Ende einer FOR-Schleife an.

Format:

```
NEXT <Bezeichnung>
```

wobei <Bezeichnung> eine Schleifenvariable ist und benutzt wird, um anzuzeigen, welche FOR-Schleife durch diese NEXT-Anweisung beendet wird.

FOR-Schleifen können in beliebiger Tiefe geschachtelt sein, dürfen sich aber nich überlappen.

Also ist

```
FOR i = 1 TO 10
FOR j = i TO 10
FOR k = j TO 10
.
.
.
NEXT k
NEXT j
NEXT i
```

mit drei ineinandergeschachtelten FOR-Schleifen erlaubt.

Jedes Programm, das jedoch die Folge:

```
FOR i =
NEXT j
```

ohne dazwischenliegende FOR- oder NEXT-Anweisungen enthält, ist falsch.

Beispiele:

```
(a) 0040 LET sum = 0
    0100 INPUT ,n
    0150 FOR i = 1 TO n
    0170 INPUT ,a(i)
    0200 LET sum = sum + a(i)
    0280 NEXT i
    0400 PRINT ,"Durchschnitt ist ",sum/n
```

Dieses Beispiel stammt aus dem Beispielprogramm am Anfang des Teils II dieses Handbuchs. Es zeigt:

- die Vorbesetzung von sum mit 0 und n mit einem Wert, der von der Tastatur eingegeben wird

- eine FOR-Schleife (Zeilen 150 bis 280), die n Werte von der Tastatur einliest und sie zu einer vorläufigen Gesamtsumme in sum addiert

- eine Anweisung, um das Ergebnis auszugeben, das den Durchschnitt der eingegebenen Zahlen darstellt.

(b)

```
      .
      .
   10 FOR i = 1 TO 10
   20 FOR j = 1 TO 10
   30 PRINT ,a(i,j)
   40 NEXT j
   50 PRINT ,CRLF
   60 NEXT i
```

Dieses Beispiel kann dazu benutzt werden, um die Elemente eines 2-dimensionalen Feldes auszugeben. Es beinhaltet geschachtelte FOR-Schleifen, wobei die Schleifenvariablen als Indizes des Feldes benutzt werden.

6.2.6 GOSUB

Mit der GOSUB-Anweisung kann der Programmierer Unterprogramme aufrufen, so daß allgemein benutzte Folgen von Anweisungen einmal definiert und überall, wo benötigt, benutzt werden können. Die Ausführung einer GOSUB-Anweisung stößt die Ausführung des zugehörigen Unterprogramms an. Nach Beendigung der Ausführung des zugehörigen Unterprogramms bewirkt die Ausführung der RETURN-Anweisung, daß das Programm bei der der GOSUB-Anweisung folgenden Anweisungen fortgesetzt wird.

Format:

GOSUB <Zeilennummer>

wobei <Zeilennummer> eine ganze Zahl zwischen 1 und 9999 ist und einer Zeilennummer des aktuellen Programms entsprechen muß.

Die GOSUB-Anweisung bewirkt, daß der Programmfluß zur angegebenen Zeilennummer übergeht (wie in der GOTO-Anweisung). Die GOSUB-Anweisung speichert jedoch auch eine Rückkehradresse im sogenannten Kellerspeicher, so daß nach Beendigung des aufgerufenen Unterprogramms eine Rückkehr zur der der GOSUB-Anweisung folgenden Anweisung erfolgen kann. Dieser Keller kann mehrere Einträge (oder Rückkehradressen) enthalten, abhängig von der Anzahl noch in Bearbeitung befindlichenr GOSUB-Anweisungen.

Die Ausführung einer RETURN-Anweisung bewirkt, daß der letzte Eintrag (d. h. der zeitlich am kürzesten zurückliegende Eintrag) aus dem Keller entfernt und benutzt wird, um eine Rückkehr zur der der entsprechenden GOSUB-Anweisung folgenden Anweisung zu erreichen.

GOSUB-Anweisungen können also in beliebiger Tiefe geschachtelt sein, d. h. GOSUB-Anweisungen können in durch GOSUB aufgerufenen Unterprogrammen auftreten.

6.2.7 RETURN

Die RETURN-Anweisung kennzeichnet das Ende eines Unterprogramms, das durch eine GOSUB-Anweisung angestoßen wurde. RETURN bewirkt, daß die Ausführung des Programms zu der der entsprechenden GOSUB-Anweisung folgenden Anweisung zurückkehrt.

Format:

```
RETURN
```

Achtung: Falls ein Programm irrtümlich in ein Unterprogramm "gerät" (d. h. das Unterprogramm wird auf andere Weise als durch eine GOSUB-Anweisung aktiviert) und dann die RETURN-Anweisung ausgeführt wird, paßt das GOSUB-RETURN-Paar nicht zusammen, und damit alle folgenden Paare. Wenn eine RETURN-Anweisung ausgeführt wird und keine Einträge in der Liste mit den Rückkehradressen, also im Keller sind, wird das Programm mit einem Fehler beendet.

Beispiel:

```
0100 STRING (80) x$
.
.
.
1000 EXIT
5000 OPEN SI,a$
5010 OPEN SO,b$
5020 INPUT SI,x$
5030 PRINT SO,QUOTE,x$,QUOTE,CRLF
5040 IF x$  "" THEN 5020
5050 CLOSE SI
5060 CLOSE SO
5070 RETURN
```

Dieses Beispiel zeigt ein Unterprogramm, das benutzt werden kann, um eine Datei zu kopieren. Die Datei, deren Name der String in a$ ist, wird in eine Datei kopiert, deren Name in b$ gespeichert ist. Die Datei a$ sollte eine Folge von Strings enthalten, die in Apostrophen eingeschlossen und durch Wagenrücklauf/

Zeilenvorschub getrennt sind. Die Datei muß vom Nullstring abgeschlossen sein.

Das Beispiel zeigt:

- eine EXIT-Anweisung vor dem Unterprogramm, um einen unerlaubten Eintritt in das Unterprogramm zu verhindern.

- einen String x$, der einmal am Anfang des Programms deklariert und bei jedem Aufruf des Unterprogramms benutzt wird.

Ein Beispiel eines Aufrufs wäre:

```
0200 LET a$ = n$
0210 LET b$ = "DATEI"
0220 GOSUB 5000
```

6.2.8 PROC

Die PROC-Anweisung zeigt den Beginn einer Prozedur an. Sie gibt den Namen der Prozedur und die Namen ihrer formalen Parameter an. Alle Variablen, die innerhalb einer Prozedur benutzt werden, sind lokal in dieser Prozedur, d. h. zwei Variable mit der gleichen Bezeichnung, eine Variable innerhalb der Prozedur und die andere innerhalb des aufrufenden Programmteils, sind zwei verschiedenen Variable und die Veränderung der einen hat keine Auswirkung auf die andere.

Eine Prozedur entspricht also einem Unterprogramm, mit der Ausnahme, daß sie statt der Variablen des Hauptprogrammes Variable benutzt, die nur für sie selbst verwendbar sind. Falls es nötig ist, daß die Prozedur auf anderswo im Programm benutzte Variable zugreift, dann müssen diese als Parameter übergeben werden.

Die formalen Parameter, deren Namen in der PROC-Anweisung angegeben sind, werden als Variable in der Prozedur benutzt. Sobald die Prozedur jedoch ausgeführt wird, werden die in der CALL-Anweisung angegebenen aktuellen Parameter für die formalen Parameter eingesetzt. Alle Bezüge auf formale Parameter werden also zu Bezügen auf die aktuellen Parameter, die Konstanten oder Variablen des Hauptprogramms sein können. Der Typ eines formalen Parameters kann explizit durch eine Deklaration innerhalb der Prozedur festgelegt werden.

Format:

PROC <Bezeichnung>(<Liste mit formalen Parametern>)

PROC <Bezeichnung>

wobei <Bezeichnung> der Name der Prozedur ist, unter

dem sie in CALL-Anweisungen aufgerufen wird.

<Liste mit formalen Parametern> ist eine Menge von durch Komma getrennten Bezeichnungen. Diese Variablennamen sind lokal innerhalb dieser Prozedur. Wenn die Prozedur ausgeführt wird, werden die in der CALL-Anweisung angegebenen aktuellen Parameter für die formalen Parameter eingesetzt.

Die Anzahl der formalen Parameter ist nur durch die Länge einer Eingabezeile begrenzt.

GOTO- und GOSUB-Anweisungen in einer Prozedur dürfen keine Zeilennummern außerhalb der Prozedur ansprechen.

Format:

```
10 STRING A(1:5)
20 CALL FRED(A)
.
.
.
100 PROC FRED(FLD)
110 STRING FLD(1:5)
```

Prozeduren dürfen andere Prozeduren aufrufen; rekursive Aufrufe sind jedoch verboten, d. h. eine Prozedur darf sich nicht selbst aufrufen, weder direkt noch indirekt. Prozedur A darf z. B. nicht A aufrufen, noch darf sie Prozedur B aufrufen, falls B (oder irgendeine von B aufgerufene Prozedur) A aufruft.

Prozeduren dürfen nicht geschachtelt sein, d. h. in einer Prozedur darf keine andere Prozedur deklariert werden.

6.2.9 PROCEND

Der PROCEND-Aufruf zeigt das Ende einer Prozedur an. Alle Zeilen zwischen einem PROC- und einem PROCEND-Aufruf sind Teile der Prozedur. Alle Variablen, die in der Prozedur benutzt werden, sind in der Prozedur lokal.

Format:

```
PROCEND
```

Wenn eine PROCEND-Anweisung ausgeführt wird, wirkt sie wie ein RETURN aus einem GOSUB-Unterprogramm. Die Steuerung geht zu der Anweisung zurück, die der CALL-Anweisung folgt, durch die die Prozedur aufgerufen wurde.
Die PROCEND Anweisung führt jedoch auch Aktionen aus, die gewährleisten, daß die Variablen, auf die zugegriffen wurde, wieder die Variablen des aufrufenden Programmteiles sind und nicht mehr die lokalen Variablen der Prozedur.

6.2.10 CALL

Die CALL-Anweisung bewirkt, daß die Ausführung einer Prozedur angestoßen wird. Dabei wird der Name der auszuführenden Prozedur angegeben, gefolgt von ihren Argumenten (oder aktuellen Parametern) in Klammern.

Format:

```
CALL <Bezeichnung>
CALL <Bezeichnung>(<Liste mit Parametern>)
```

wobei <Bezeichnung> eine der in den PROC-Anweisungen des aktuellen Programms benutzten Bezeichnungen ist.

<Liste mit Parametern>, falls angegeben, ist eine Menge von arithmetischen und Stringausdrücken. Wenn dies einzelne Variable sind, werden sie so für die formalen Parameter eingesetzt, daß ihr Wert durch eine Zuweisung geändert werden kann. Wenn sie jedoch Konstanten oder komplexere Ausdrücke als einzelne Variablen sind, dann bewirkt die Zuweisung zum entsprechenden formalen Parameter einen Fehler.

Die aktuellen Parameter müssen den in der PROC-Anweisung angegebenen formalen Parametern in Anzahl und Typ entsprechen.

Eine Prozedur darf sich nicht selbst aufrufen; aber sie kann andere Prozeduren aufrufen.

Beispiel:

```
10   STRING (6) x$
.
.
.
200  CALL COPY(x$,"DATEI")
.
.
.
600  END
6000 PROC COPY(a$,b$)
6010 STRING (80) x$
6020 OPEN SI,a$
6030 OPEN SO,b$
6040 INPUT SI,x$
6050 PRINT SO,QUOTE,x$,QUOTE,CRLF
6060 IF x$  "" THEN 6040
6070 CLOSE SI
6080 CLOSE SO
6090 PROCEND
```

Dieses Beispiel zeigt eine Prozedur, die eine Datei kopiert. Der Name der zu kopierenden Datei und der Name der Kopie werden als Parameter übergeben (a$ und b$). Die durch a$ bezeichnete Datei muß eine Folge von Strings sein, die in Apostrophe eingeschlossen und durch Wagenrücklauf/Zeilenvorschub getrennt sind, und durch den Nullstring abgeschlossen sein.

Das Beispiel zeigt:

- zwei verschiedene Strings x$. Einer im Hauptprogramm, der den Dateinamen beinhaltet, und einen anderen in der Prozedur für die Aufnahme eines Strings mit einer Länge bis zu 80 Zeichen.

- eine Konstante ("DATEI"), die als Parameter an die Prozedur übergeben wird.

- Ein Beispiel für einen Aufruf, durch den eine neue Datei, genannt "DATEI", kreiert wird, und die denselben Inhalt hat wie die Datei, deren Name in x$ steht.

6.2.11 ENTER

Die ENTER-Anweisung ermöglicht es dem Programmierer, Assembler-Unterprogramme an BASIC anzuschliessen, die separat codiert und im Speicher abgelegt wurden. Sie müssen deshalb vorher vom Betriebssystem BS 610 an eine bekannte Adresse im Speicher geladen worden sein und der BASIC-Interpreter muß über ihre Lage informiert sein, damit er diesen Speicherbereich nicht überschreibt (siehe NEW-Kommando).

Der erste Befehl einer Assembler-Routine muß ein JMP-Befehl sein.

Format:

ENTER<Hex-Adresse>

wobei <Hex-Adresse> eine Speicheradresse in Hexadezimalschreibweise ist. Sie ist also eine Zahl mit vier "Ziffern", wobei jede "Ziffer" aus der Menge 0, 1, ...9, A,B,C,D,E,F stammt (A=10, B=11, ... F=15). Wichtig ist, daß <Hex-Adresse> mit einer Ziffer (0 bis 9) beginnt.

<Hex-Adresse> kann deshalb jede Stelle des 64 K großen Speichers adressieren.

Es gibt eine Reihe von Interpreter-Routinen, die vom Programmierer benutzt werden können (s. Anhang G). Nach Ausführung einer ENTER-Anweisung zeigt das Registerpaar DE des Mikroprozessors auf eine Tabelle mit Adressen, eine für jede dieser Routinen. Um eine dieser Routinen zu benutzen, muß der Programmierer seine eigene Routine mit ENTER aufrufen, die dann Unterprogrammsprünge im Maschinencode an die geeigneten, aus der Tabelle entnommenen Adressen macht.

Beispiel:

```
0550 ENTER 23FO
```

Die folgenden Seiten enthalten ein Beispiel einer Assembler-Routine. Die letzte Seite des Beispiels zeigt ein einfaches BASIC-Programm, das zeigt, wie diese Assembler-Routine benutzt werden kann.

Das BASIC-Programm verlangt die Eingabe einer Folge von Strings über ein Bildschirmfeld und gibt den Dezimalwert des ersten Zeichens jedes Strings aus, bis ein String eingegeben wird, der mit dem ASCII-Zeichen 32 beginnt.

Achtung: Die Zeile

```
0060 ENTER 2000
```

im BASIC-Programm und

```
org    2000h
```

die erste Zeile des Assembler-Programms, spezifizieren dieselbe hexadezimale Adresse 2000. Die ENTER-Anweisung muß eine Adresse angeben, die einen JMP-Befehl enthält.

Die hier beschriebene Assembler-Routine wird zusammen mit dem BASIC-System benutzt, um die Prozedur ASCI zu implementieren, die einen String und eine Dezimalzahl als Parameter hat. Die Routine konvertiert das erste Zeichen eines als nicht leer angenommenen Strings in das entsprechende ASCII-Äquivalent und speichert es in den dezimalen Parameter.

```
        org   2000h
asci:   jmp   next    ; muß mit einem JMP beginnen

;ASCI wird aufgerufen mit:
;hl Zeiger auf Parameterblock, der ein Block mit Zeigern
;   auf die Attribut-Vektoren (A.V.) der Prozedur-Parameter
;   ist
;de Zeiger auf Routinen-Liste, die ein Block mit Adressen
;   der benutzbaren BASIC-Routinen ist (s. Anhang G )

;a  Am Ende muß das Carry-Bit (cy) gelöscht sein, falls
;   keine Fehler aufgetreten sind (sonst a = Fehler-
;   schlüssel)

next    shld    pb        ; rette Zeiger auf Parameterblock
        xchg
        shld    routl     ; rette Zeiger auf Routinen-Liste
        lxi     h,0       ; setze hl auf 0. Parameter (String)
        call    avbrn     ; hole hl=0. Zeiger und rufe die
        db      2         ; 2. Routine auf (XAVDP)
                          ; de zeigt jetzt auf Stringbereich
        xchg
        call    branch    ; rufe GPTST auf, so daß de auf
        db      5         ; das 1. (Längenbyte) des Strings
                          ; zeigt
        xchg
        inx     h         ; hl zeigt auf 1. Datenbyte
        mov     e,m       ; hole 1. Zeichen des Strings
        mvi     d,0       ; erweitere de auf 16 Bits
        lxi     h,1       ; setze hl auf 1. Parameter (De-
                          ; zimal)
        call    avbrn     ; rufe CVID auf, um de in Dezimal-
                          ; zahl
        db      15        ; zu konvertieren
        ora     a         ; carry löschen, da kein Fehler
        ret               ; Ende

;AV - diese Routine bildet einen A.V.-Zeiger aus dem
;     Parameterblock
;     Eingabe: hl Nummer des Parameters (0,1,2,...)
;     Ausgabe: hl Zeiger auf entsprechenden A.V.

av:     push    d
        xchg
        lhld    pb
        dad     d
        dad     d         ; bilde Adresse des hl.ten Eintrags
        pop     d
        mov     a,m
        inx     h
        mov     h,m
        mov     l,a
        ret               ; hl := Inhalt des hl.ten Eintrags
```

```
;AVBRN - diese Routine kombiniert AV mit BRANCH

avbrn: call   av        ; bilde in hl Zeiger auf A.V.
                        ; dann gehe zu BRANCH

;BRANCH - diese Routine verzweigt auf eine Routine aus
;         der Routinenliste.
;         Aufruf:
;              call   branch
;              db     n      ; wobei n Nummer der Routine
;                            ; 0=TAVDV,1=XAVTP,...

branch:xthl
       mov    a,m      ; Parameter hinter Aufrufstelle
       inx    h        ; Rückkehradresse bilden
       xthl            ; Rückkehradresse in Stack
       push   h
       push   d
       mov    e,a
       mvi    d,0      ; de = Nummer der Routine
       lhld   routl    ; Zeiger auf Routinen-Liste
       dad    d
       dad    d        ; hl = Adresse der n.ten Routine
       mov    a,m
       inx    h
       mov    h,m
       mov    l,a      ; hl = Adresse der n.ten Routine
       pop    d
       xthl
       ret             ; "Rückkehr" in entsprechende
                       ;  Routine

pb:    dw     0        ; Zeiger auf Parameterblock
routl: dw     0        ; Zeiger auf Routinen-Liste

;das folgende RET dient nur dazu, um das Programm laden
;zu können. Nach dem Laden sofort wieder Ende

lad:   ret
       end    lad      ; Startadresse nach dem Laden
```

```
0010 DISPLAY s$,10,10,10,8
0020 CALL asci (NEX(s$),d)
0030 IF d=32 THEN 44
0040 PRINT ,d,CRLF
0042 GOTO 20
0044 END
0045 PROC asci(str$,dez)
0060 ENTER 2000
0070 PROCEND
```

6.2.12 STOP

Die STOP-Anweisung bewirkt, daß das Programm die Ausführung zeitweilig unterbricht und in den Stop-Modus geht. Eine Meldung, die anzeigt, daß dieser Fall eingetreten ist, wird ausgegeben, gefolgt von einer Aufforderung - dem Zeichen ">" - an den Benutzer, eine Eingabe zu machen.

Format:

```
STOP
```

Weitere Einzelheiten des Stop-Modus und seiner Wirkung sind aus Teil III zu ersehen.

6.2.13 EXIT

Die EXIT-Anweisung hat zwei verschiedene Wirkungen, abhängig davon, ob sie in einer Prozedur oder innerhalb des Hauptprogramms auftritt.

- In einer Prozedur: EXIT bewirkt die Beendigung der Prozedur in genau derselben Weise wie PROCEND. Es wird also der Programmteil, der die Prozedur aufgerufen hat, mit der Anweisung hinter dem CALL fortgesetzt.

- Im Hauptprogramm: EXIT bewirkt die vollständige Beendigung des Programms. Alle Dateien werden geschlossen und das System geht wieder in den Kommando-Modus. Es kann dann auf keine Variable des Programms mehr zugegriffen werden.

Format:

```
EXIT
```

6.2.14 END

Diese Anweisung muß die letzte Anweisung jedes BASIC-Hauptprogramms sein. Wenn sie ausgeführt wird, wird das Programm vollständig beendet. Alle Dateien werden geschlossen und das System geht wieder in den Kommando-Modus. Nach Ausführung der END-Anweisung kann auf keine Variable des Programms mehr zugegriffen werden.

Alle Prozeduren (PROC) eines BASIC-Programms müssen nach der Zeile mit der END-Anweisung aufgeführt werden, alle Unterprogramme (GOSUB) davor.

Format:

```
END
```

6.2.15 START

Die START-Anweisung ermöglicht es, vom Programm aus eine neue Kommandodatei zu eröffnen. Existiert die Datei, so wird das laufende Programm wie durch eine END-Anweisung abgeschlossen. Die Kommando-Datei wird anschliessend so abgearbeitet, wie es beim START-Kommando beschrieben wird (s.III,9.1)

Format:

START< Dateiname>

Beispiel:

0150 START KOMDAT

6.3 Ein/Ausgabe-Anweisungen

Die Ein/Ausgabe-Anweisungen werden benutzt, um an Variable Werte aus externen Quellen zuzuweisen und externen Zielen die aktuellen Werte von Programmvariablen zu übergeben.

6.3.1 OPEN

Die OPEN-Anweisung ordnet einer logischen Dateibezeichnung eine echte Datei oder ein Hardwaregerät zu. Wenn dann diese logische Dateibezeichnung darauffolgend in INPUT- oder PRINT-Anweisungen benutzt wird, bewirkt sie die Eingabe von, bzw. die Ausgabe zu der in der OPEN-Anweisung angegebenen Datei oder dem entsprechenden Gerät. Wenn eine logische Dateibezeichnung in einer anderen als der OPEN-Anweisung verwendet wird, bezieht sie sich immer auf die als letztes mit dieser Dateibezeichnung eröffnete Datei oder das entsprechende Gerät.

Format:

OPEN<log. Dateibezeichnung>,<Dateiname>

wobei <log. Dateibezeichnung> eine der folgenden sein muß:

- SI Standard Input (Normaleingabe)
- SO Standard Output (Normalausgabe)
- SL Standard List (Normaldruck)

- AI Auxiliary Input (Zusatzeingabe)

- AO Auxiliary Output (Zusatzausgabe)

- AL Auxiliary List (Zusatzdruck)

<Dateiname> ist ein Stringausdruck, der einen Dateinamen spezifiziert.

An SI und AI müssen Eingabegeräte oder -dateien zugewiesen werden, die bereits im System existieren. SO, AO, SL und AL müssen sich auf Ausgabegeräte oder -dateien beziehen, wobei Dateien im INTEL-Format noch nicht existieren dürfen und bei der Ausführung der OPEN-Anweisung angelegt werden.

Beispiele:

```
0400 OPEN SI,"EINDAT"
1010 OPEN SO,:LP:
```

Zwei OPEN-Anweisungen, die sich auf dieselbe logische Dateibezeichnung beziehen, dürfen nicht ohne dazwischenliegende CLOSE-Anweisung für diese logische Dateibezeichnung auftreten.

6.3.2 CLOSE

Die CLOSE-Anweisung hebt die Zuordnung einer aktuellen Datei oder eines Geräts zu einer logischen Dateibezeichnung auf. Wenn der angegebenen logischen Dateibezeichnung eine aktuelle Datei zugeordnet ist, wird diese Datei geschlossen und steht für die weitere Benutzung nicht mehr zur Verfügung bis sie wieder in einer neuen OPEN-Anweisung verwendet wird.

Format:

CLOSE<log. Dateibezeichnung>

wobei <log. Dateibezeichnung> eine der folgenden sein muß:

- SI Standard Input

- SO Standard Output

- SL Standard List

- AI Auxiliary Input

- AO Auxiliary Output

- AL Auxiliary List

Beispiel:

```
1800 CLOSE SI
```

6.3.3 INPUT

Die INPUT-Anweisung ermöglicht es einem Programm, Daten von einer externen Quelle zu empgangen, die entweder ein Hardwaregerät ode eine Datei sein kann. Die verfügbaren Hardwaregeräte sind die Tastatur und das UART (Universal Asynchronous Receive-Transmit-Gerät).

Format:

INPUT <log. Dateibezeichnung><Eingabeliste>

wobei <log. Dateibezeichnung> eine der folgenden sein muß:

- SI Standard Input, spezifiziert eine Datei/ein Gerät
- AI Auxiliary Input, spezifiziert eine Datei/ ein Gerät
- :CI: spezifiziert die Tastatur
- :RC: spezifiziert Eingabe vom UART
- fehlt die log. Dateibezeichnung , so wird wie bei :CI: die Tastatur angesprochen.

<Eingabeliste> ist eine Liste von Variablen (die indiziert sein können) und Aufrufen von Eingaberoutinen, die jeweils durch Komma getrennt sind. Die Daten werden der Reihe nach von der angegebenen Quelle in die einzelnen Variablen eingelesen.

Es gibt zwei Eingaberoutinen (SKIP und FIXED), die es dem Benutzer erlauben, die Eingabedaten zu manipulieren (s. Abschnitt 8).

Falls SI oder AI benutzt werden, müssen sie vorher in einer OPEN-Anweisung aufgetaucht sein. Es kann entweder eine bereits existierende Datei oder ein Hardware-Eingabegerät spezifizieren.

Die Elemente der Eingabeliste werden der Reihe nach behandelt. Falls eine Eingaberoutine aufgerufen wird, wird die entsprechende Aktion ausgeführt (s. Abschnitt 8). Falls das Element eine Variable ist, wird eine Eingabe des entsprechenden Typs erwartet.

- Stringvariable: Es wird ein String erwartet, der in Apostrophe (") eingeschlossen sein muß.

- Dezimalvariable: Es wird eine Zahl erwartet.

DIE EINGABEDATEN MÜSSEN IN SÄTZE AUFGETEILT SEIN, DIE NICHT LÄNGER ALS 80 ZEICHEN LANG SIND. SÄTZE WERDEN VON DER ZEICHENFOLGE Wagenrücklauf/Zeilenvorschub ABGESCHLOSSEN. STRINGVARIABLE MIT MEHR ALS 78 ZEICHEN KÖNNEN ALSO NICHT DIREKT EINGEGEBEN WERDEN.

6.3.4 INPUTC

Format und Syntax der INPUTC-Anweisung entsprechen der Beschreibung von INPUT.
Der Unterschied zur INPUT-Anweisung liegt darin, daß

- beim Lesen des ersten Satzes nicht INPUTC sondern INPUT benutzt werden muß

- INPUTC solange aus dem Dateipuffer liest, bis dieser leer ist. In diesem Fall liest INPUTC den nächsten Satz ein. Bei der Verwendung der INPUT-Anweisung wird bei jedem Aufruf der Puffer neu eingelesen.

6.3.4 INCHAR

Die INCHAR-Anweisung dient dazu, byte-weise von einem Gerät oder einer Datei einzulesen, ohne Steuerzeichen auszuwerten.

Format:

INCHAR<log.Dateibezeichnung>,<Eingabeliste>

<Log.Dateibezeichnung> entspricht der Beschreibung unter INPUT, <Eingabeliste> muß eine Liste von Stringvariablen sein.

Die Anzahl der Bytes, die eingelesen wird, entspricht der Länge der Stringvariable(n) in der Liste. Wird das Dateiende gefunden, so werden die Variablen nur bis zu diesem Ende eingelesen.

6.3.5 PRINT

Die PRINT-Anweisung ermöglicht es einem Programm, Daten an externe Ziele zu senden. Diese können entweder ein Hardwaregerät oder eine Datei sein. Die verfügbaren Hardwaregeräte sind das UART, der Drucker und der Bildschirm.

Format:

PRINT<log. Dateibezeichnung>,<Ausgabeliste>

wobei<log. Dateibezeichnung> entweder ein Hardwaregerät oder eine Datei bezeichnet.

<Ausgabeliste> ist eine Liste von durch Komma getrennten Ausdrücken, deren Werte an die angegebene Datei/ das Gerät ausgegeben werden. Die Ausdrücke können arithmetische Ausdrücke, Stringausdrücke oder Ausgabefunktionen sein, die Steuerzeichen für die Ausgabe auf den Bildschirm ausgeben.

Die folgenden logischen Dateibezeichnungen sind für die Benutzung in PRINT-Anweisungen verfügbar:

- SO Standard Output, spezifiziert eine Datei/ ein Gerät
- AO Auxiliary Output, spezifiziert eine Datei/ ein Gerät
- SL Standard List, spezifiziert eine Datei/ ein Gerät
- AL Auxiliary List, spezifiziert eine Datei/ ein Gerät
- :CO: spezifiziert den Bildschirm
- :LP: spezifiziert den Drucker
- :XM: spezifiziert Ausgabe über das UART
- fehlt die log. Dateibezeichnung, so wird wie bei:CO: der Bildschirm angesprochen.

Falls SO, AO, SL oder AL benutzt werden, so müssen sie vorher in einer OPEN-Anweisung eröffnet worden sein. Es kann entweder eine Datei, die beschrieben werden soll, oder eines der Hardware-Ausgabegeräte spezifiziert werden. Die Ausdrücke in der Liste werden der Reihe nach behandelt. Es wird entweder eine Dezimalzahl mit vorangehendem Blank, ein String oder eine Folge von Steuerzeichen für das Gerät ausgegeben.

Beispiel:

```
0025 PRINT ,"NO",12
```

gibt "NO 12" auf den Bildschirm aus.

6.4 Andere Anweisungen

Es gibt noch zwei weitere ausführbare Anweisungen, die bis jetzt noch nicht beschrieben worden sind. Dies sind:

- PRECISION: wird benutzt, um die Genauigkeit zu definieren, also die Anzahl der Nachkommastellen von Dezimalzahlen.
- DISPLAY: wird benutzt, um Bildschirmfelder zu definieren und sie mit Stringvariablen zu verbinden. Wenn der Variablen ein Wert zugewiesen wird, dann wird dieser Wert im Feld angezeigt.

6.4.1 PRECISION

Dezimalvariable können sowohl Dezimalzahlen (also Zahlen mit Stellen hinter dem Komma) als auch ganze Zahlen aufnehmen. Das wird dadurch erreicht, daß für jede Variable angegeben wird, wieviele der gespeicherten Stellen als Dezimalstellen behandelt werden sollen. Die PRECISION-Anweisung wird benutzt, um diesen Wert für bestimmte Variable zu spezifizieren oder zu ändern.

Format:

PRECISION <Ausdruck>,<Variablenliste>

wobei <Ausdruck> ein arithmetischer Ausdruck ist, dessen Wert gerundet wird, falls er nicht ganzzahlig ist.

<Variablenliste> ist eine Menge von durch Komma getrennten Bezeichnungen für Dezimalvariable.

Sobald die Anweisung ausgeführt wird, werden alle Variablen in der Liste mit der durch den Ausdruck festgelegten Anzahl von Dezimalstellen verarbeitet.

Ein einzelnes Element eines Feldes kann nicht spezifiziert werden.

Ein ganzes Feld kann nur dann spezifiziert werden, falls noch keinem der Elemente ein Wert zugewiesen wurde.

Der Wert des Ausdrucks darf die Anzahl der für Dezimalzahlen gespeicherten Stellen (SIZE) nicht überschreiten.

Beispiele:

```
0010 PRECISION 3,A,B,C
0020 LET A = 3.142
0030 PRECISION 2,A
0040 PRINT ,A
```

gibt 3.14 aus.

6.4.2 DISPLAY

Die DISPLAY-Anweisung verbindet eine Stringvariable mit einem Bildschirmfeld. Sobald einer in einer DISPALY-Anweisung verwendeten Variablen ein neuer Wert zugewiesen wird, wird dieser Wert im zugeordneten Bildschirmfeld angezeigt. Ein neuer Wert für diese Variable kann eingegeben werden, indem sie als Parameter beim Aufruf der NEW-Funktion verwendet wird.

Format:

```
DISPLAY <Bezeichnung>,<Ausdruck>,<Ausdruck>,
        <Ausdruck>,<Ausdruck>
```

wobei <Bezeichnung> die Bezeichnung einer Stringvariablen sein muß.
Alle vier Ausdrücke sind arithmetische Ausdrücke, deren Werte gerundet werden, falls sie nicht ganzzahlig sind.
Die ersten zwei Ausdrücke legen Zeile und Spalte (1-25, 1-80) des Beginns des Bildschirmfeldes fest.
Der dritte Ausdruck legt die Länge des Feldes fest, das sich über Zeilengrenzen hinaus erstrecken kann.
Die maximale Länge eines Bildschirmfeldes ist 255.
Der vierte Ausdruck wird benutzt, um den Typ der Daten zu spezifizieren, die in diese Variable beim Aufruf der NEW-Funktion eingegeben werden können.

Es sind fünf Typen verfügbar:

- 1: Dezimalzahl (Ziffern,.,+,-)
- 2: Buchstaben (A ... Z, a ... z, Leerzeichen)
- 4: Ziffern (0 ... 9)
- 8: alphanumerische Zeichen (d. h. alle Zeichen)
- 16: automatischer Carriage Return, d.h. bei Dateneingabe mit der NEW-Funktion muß nicht mit der CR-Taste abgeschlossen werden, wenn das Feld vollständig mit Zeichen gefüllt ist

Diese Schlüssel können bei Bedarf (durch Addition) kombiniert werden, um zusätzliche Überprüfungen zu ermöglichen (falls z.B. ein Typ von 6 angegeben ist, können die Eingabedaten nur Buchstaben oder Ziffern sein). Bei der Ausführung der NEW-Funktion können nur Zeichen des angegebenen Typs in das Bildschirmfeld eingegeben werden. Bildschirmfelder können sich beliebig überlappen. Wenn das der Fall ist, werden die Daten der DISPLAY-Variablen angezeigt, auf die zuletzt zugegriffen wurde.

Beispiele:

```
0110 DISPLAY A$,1,1,10,8
0120 DISPLAY B$,1,4,10,8
0130 LET A$ = "STRING1"
0140 LET B$ = "STRING2"
```

zeigt an:

- Zuerst "STRING1 " in der linken oberen Ecke des Bildschirms.

- Sobald die LET B$-Anweisung ausgeführt ist, wird "STRSTRING2 " angezeigt.

Wenn einer Variablen bereits ein Wert zugewiesen ist, bewirkt die DISPLAY-Anweisung, daß dieser Wert angezeigt wird.

7 Standardfunktionen

Die Standardfunktionen sind vordefinierte Systemroutinen, die allgemein benötigte Hilfsmittel bereitstellen (wie z.B. ABS) oder Hilfsmittel, die sonst außerhalb des Bereichs eines BASIC-Programms lägen (wie die Behandlung des Bildschirms). Jede Funktion führt eine festgelegte Prozedur aus und gibt ein Ergebnis zurück, wobei die übergebenen Parameter benutzt werden. Das Ergebnis kann eine Dezimalzahl oder ein String sein, abhängig von der Funktionsdefinition.

7.1 Zahlfunktionen

Die Zahlfunktionen führen Berechnungen auf Zahlen aus und übergeben anschließend eine Zahl. Sie benötigen alle einen arithmetischen Ausdruck als Parameter und haben eine Dezimalzahl als Ergebnis.

7.1.1 ABS

Die ABS-Funktion bildet den Absolutbetrag des Parameters, d. h. falls der Ausdruck negativ ist, wird das Vorzeichen umgedreht, sonst bleibt der Wert unverändert.

Beispiele:

```
ABS(3.5) ist 3.5
ABS(-4)  ist   4
ABS(0)   ist   0
```

7.1.2 INT

Die INT-Funktion übergibt einen Wert, der die größte ganze Zahl ist, die kleiner als oder gleich dem angegebene Parameter ist. Für positive Parameter heißt das, daß das Ergebnis der ganzzahlige Anteil des Ausdrucks, bzw. - falls der Ausdruck bereits ganzzahlig ist- gleich dem Ausdruck ist.

Beispiele:

```
INT(3.5)   = 3
INT(-3.5)  = -4
INT(0.1)   = 0
INT(0)     =
INT(-0.1)  = -1
INT(-3)    = -3
```

7.1,3 SGN

Die SGN-Funktion hat drei verschiedene Ergebniswerte, die das Vorzeichen des angegebenen Ausdrucks anzeigen.

```
 1  : Der Parameter ist größer als 0

-1  : Der Parameter ist kleiner als 0

 0  : Der Parameter ist gleich 0
```

Beispiele:

```
SGN(-3.5) = -1
SGN(3.57) =  1
SGN(0)    =  0
```

7.2 Stringverarbeitungsfunktionen

Die Stringverarbeitungsfunktionen führen Berechnungen auf Zahlen oder Strings aus und haben Strings oder Zahlen als Ergebnis. Anzahl und Typ der Parameter sind deshalb bei den einzelnen Funktionen unterschiedlich. Sie können einfach in zwei Gruppen geteilt werden:

- Jene, die einen String als Ergebnis haben:
  - STR
  - SUB
  - JNC
  - MSK
- Jene, die eine Dezimalzahl als Ergebnis haben:
  - LEN
  - VAL
  - IDX

7.2,1 STR

Die STR-Funktion benötigt als Parameter einen arithmetischen Ausdruck und hat eine String als Ergebnis. Das Stringergebnis ist das Stringäquivalent des Ausdrucks. Es enthält daher die Zeichen "0" bis "9" und eventuell "." und "-", aber nie die Zeichen "+" und das Leerzeichen.

Beispiele:

```
STR(10.2) = "10.2"
STR(-3)   =  "-3"
STR(0)    =   "0"
STR(0.00) =   "0"
STR(3.00) =   "3"
```

7.2.2 SUB

Die SUB-Funktion benötigt 3 Parameter:

- einen String S

- zwei arithmetische Ausdrücke n1, n2

Sie gibt einen String als Ergebnis zurück.

Die Funktion bildet einen Unterstring von S, der beim n1-ten Zeichen beginnt und die Länge n2 hat. Falls n1 größer ist als die Länge von S, ist das Ergebnis der Nullstring. Falls n1 + n2 größer ist als die Länge von S, ist das Ergebnis der String vom n1-ten Zeichen an bis zum Ende von S.

Beispiele:

```
SUB("Dies ist ein String",6,14) ist "ist ein String"
SUB("String",3,10)              ist "ring"
SUB("String",20,8)              ist ""
```

7.2.3 JNC

Die JNC-Funktion benötigt zwei Parameter, die beide Strings sein müssen. Sie gibt als Ergebnis einen String zurück, der durch die Verkettung der beiden Parameter entsteht. Die Länge des Ergebnisses ist daher die Summe der Länge der beiden Ausgangsstrings. Das Ergebnis darf jedoch nie länger als 255 Stellen lang sein.

Beispiele:

```
JNC ("P","ROBE")      ist  "PROBE"
JNC (STR(3),STR(10))  ist  "310"
```

7.2.4 MSK

Die MSK-Funktion benötigt zwei Parameter. Der erste ist eine Maske. Sie muß ein String sein und darf nur die unten aufgeführten Zeichen enthalten. Der zweite Parameter muß ein arithmetischer Ausdruck sein. Die Zahl wird entsprechend den durch die Maske festgelegten Regeln zum Druck aufbereitet und das Ergebnis als String zurückgegeben.

Folgende Zeichen sind in der Maske erlaubt und haben die beschriebene Wirkung:

Z	An dieser Position sollen in der Zahl führende Nullen unterdrückt werden. Vor Z darf kein Dezimalpunkt, keine 9 und kein CR stehen.
9	An dieser Position soll eine Ziffer erscheinen. Vor 9 darf jedes Zeichen außer CR stehen.
Leerraum	An dieser Stelle soll immer ein Leerraum stehen.
,	An dieser Stelle soll ein Komma stehen. Wenn vor dem Komma - oder Z steht und Nullunterdrückung erfolgte, so wird das Komma ebenfalls unterdrückt.
.	An dieser Stelle soll ein Dezimalpunkt stehen. Der aufzubereitende Ausdruck wird auf die richtige Anzahl von Dezimalstellen gerundet. Vor einem Dezimalpunkt dürfen Z, 9, Leerraum, Komma und - stehen. Es darf nur ein Dezimalpunkt vorhanden sein.
-	Es zeigt die Position des Vorzeichens an. Es kann am Anfang oder am Ende der Maske benutzt werden. Es darf nicht zusammen mit CR benutzt werden. Falls mehr als ein Minuszeichen am Anfang steht, handelt es sich um eine "wandernde" Vorzeichenposition. Das Vorzeichen steht dann entweder vor der ersten signifikanten Ziffer oder auf der Position des letzten Minuszeichens, je nachdem was zuerst auftritt. Falls der Ausdruck negativ ist, wird ein Minuszeichen ausgegeben: falls er positiv ist, entfällt das Vorzeichen.
CR	("CR" = Credit, nicht Carriage Return, also Wagenrücklauf). Es legt zwei Vorzeichenpositionen am Ende der Maske fest. Falls der Ausdruck negativ ist, werden die Zeichen CR eingesetzt, falls nicht, zwei Leerräume.

Beispiele:

```
MSK("ZZ9.99",3)     = "  3.00"
MSK("ZZ9.99",0.75)  = "  0.75"
MSK("ZZ9.99",0.5)   = "  0.50"
```

7.2.5 LEN

Die LEN-Funktion benötigt eine String als Parameter und hat eine Dezimalzahl als Ergebnis, die die Länge des Strings angibt.

Beispiele:

```
LEN("STRING")       = 6
LEN("")             = 0
LEN(JNC("3","10"))  = 3
```

7.2.6 VAL

Die VAL-Funktion benötigt einen String als Parameter und hat eine Dezimalzahl als Ergebnis. Das Ergebnis ist die zum String äquivalente Dezimalzahl. Der String muß eine gültige Dezimalzahl darstellen.

Beispiele:

```
VAL("10.2")   = 10.2
VAL("-3")     =  -3
VAL("")       ist unzulässig
VAL("A")      ist unzulässig
VAL("3.2E7")  ist unzulässig

0100 LET a = VAL(NEW(a$))
```

weist a einen Wert zu, der in das der Variablen a$ zugeordnete Bildschirmfeld eingegeben wird.

7.2.7 IDX

Die IDX-Funktion benötigt zwei Strings als Parameter und hat eine Dezimalzahl als Ergebnis. Das Ergebnis stellt die Zeichenposition des erstmaligen Auftretens des zweiten Strings im ersten String dar. Falls der zweite String nicht im ersten String vorkommt oder der erste String der Nullstring ist, so ist das Ergebnis Null.

Beispiele:

```
IDX("DAS IST EIN STRING","STRING") = 13
IDX("DAS IST EIN STRING","S")      =  3
IDX("DAS IST EIN STRING","B")      =  0
IDX("","C")                        =  0
```

7.3 Ausgabefunktionen

Die Ausgabefunktionen sind eine Gruppe von Funktionen, die normalerweise direkt in Ausgabeanweisungen (PRINT) benutzt werden, die aber alle einen String als Ergebnis haben und deshalb auch überall dort benutzt werden können, wo ein Stringwert erlaubt ist.

CRLF	ergibt als Ergebnis die zwei Zeichen, die den Wagenrücklauf/Zeilenvorschub darstellen.
QUOTE	ergibt als Ergebnis das Zeichen Apostroph (").
TAB	ergibt das ASCII-Zeichen für den Horizontaltabulator.
BIN	ergibt als Ergebnis das ASCII-Zeichen, dessen Dezimaläquivalent als Parameter übergeben wurde.

7.3.1 CRLF

Die CRLF-Funktion benötigt keine Parameter und hat immer das gleiche Ergebnis. Das Ergebnis sind die zwei Steuerzeichen, die den Wagenrücklauf/Zeilenvorschub darstellen.

7.3.2 QUOTE

Die QUOTE-Funktion benötigt keine Parameter und hat ein konstantes Ergebnis. Dies ist das ASCII-Zeichen Apostroph (").

Beispiel:

```
0100 PRINT SO,QUOTE,A$,QUOTE,CRLF
```

gibt A$ in einer Form auf eine Datei aus, die mit der INPUT-Anweisung wieder gelesen werden kann.

7.3.3 TAB

Die TAB-Funktion benötigt keine Parameter und hat ein konstantes Ergebnis. Das Ergebnis ist das ASCII-Zeichen für den Horizontaltabulator. Die Wirkung des Zeichens ist hardwareabhängig. Am Bildschirm zeigt es überhaupt keine Wirkung. Bei manchen Druckern bewirkt es die Tabulation auf die nächste 10-Zeichen-Position, bei anderen auf die nächste eingestellte Tabulatorposition.

7.3.4 BIN

Die BIN-Funktion benötigt einen Parameter, der eine Dezimalzahl sein muß. Das Ergebnis ist ein einstelliger String, dessen dezimales ASCII-Äquivalent als Parameter übergeben wurde.

Beispiele:

```
BIN (97) ist "a"
BIN (65) ist "A"
BIN (7)  erzeugt ein akustisches Signal
BIN (13) ist Carriage Return
```

7.4 Funktionen für die Bildschirmausgabe

Diese Bildschirmfunktionen können überall dort benutzt werden, wo eine Funktion erlaubt ist, die einen String als Ergebnis hat. Sie lösen jedoch spezielle Aktionen bei der Ausgabe auf den Bildschirm aus und ihre Wirkung ist undefiniert, falls die Ausgabe auf eine Datei oder ein anderes Hardwaregerät als den Bildschirm erfolgt. Die verfügbaren Funktionen sind:

LOC erzeugt eine Folge von Steuerzeichen für die Positionierung des Cursors auf dem Bildschirm.

ATT erzeugt ein Attributzeichen, das die Darstellungsweise des Bildschirms beeinflußt.

UND erzeugt das Attribut für den Unterstreichungsmodus.

NOR erzeugt das Attribut für den Normalmodus.

CLR erzeugt das Steuerzeichen, das den Bildschirm löscht.

Falls etwa NOR-Zeichen an das Ende jeder Zeile geschrieben werden, z. B. mit

```
0010 FOR i = 1 TO 25
0020 PRINT ,LOC(i,80),NOR
0030 NEXT i
```

dann können Attribute an den Anfang einzelner Zeilen gesetzt werden, die nur diese Zeile beeinflussen.

```
0100 PRINT ,LOC(12,1),ATT(48)
```

läßt z. B. die Zeile 12 blinken.

```
0200 PRINT ,LOC(7,1),ATT(97)
```

macht z. B. die Zeile 7 unsichtbar (die Zeichen werden gespeichert, aber nicht angezeigt).

7.4.1 LOC

Die LOC-Funktion benötigt zwei Dezimalzahlen als Parameter. Das Ergebnis ist eine Folge von Steuerzeichen, die den Cursor auf die X-Y Position setzen, die von den beiden Parameteren festgelegt wird.

Der erste Parameter muß eine Zahl zwischen 1 und 25 sein und stellt die Nummer der Bildschirmzeile dar. Der zweite Parameter muß eine Zahl zwischen 1 und 80 sein und stellt die Zeichenposition oder Spalte innerhalb einer Zeile dar.

Beispiel:

```
0100 PRINT,LOC(25,75),"WORT"
```

setzt "WORT" in die rechte untere Ecke des Bildschirms.

7.4.2 ATT

Die ATT-Funktion ermöglicht es dem Programmierer, den Darstellungsmodus des Bildschirms zu ändern. Die Wirkung hält bis zum nächsten dargestellten Attributzeichen an.

Die ATT-Funktion benötigt einen arithmetischen Ausdruck als Parameter. Es gibt sechs Werte mit Bedeutung:

- 65 inverse Darstellung
- 48 blinkende Darstellung
- 32 Darstellung mit halber Intensität
- 81 unterstrichene Darstellung
- 97 unsichtbare Darstellung
- 113 Normale Darstellung

Die Ausführung von CLR löscht sämtliche Attribute am Bildschirm und setzt ihn in Normalmodus.

Die ATT-Funktion hat einen gültigen String als Ergebnis und darf überall dort benutzt werden, wo ein solcher String erlaubt ist.

Beispiel:

```
0100 PRINT ,CLR,ATT(65),LOC(13,1),NOR
```

invertiert die obere Hälfte des Bildschirms (macht sie also gelb bzw. grün) und läßt die untere Hälfte normal, also schwarz.

7.4.3 NOR

Die NOR-Funktion benötigt keine Parameter und liefert als Ergebnis das Steuerzeichen, das den Bildschirm in Normalmodus setzt.

7.4.4 UND

Die UND-Funktion benötigt keine Parameter und liefert als Ergebnis das Steuerzeichen, das den Bildschirm in Unterstreichungsmodus setzt. Alle Zeichen zwischen diesem und dem nächsten Attribut werden unterstrichen dargestellt.

7.4.5 CLR

Die CLR-Funktion benötigt keine Parameter und hat ein konstantes Ergebnis. Die Konstante ist das Steuerzeichen, das den Bildschirm löscht.

7.5 Sonstige Funktionen

7.5.1 ASCII

Die ASCII-Funktion benötigt einen String-Parameter und gibt einen dezimalen Wert als Ergebnis zurück. Diese Zahl ist die dezimale Darstellung des ersten Zeichens im String.

Beispiele:

```
ASCII("STRING") ist 83
ASCII("60")     ist 54
```

7.5.2 NEW

Display-Variable sind solche Variable, die in DISPLAY-Anweisungen im Programm spezifiziert wurden. Jeder Variablen ist ein Bildschirmfeld zugeordnet durch die zuletzt ausgeführte DISPLAY-Anweisung für diese Variable. Ein neuer Wert wird angezeigt, wenn

- eine Zuweisung mit der Variablen auf der linken Seite der Zuweisung ausgeführt wird.
- ein Aufruf der NEW-Funktion mit der DISPLAY-Variablen als Parameter ausgeführt wird.
- eine neue DISPLAY-Anweisung für eine Display-Variable ausgeführt wird, die den betreffenden Bildschirmbereich berührt.

Die NEW-Funktion verlangt die Eingabe eines neuen Wertes in das Bildschirmfeld. Wenn der Aufruf der NEW-Funktion ausgeführt wird, wird das Bildschirmfeld mit Blockzeichen gefüllt (das Feld wird also invertiert dargestellt), der Cursor auf den Beginn des Feldes positioniert und die Eingabe angefordert. Jedes Zeichen wird bei der Eingabe überprüft und muß dem Typ entsprechen, der in der DISPLAY-Anweisung festgelegt wurde. Wenn die Eingabe in das Feld vollständig ist (dies wird am Wagenrücklauf CR erkannt), wird der Wert im Bildschirmfeld dem Funktionsparameter zugewiesen und auch als Ergebnis des Funktionsaufrufs zurückgebeben.

Beispiel:

```
0100 LET A$ = NEW(B$)
0200 LET x  = VAL(NEW(x$))
```

8 Standardeingabefunktionen

Die Standardeingabefunktionen ermöglichen eine zeichenweise Steuerung der Eingabe. Sie dürfen nur in INPUT-Anweisungen benutzt werden. Es gibt zwei Eingabefunktionen:

SKIP bewirkt, daß die nächsten n Zeichen ignoriert werden.

FIXED bewirkt, daß die nächsten n Zeichen an eine Stringvariable übergeben werden.

8.1 SKIP

Die SKIP-Funktion benötigt einen Parameter. Dieser legt die Anzahl der Zeichen fest, die bei der Eingabe von der in dieser INPUT-Anweisung spezifizierten Datei/dem Gerät ignoriert werden.

Beispiel:

```
0100 INPUT ,B$,SKIP(10),C$
```

Diese Anweisung liest zwei Strings von der Tastatur ein und ignoriert 10 Zeichen, die der Benutzer zwischen diesen beiden Strings eingibt, z. B.

```
"STRING 1"ABCDEFGHIK"STRING 2"
```

als Eingabe liefert

```
B$ = "STRING 1"
C$ = "STRING 2"
```

8.2 FIXED

Die FIXED-Funktion benötigt zwei Parameter. Der erste muß eine Stringvariable sein, der zweite eine Zahl. Der Stringvariablen werden als Wert die nächsten n Zeichen des Eingabestromes zugewiesen, wobei n die durch den zweiten Parameter festgelegte Zahl ist. Falls das erste Zeichen ein Apostroph (") ist, wird der in Apostrophe eingeschlossene String (unabhängig von seiner Länge) an die angegebene Variable zugewiesen.

Beispiel:

```
0100 INPUT ,FIXED(B$,10)
```

Falls über die Tastatur

```
DAS IST EINE EINGABE
```

eingegeben wird, enthält B$ "DAS IST EI".
Falls jedoch die Eingabe

```
"DAS IST EINE EINGABE"
```

lautet, wird an B$ der String

```
"DAS IST EINE EINGABE"
```

zugewiesen.

Falls die Anzahl der Zeichen im Eingabestrom kleiner als n ist, dann werden der Stringvariablen nur die verfügbaren Zeichen zugewiesen.

TEIL III: BENUTZUNG DES SYSTEMS

1. Einführung

Sobald das Betriebssystem geladen ist, kann das BASIC-System durch die Eingabe des Programmnamens 'BASIC' gestartet werden, eventuell mit vorangestellter Gerätespezifikation (s. Abschnitt 2).

Am Anfang (d. h. sofort nach dem Laden) befindet sich das BASIC-System im Kommando-Modus und erwartet die Eingabe von Systemkommandos oder Quellzeilen von der Tastatur. Sobald ein Programm geladen ist, entweder durch Eingeben über die Tastatur oder durch Laden aus einer Datei auf der Floppy Disk , kann es durch das RUN-Kommando gestartet werden. Während der Ausführung eines Programms ist das System im Run-Modus.
Falls ein Fehler auftritt oder eine STOP-Anweisung ausgeführt wird, wird die Programmausführung unterbrochen und das System geht in den Stop-Modus. Eine entsprechende Meldung wird am Bildschirm ausgegeben, die den Grund für die Unterbrechung angibt.

Während das System im Stop-Modus ist, kann das Programm mit Hilfe von bestimmten Systemkommandos oder BASIC-Quellanweisungen getestet werden. Das Programm selbst kann nicht verändert werden, bis der Kommando-Modus wieder eingestellt ist. Dies wird mit Hilfe des ABORT-Kommandos erreicht. Anstelle der Benutzung des ABORT-Kommandos kann die Programmausführung mit dem CONTINUE-Kommando an der Unterbrechungsstelle wieder aufgenommen werden.

Die Ausführung des Programms kann unterbrochen werden durch Eingabe des CTRL/C-Zeichens, es sei denn, das Programm erwartet eine Eingabe. Dies bewirkt die Unterbrechung des Programms; das System geht in den Stop-Modus.

Wenn das System im Stop- oder Kommando-Modus ist, gibt es das Zeichen ">" aus, um den Benutzer mitzuteilen, daß es auf eine Eingabe über die Tastatur wartet.

Die Systemkommandos und ihre Wirkung werden im einzelnen im Rest des Teils III dieses Handbuchs beschrieben.

2 Beginn und Ende eines Dialogs mit dem BASIC-System

Bevor das BASIC-System gestartet werden kann, muß es geladen werden. Zuerst muß die gesamte Hardware eingeschaltet werden, also der Bildschirm und die zu benutzenden Floppy Disk- und Kassettenlaufwerke. Das System gibt die Meldung "WAITING" auf den Bildschirm aus, und zeigt damit an, daß es darauf wartet, daß die System-Floppy Disk geladen wird. Sobald dies erfolgt ist, wird automatisch das Betriebssystem BS 610 geladen und die Meldung

"BS610 VERn.m"

ausgegeben. Nun kann BASIC gestartet werden.

Falls sich das BASIC-System nicht auf der System-Floppy Disk befindet, muß die entsprechende Floppy Disk in das Laufwerk eingeschoben werden. Das BASIC-System wird geladen, indem über die Tastatur eingegeben wird:

BASIC falls sich BASIC im Laufwerk 0 befindet

:fn:BASIC falls sich BASIC im Laufwerk 'n' befindet

(Das Kommando "BASIC" kann auch als "basic" geschrieben werden).

Es wird die Meldung

"SIEMENS COMMERCIAL BASIC VERi.j"

am Schirm ausgegeben. Das System befindet sich nun im Kommando-Modus und erwartet die Eingabe von Kommandos oder Quellzeilen.

Wenn der Benutzer den Dialog mit dem BASIC-System beenden will, kann er dies mit dem BYE-Kommando. Er befindet sich dann wieder im Betriebssystem BS 610.

2.1 BYE

Das BYE-Kommando ermöglicht es, den Dialog mit dem BASIC-System zu beenden und in das Betriebssystem BS 610 zurückzukehren. Nach Ausführung des BYE-Kommandos meldet sich wieder das Betreibssystem mit

```
"BS610 VERn.m"
```

Das BYE-Kommando kann nur im Kommando-Modus benutzt werden.

Format:

```
BYE
```

3 Laden eines BASIC-Programms

Sobald das BASIC-System gestartet ist, befindet es sich im Kommando-Modus und erwartet eine Eingabe. Zu diesem Zeitpunkt ist kein Programm geladen. Bevor also irgendwelche weiteren Aktionen ausgeführt werden können, muß ein Programm geladen werden.

Das kann dadurch geschehen, daß das Programm Zeile für Zeile über die Tastatur eingegeben wird, oder dadurch, daß das Programm von einer Datei auf der Floppy Disk geladen wird. Durch Benutzung des GET-Kommandos können Dateien mit Quellprogrammen geladen werden.

Wenn ein Programm auch für eine spätere Benutzung benötigt wird, kann es in eine Datei abgespeichert werden. Dies wird durch die Benutzung des SAVE-Kommandos erreicht. Existiert auf der Floppy Disk bereits eine ältere Version des Programmes und möchte man das geladene Programm unter dem dem gleichen Namen wieder zurückschreiben, so kann hierzu das REPLACE-Kommando benutzt werden.

Wenn ein Programm nicht mehr benötigt wird, muß es gelöscht werden bevor ein neues Programm geladen wird. Dies wird durch die Benutzung des NEW-Kommandos bewirkt. Dieses Kommando hat auch zur Folge, daß alle geöffneten Dateien geschlossen werden und daß SIZE und SIZE$ auf ihre ursprünglichen Werte gesetzt werden.

Wenn eine Datei ein Quellprogramm enthält, das nicht mehr benötigt wird, kann sie durch Benutzung des KILL-Kommandos gelöscht werden.

3.1 GET

Das GET-Kommando ermöglicht es, ein Programm aus einer Datei zu laden. Das Programm wird Zeile für Zeile aus der Datei eingelesen, so als ob es über die Tastatur eingegeben würde. Jede Zeile wird beim Lesen auf syntaktische Richtigkeit überprüft. Der Ladevorgang wird entweder beim Entdecken eines Fehlers oder beim Erreichen des Dateiendes beendet.

Das GET-Kommando kann nur im Kommando-Modus benutzt werden.

Format:

GET<Dateiname>

wobei<Dateiname>definiert ist als:

<Laufwerk-Bezeichnung><Dateiname>

Die<Laufwerk-Bezeichnung>hat die Form:

:fn:

wobei 'n' die Laufwerknummer ist und im Bereich 0 bis N liegen kann (N ist die Anzahl der an das System angeschlossenen Laufwerke).

<Dateiname> hat die Form:

<Name>.<Erw>

wobei <Name> maximal 6 Zeichen und <Erw> maximal 3 Zeichen lang ist. <Name> muß mit einem Buchstaben beginnen; ansonsten dürfen <Name> und <Erw> nur Buchstaben und Ziffern enthalten. "." und <Erw> konnen auch fehlen.

Beim auszuführenden GET-Kommando muß der <Dateiname> ein Laufwerk spezifizieren, das eingeschaltet ist, sowie eine Datei, die auf dem angegebenen Laufwerk existiert.

Falls keine Laufwerk-Bezeichnung angegeben ist, wird :f0: angenommen.

3.2 SAVE

Das SAVE-Kommando bewirkt die Abspeicherung des aktuellen Programmes in eine Datei. Das so abgespeicherte Programm kann durch das GET-Kommando geladen werden. SAVE darf nur im Kommando-Modus benutzt werden.

Format:

SAVE<Dateiname>

Zu<Dateiname>siehe die Angaben unter GET.

Beim Aufruf des SAVE-Kommandos darf noch keine Dateien gleichen Namens auf der angegebenen Floppy Disk vorhanden sein.

3.3 NEW

Das NEW-Kommando bewirkt die Löschung des aktuellen Programms im Arbeitsspeicher und versetzt das System in den Anfangsstatus; das System wartet wieder auf die Eingabe von Kommandos oder Quellzeilen. Alle geöffneten Dateien werden geschlossen und SIZE und SIZE$ werden auf ihre Anfangswerte gesetzt. Das Kommando kann im Stop- oder Kommando-Modus benutzt werden.

Format:

```
NEW
NEW<Hex-Adresse>
```

wobei Hex-Adresse eine 4-stellige Zahl ist, die eine Speicheradresse darstellt. Als Ziffern sind 0 ... 9, A ... F erlaubt. Die erste Stelle der Adresse muß jedoch eine Ziffer sein (auch 0).

Beide Formen haben die oben beschriebene Wirkung.

Wenn eine hexadezimale Adresse angegeben ist, legt sie die höchste Speicheradresse fest, die vom Interpreter für die Speicherung von Daten benutzt werden kann. Diese Form wird nur dann benutzt, wenn Assembler-Routinen des Benutzers in einen Speicherbereich geladen wurden, den normalerweise der Interpreter für seine Daten verwendet.

3.4 KILL

Wenn eine Datei ein Quellprogramm enthält, das nicht mehr benötigt wird, kann sie durch Verwendung des KILL-Kommandos gelöscht werden. Mit diesem Kommando können auch BS-Dateien gelöscht werden, die auf eine andere Art und Weise entstanden sind. Das KILL-Kommando kann im Stop- oder Kommando-Modus verwendet werden.

Format:

```
KILL<Dateiname>
```

Zu <Dateiname> siehe die Anmerkungen unter GET.

Es wird eine Meldung ausgegeben, die anzeigt ob

- die Datei existierte und gelöscht wurde
- die Datei nicht existierte und das Kommando deshalb ignoriert wurde.

Achtung:

Das KILL-Kommando kann auch als BASIC-Quellanweisung innerhalb eines Programms verwendet werden, z. B.

```
0100 KILL "Datei".
```

3.5 REPLACE

Das REPLACE-Kommando kombiniert die beiden Kommandos KILL und SAVE. Die angegebene Datei wird durch das Programm, das sich gerade im Kernspeicher befindet, ersetzt.
REPLACE kann nur im Kommando-Modus benutzt werden.

Format:

REPLACE <Dateiname>

wobei für <Dateiname> die Beschreibung unter GET gilt.

Es wird eine Meldung ausgegeben, die angibt, ob

- die Datei existierte und gelöscht worden ist
- die Datei nicht vorhanden war und der Befehl deswegen nicht ausgeführt wurde.

4 Auflisten eines BASIC Quellprogramms

Aufeinanderfolgende Zeilen des aktuellen BASIC-Programms können mit dem LIST-Kommando ausgegeben werden. Damit kann der Benutzer (im Stop- oder Kommando-Modus) sein Quellprogramm anschauen. Wenn ein Fehler aufgetreten ist, kann er damit feststellen, ob er in einem Bereich seines Programms Zeilen vergessen oder falsch eingegeben hat. Die Ausgabe erfolgt auf den Bildschirm, es sei denn, ein anderes Ausgabegerät wird explizit angegeben.

4.1 LIST

Das LIST-Kommando bewirkt die Ausgabe von Zeilen des aktuellen Programms auf das angegebene Ausgabegerät.

Format:

```
(i)     LIST <Gerät>
(ii)    LIST <Gerät>,<Zeilennummer>
(iii)   LIST <Gerät>,<Zeilennummer>,<Zeilennummer>
(iv)    LIST
(v)     LIST <Zeilennummer>
(vi)    LIST <Zeilennummer>,<Zeilennummer>
(vii)   LIST ,<Zeilennummer>
(viii)  LIST <Zeilennummer>,
(ix)    LIST <Gerät>,<Zeilennummer>,
(x)     LIST <Gerät>,,<Zeilennummer>
```

wobei <Gerät> eines der folgenden Ausgabegeräte spezifiziert:

:LP:	Drucker
:CO:	Bildschirm
:XM:	UART, für die Ausgabe an einen anderen Rechner

<Zeilennummer> ist eine ganzzahlige Konstante. Diese Zahl muß nicht unbedingt zu einer Zeilennummer des Programms korrespondieren. Die Ausgabe beginnt bei der angegebenen Startzeilennummer, oder falls diese nicht existiert, bei der nächsthöheren Zeilennummer und endet bei oder vor der angegebenen Endzeilennummer.

In allen Fällen wo Gerät fehlt, erfolgt die Ausgabe auf den Bildschirm, z. B.

```
LIST
```

ist äquivalent zu

```
LIST :CO:
```

Wenn keine Zeilennummer angegeben ist (Format (i) und (iv)), wird das gesamte Programm ausgegeben.

Wenn zwei Zeilennummern angegeben sind (Format (iii) und (vi)), werden alle Zeilen zwischen der Startzeilennummer und der Endzeilennummer ausgegeben.

Bei drei verschiedenen Formaten wird nur eine Zeilennummer angegeben:

- <Zeilennummer> gefolgt von einem Komma (Format (viii) und (ix)). Das Programm wird von der angegebenen Zeilennummer an bis zum Programmende ausgegeben.

- <Zeilennummer> mit voranstehendem Komma (Format (vii) und (x)). Das Programm wird von seinem Anfang bis zur angegebenen Zeilennummer ausgegeben.

- <Zeilennummer> allein (Format (ii) und (v)). In diesem Fall wird nur die angegebene Zeilennummer ausgegeben.

5 Einstellungen des Interpreters

Es gibt zwei Systemkommandos, die es dem Benutzer ermöglichen, die voreingestellten Werte des Systems zu ändern:

SIZE damit ist die Anzahl der für Dezimalvariable gespeicherten Stellen festlegbar.

SIZE$ damit kann man die Länge von Stringvariablen festlegen, die entweder emplizit oder in einer STRING-Anweisung ohne Längenangabe spezifiziert werden.

5.1 SIZE

Mit dem SIZE-Kommando kann der Benutzer die Anzahl der für Dezimalvariable zu speichernden Stellen festlegen. Dies beeinflußt erstens die maximale Zahlgröße und zweitens die maximale Genauigkeit. Alle Dezimalvariablen werden mit der durch das SIZE-Kommando festgelegten Stellenzahl gespeichert. Die Genauigkeit einer Variablen (d. h. die Anzahl der Stellen nach dem Komma) wird durch die PRECISION-Anweisung bestimmt (s. Teil II Abschnitt 6.4).

Das SIZE-Kommando kann nur im Kommando-Modus benutzt werden.

Format:

```
SIZE
SIZE <ganzzahlige Konstante>
```

wobei <ganzzahlige Konstante> eine ganze Dezimalzahl zwischen 1 und 254 ist.

Falls keine <ganzzahlige Konstante> angegeben ist, wird ein Wert von 10 angenommen.

5.2 SIZE$

Wenn eine Stringvariable in einem Programm deklariert wird, muß sie eine Länge besitzen. Falls die Variable implizit oder in einer STRING-Anweisung ohne Längenangabe deklariert wird, dann wird ein voreingestellter Wert angenommen. Dieser voreingestellte Wert kann durch das SIZE$-Kommando geändert werden. Wird kein SIZE$-Kommando gegeben, wird ein voreingestellter Wert von 25 verwendet.

Das SIZE$-Kommando kann nur im Kommando-Modus benutzt werden.

Format:

```
SIZE$
SIZE$ <ganzzahlige Konstante>
```

wobei <ganzzahlige Konstante> eine ganze Dezimalzahl zwischen 1 und 255 sein muß.

Falls keine <ganzzahlige Konstante> angegeben ist, wird ein Wert von 25 voreingestellt.

6 Ausführen eines BASIC-Programms

Die Ausführung des aktuellen BASIC-Programms wird durch das Eingeben des RUN-Kommandos über die Tastatur angestoßen. Dies kann nur im Kommando-Modus erfolgen. Im Stop-Modus verursacht es einen Fehler.

Das Programm läuft solange bis

- ein Fehler entdeckt wird
- eine STOP-Anweisung ausgeführt wird
- eine der im RUN-Kommando angegebenen Zeilennummern erreicht wird
- ein CTRL/C-Zeichen an der Tastatur eingegeben wird
- eine EXIT- oder END-Anweisung ausgeführt wird.

Solange das Programm ausgeführt wird, ist das System im Run-Modus.

In den ersten vier Fällen, die oben angegeben sind, wird die Ausführung zeitweilig unterbrochen und das System geht in den Stop-Modus. Es wird eine Fehlermeldung auf den Bildschirm ausgegeben, um den Grund für die Beendigung anzuzeigen. Im Stop-Modus können bestimmte Systemkommandos und BASIC-Anweisungen benutzt werden, um das Programm zu testen.

Wenn eine EXIT- oder END-Anweisung ausgeführt wird, wird das Programm beendet, es erscheint keine Fehlermeldung und das System geht wieder in den Kommando-Modus. Alle Dateien werden geschlossen und auf keine der Variablen kann mehr zugegriffen werden. Es wird das Zeichen ">" auf den Bildschirm ausgegeben, um anzuzeigen, daß sich das System wieder im Kommando-Modus befindet und bereit ist für die Eingabe neuer Kommandos oder Quellanweisungen.

6.1 RUN

Das RUN-Kommando stößt die Ausführung des aktuellen BASIC-Programms an. RUN kann nur im Kommando-Modus benutzt werden.

Format:

```
RUN
RUN <Liste mit Zeilennummern>
```

wobei <Liste mit Zeilennummern> eine Menge von ganzen Zahlen aus dem Bereich zwischen 1 und 9999 ist.

Wenn Zeilennummern angegeben sind, müssen sie zu Zeilennummern im aktuellen Programm korrespondieren. Falls eine dieser Zeilennummern erreicht wird, wird die Ausführung zeitweilig unterbrochen und das System geht in den Stop-Modus.

Sonst dauert die Ausführung solange, bis

- ein Fehler entdeckt wird

- eine STOP-Anweisung ausgeführt wird

- ein CTRL/C Zeichen an der Tastatur eingegeben wird

- eine EXIT- oder END-Anweisung ausgeführt wird.

6.2 CONTINUE

Das CONTINUE-Kommando startet die Ausführung des Programms erneut, und zwar an der Stelle, an der sie zeitweilig unterbrochen wurde. Deshalb kann dieses Kommando nur im Stop-Modus benutzt werden. In der gleichen Weise wie beim RUN-Kommando kann eine List mit Zeilennummern angegeben werden, die festlegt, wo die Ausführung wieder unterbrochen werden soll.

Format:

```
CONTINUE
CONTINUE <Liste mit Zeilennummern>
C
C <Liste mit Zeilennummern>
```

wobei <Liste mit Zeilennummern> eine Folge von ganzen Zahlen aus dem Bereich zwischen 1 und 9999 ist.

Beide Formen des Kommandonamens (d. h. CONTINUE und C) sind in ihrer Wirkung identisch.

Wenn eine Zeilennummer angegeben ist, muß sie zu einer

Zeilennummer des aktuellen Programms korrespondieren. Falls die Zeilennummer erreicht wird, wird die Ausführung zeitweilig unterbrochen und das System geht in den Stop-Modus.

Wenn die Ausführung nach Auftreten eines Fehlers fortgesetzt wird, wird die fehlerhafte Zeile nicht wieder ausgeführt; in allen anderen Fällen wird die Ausführung bei der in der Fehlermeldung angegebenen Zeile fortgesetzt.

6.3 ABORT

Das ABORT-Kommando wird benutzt, wenn keine weitere Ausführung des Programms gewünscht wird. Es kann nur im Stop-Modus verwendet werden. Es werden sämtliche Dateien geschlossen und auf die Variablen des Programms kann nich mehr zugegriffen werden. Das System geht wieder in den Kommando-Modus. Es wird das Zeichen ">" ausgegeben, das anzeigt, daß sich das System wieder im Kommando-Modus befindet und die Eingabe von Kommandos oder Quellanweisungen erwartet.

Format:

```
ABORT
```

7 Ändern eines Programms

Wenn ein Programm eingegeben ist, kann es nötig sein, daß es geändert werden muß, d.h. daß Quellzeilen eingefügt, geändert oder gelöscht werden. Das Ändern kann nur im Kommando-Modus erfolgen.

7.1 Einfügen und Ändern von Quellzeilen

Um eine neue Zeile einzufügen oder eine bestehende zu verändern, wird die benötigte Quellzeile einfach zusammen mit ihrer Zeilennummer eingegeben. Falls die angegebene Zeilennummer bereits existiert, wird die alte durch die neue Quellzeile ersetzt. Falls die Zeilennummer noch nicht existiert, wird die neue Zeile richtig in die Folge der Zeilennummern des Programms eingefügt.

7.2 Löschen von Quellzeilen

Zum Löschen von Quellzeilen wird das DELETE-Kommando benutzt. Der Benutzer kann eine Zeile oder eine Folge von Zeilen, die zu löschen sind, angeben.

Format:

```
DELETE <Zeilennummer>
DELETE <Zeilennummer>,<Zeilennummer>
```

wobei <Zeilennummer> eine ganzzahlige Konstante ist.

Die erste Form des DELETE-Kommandos bewirkt die Löschung einer einzelnen Zeile. Die zweite Form löscht alle Zeilen, beginnend bei der ersten Zeilennummer bis zur zweiten Zeilennummer.

8 Testen eines Programms

Falls ein Programm nicht richtig arbeitet, muß der Benutzer den Grund für diesen Fehler suchen und sein Programm entsprechend korrigieren. Der Fehler kann einen Ausführungsfehler zur Folge haben oder nur eine falsche Verarbeitung. Um den Benutzer bei der Fehlersuche zu unterstützen, stehen eine Reihe von Testhilfen zur Verfügung. Sie zerfallen in zwei Klassen, nämlich in Systemkommandos und Quellsprachanweisungen.

8.1 Systemkommandos

Ein Teil der Systemkommandos kann zum Testen von Programmen verwendet werden. Es sind dies sämtliche Kommandos, die im Stop-Modus verwendet werden können, sowie das RUN-Kommando.

Mit dem RUN-Kommando kann der Benutzer die Ausführung seines Programms anstoßen und eventuell Zeilennummern angeben, bei denen die Ausführung unterbrochen werden soll. Das System geht dann in den Stop-Modus.

Wenn das System im Stop-Modus ist, können die Kommandos LIST,KILL und CONTINUE verwendet werden, um Quellzeilen auszugeben, Dateien zu löschen oder die Ausführung des Programms fortzusetzen.

Um jede weitere Ausführung des Programms zu beenden, kann das ABORT-Kommando benutzt werden.

Falls das Programm bei der Ausführung in eine Schleife gerät (das Programm wird ohne Fehler ausgeführt, wird aber nicht beendet), kann die Ausführung des Programms durch Eingeben des CTRL/C-Zeichens unterbrochen werden. Der Grund für die unendliche Schleife kann dann mit den zur Verfügung stehenden Testhilfen gesucht werden.

8.2 Quellanweisungen

Wenn sich das System im Stop-Modus befindet, kann eine Reihe von BASIC-Quellsprachanweisungen zum Testen eines Programms benutzt werden. Das Format dieser Anweisungen ist dasselbe wie das im Teil II für die BASIC-Quellzeilen beschriebene, abgesehen davon, daß jetzt keine Zeilennummer angegeben wird. Die Anweisung wird sofort nach der Eingabe über die Tastatur ausgeführt.

Die Anweisungen, die auf diese Weise verwendet werden können, sind unten angegeben.

LET	Wertzuweisung an eine Variable
OPEN	eröffnen einer Datei/ eines Geräts
CLOSE	schließen einer Datei/ eines Geräts
INPUT	einlesen von Daten aus einer Datei/ einem Gerät
PRINT	ausgeben von Daten auf eine Datei/ ein Gerät
PRECISION	festlegen der Genauigkeit für eine Dezimalvariable
DISPLAY	definieren eines DISPLAY-Feldes

Achtung:

Falls die Unterbrechung innerhalb einer Prozedur auftritt, kann nur auf die lokalen Variablen dieser Prozedur zugegriffen werden.

Beispiel:

```
>list
0010 REM Programm zum Berechnen des Durchschnitts von
0015 REM n Zahlen
0020 DECIMAL a(1:100)
0030 REM für die Speicherung von n Zahlen
0040 LET sum = 0
0050 REM sum : Summe der Zahlen bis jetzt
0060 LET x = 3
0070 LET y = 2
0080 REM (x,y) ist ein Eingabezeiger
0090 PRINT ,CLR,LOC(2,1),"Wie viele Zahlen?"
0100 INPUT ,n
0110 REM n : Anzahl der Zahlen
0120 DISPLAY i$,24,10,4,8
0130 LET i$ = "0"
0140 PRINT ,LOC(24,15),"Zahlen bis jetzt",LOC(x,1),"?"
0150 FOR i = 1 TO n
0160 REM lies naechste Zahl
0170 INPUT ,a(i)
0180 LET i$ = STR(i)
0190 PRINT ,LOC(x,y-1),a(i),"?"
0200 LET sum = sum + a(1)
0210 REM die naechsten 6 Zeilen stellen den Eingabezeiger
0220 LET y = y + LEN(STR(a(i))) + 1
0230 IF y  = 60 THEN 280
0240 PRINT ,LOC(k,y-1)," "
0250 LET y = 2
0260 LET x = x + 1
0270 PRINT ,LOC(x,1),"?"
0280 NEXT i
0285 PRINT ,LOC(24,1),"
0290 PRINT ,LOC(24,1),"Der Durchschnitt ist ",sum/n
0300 END
>
```

In diesem Beispiel sind alle Eingaben des Benutzers unterstrichen, um sie von dem Text zu unterscheiden, den Programm und System ausgeben.

Das oben abgedruckte Programm kann eingegeben oder mit dem GET-Kommando aus einer Datei geladen worden sein.

Es liest eine Zahl n ein, gefolgt von n Zahlen.
Es zeigt laufend einen Zähler für die Anzahl der bis dahin eingegebenen Zahlen an, und wenn dieser Zähler gleich n ist, den Durchschnitt dieser n Zahlen.

Sobald das Programm geladen ist, befindet sich das System im Kommando-Modus und der Benutzer kann die Ausführung des Programms anstoßen durch Eingabe von

```
>run
```

Der erste sichtbare Beweis, daß das Programm richtig arbeitet, erfolgt, wenn es die Zeile 90 erreicht. Es löscht den Bildschirm und wartet auf eine Eingabe, nachdem es

```
Wie viele Zahlen?
```

ausgegeben hat. Wenn der Benutzer seine Antwort eingegeben hat, gibt das Programm (Zeilen 130,140)

```
0 Zahlen bis jetzt
```

links unten am Schirm und ein "?" am Anfang der dritten Bildschirmzeile aus. Dieses "?" ist der Hinweis, daß das System weitere Eingabe erwartet.

Sobald der Benutzer eine Zahl eingegeben hat, wird der Zähler für die bis jetzt eingegebenen Zahlen vor der Erklärung

```
Zahlen bis jetzt
```

ausgegeben. Danach wird die eingegebene Zahl wieder ausgegeben, so daß das "?" davor überschrieben wird. Dafür erscheint ein "?" nach der Zahl.

Das Programm erwartet dann die Eingabe der nächsten Zahl.

Wenn eine Zeile mit Eingabezahlen bis zur 60. Stelle gefüllt ist, ist die Bedingung (y = 60) in Zeile 230 falsch und die Zeilen 240 bis 279 werden zum ersten Mal ausgeführt.

An dieser Stelle gibt das System

```
ERROR AD
0240 PRINT ,LOC(k ,y-1)," "
>
```

aus. Die mit dem Fehlerschlüssel AD (s.Anhang C) verbundene Information ist 'Variable nicht besetzt'.

Dies zeigt an, daß einer der Variablen aus Zeile 240 noch kein Wert zugewiesen ist. Der senkrechte Strich zeigt genau an, daß k noch nicht besetzt ist.

Der Benutzer kann sich diesen Bereich des Programms anschauen durch die Eingabe von

```
>list 230,250

0230 IF y  = 60 THEN 280
0240 PRINT ,LOC(k,y-1)," "
0250 LET y = 2
>
```

An diesem Programmteil sieht der Benutzer:

- in Zeile 240 werden nur zwei Variable (k,y) benutzt

- bei der Ausführung von Zeile 230 war einer von ihnen (y) ein Wert zugewiesen.

Der Benutzer muß deshalb herausfinden, warum k noch kein Wert zugewiesen wurde. Das Durchsuchen des gesamten Programms zeigt, daß dies das einzige Auftreten von k im Programm ist, und es wird deutlich, daß dies ein Eingabefehler ist (k statt x).

Um die Zeile zu ändern, muß aus dem Stop-Modus in den Kommando-Modus zurückgegangen werden. Dazu muß der Benutzer eingeben:

```
>abort
>240 print ,loc(x,y-1)," "
```

Das Programm sollte nun neu gestartet werden, um sicher zu sein, daß die Änderung das gewünschte Resultat erzielt hat. Das Programm wird nun vollständig ausgeführt; es werden n Zahlen eingelesen und als Antwort "Der Durchschnitt ist" ausgegeben.

Falls z.B. die 6 Zahlen 2,4,6,8,10,12 eingegeben werden, gibt das Programm

Der Durchschnitt ist 2

aus. Diese Antwort ist falsch. Der Benutzer muß jetzt die Logik seines Programms untersuchen, um den Grund für die falsche Berechnung des Durchschnitts herauszufinden. Der Durchschnitt berechnet sich als "Summe" geteilt durch n. Deshalb können drei Dinge falsch gelaufen sein:

- der angezeigte Wert ist nicht sum/n

- n enthält einen falschen Wert

- sum ist falsche aufaddiert worden.

```
>list 290

0290 PRINT ,LOC(24,1),"Der Durchschnitt ist ",sum/n
>
```

zeigt sofort, daß der angezeigte Wert sum/n ist.

Obwohl ersichtlich ist, daß n nicht geändert wird, kann das RUN-Kommando benutzt werden, um dies zu beweisen, indem nämlich eine Zeilennummer mit angegeben wird.

```
>run 290
```

Dies bewirkt dieselbe Ausführung wie vorher, aber sobald die Zeile 290 erreicht wird, gibt das System aus:

```
ERROR C5
0290 PRINT ,LOC(24,1),"Der Durchschnitt ist ",sum/n
>
```

Das System ist nun im Stop-Modus und die Benutzung des PRINT-Kommandos zeigt, das n den richtigen Wert enthält.

```
>print ,n,crlf
6
>
```

Deshalb enthält sum einen falschen Wert.

```
>list 200
0200 LET sum = sum + a(1)
>
```

zeigt die Zeile, die sum aufaddiert.

Falls der Benutzer den Schreibfehler nicht sofort entdecken kann, könnte ein ähnliches RUN-Kommando in Verbindung mit gleich aufgebauten CONTINUE-Kommandos benutzt werden, um zu untersuchen, ob bei jedem Durchlauf der Schleife der richtige Wert zu sum addiert wird. Dies zeigt dann, daß jedesmal derselbe Wert (nämlich 2) addiert wird, was den Fehler deutlich macht.
Anstatt a(i) zu sum dazuzuzählen, wird jedesmal a(1) addiert.
Der Benutzer kann jetzt in den Kommando-Modus zurückgehen, seine Änderung vornehmen und erneut sein Programm starten. Falls a(1) durch a(i) ersetzt wird, arbeitet das Programm fehlerfrei.

9 Produktions-Modus

Sobald ein Programm einmal entwickelt und getestet ist, kann es im Betrieb eingesetzt werden. Es kann erforderlich sein, den Vorgang des Ladens und Startens von Programmen zu vereinfachen, damit dies auch von weniger erfahrenen Benutzern ausgeführt werden kann. Für diesen Zweck gibt es das START-Kommando.

9.1 START

Das START-Kommando bestimmt eine Datei, die Systemkommandos enthält. Es sind sämtliche Kommandos erlaubt, mit der Einschränkung, daß ein eventuell enthaltenes START-Kommando das letzte Kommando dieser Datei sein muß. Eine START-Datei enthält im allgemeinen GET- und RUN-Kommandos, um ein oder mehrere Programme zu laden und zu starten.

Sobald das START-Kommando gegeben ist, werden alle Kommandos in der Datei genauso ausgeführt, als ob sie vom Benutzer eingegeben würden.

Dieses Kommando kann nur gegeben werden, wenn sich das System im Kommando-Modus befindet.

Format:

START <Dateiname>

Zu <Dateiname> siehe die Anmerkungen unter GET.

Es gibt in BASIC kein Hilfsmittel, um eine START-Datei zu kreieren. Dies kann auf zwei Arten geschehen:

- Benutzung von EDIT im BS 610
- Erstellen eines BASIC-Programms, um solche Dateien aufzubauen.

ANGANG A : LISTE DER SCHLÜSSELWÖRTER

ABORT
ABS
ASCII
ATT
BIN
CALL
CLOSE
CLR
CONTINUE
CRLF
DECIMAL
DELETE
DEF
DEF$
DISPLAY
END
ENTER
EXIT
FIXED
FOR
GET
GOSUB
GOTO
IDX
IF
INCHAR
INPUT
INPUTC
INT
JNC
KILL
LEN
LET
LIST
LOC
MSK
NEW
NEXT
NOR
ON
OPEN
PRECISION
PRINT
PROC
QUOTE
REM
REPLACE
RETURN
RUN

SAVE
SGN
SIZE
SIZE$
SKIP
START
STOP
STR
STRING
SUB
TAB
THEN
UND
VAL

ANHANG B : Zeichenvorrat

Dez. Code	Bezeichnung	Reaktion
0	NULL	Keine Reaktion
1	SOH	Keine Reaktion
2	VIDEO OFF (STX)	Leert den Bildschirm ohne den Speicher zu löschen. Cursor bleibt stehen.
3	VIDEO ON (ETX)	Löscht STX und läßt das Bild wieder erscheinen
4	ERASE LINE (EOT)	Löscht die Zeile und setzt den Cursor auf ihren Anfang
5	ENQ	Das ENQUIRY-Lämpchen leuchtet
6	ACK	Das ACK-Lämpchen leuchtet
7	BELL	Erzeugt einen Pieps-Ton
8	BACKSPACE	Setzt den Cursor eine Position nach links
9	TAB (HT)	Keine Reaktion
10	LINE FEED (LF)	Setzt den Cursor auf die gleiche Spalte in der nächsten Zeile. Rollt in der letzten Zeile.
11	VERTICAL TAB (VT)	Setzt den Cursor auf die gleiche Spalte in der nächsten Zeile. Stopt in der letzten Zeile.
12	ROLL UP (FF)	Schiebt den Bildschirm um eine Zeile nach oben und löscht die unterseste Zeile
13	CARRIAGE RETURN	Setzt den Cursor auf die erste Position der gleichen Zeile
14	UNDERLINE	Die Zeichen, die dem Code folgen, werden auf dem Bildschirm unterstrichen,. bis ein NORMAL-Code erreicht wird
15	NORMAL	Die Zeichen, die diesem Code folgen, werden normal dargestellt
16	CURSOR LOAD	Setzt den "Direkt-Adressier-Modus" für den Cursor. Die nächsten zwei Zeichen werden als Zeilen- und Spaltennummer interpretiert.

Dez. Code	Bezeichnung	Reaktion

17	DC1	Keine Reaktion
18	DC2	Keine Reaktion
19	DC3	Keine Reaktion
20	DC4	Keine Reaktion
21	NAK	Das NAK-Lämpchen leuchtet
22	SYN	Keine Reaktion
23	ROLL DOWN (ETB)	Schiebt den Bildschirm eine Zeile nach unten und löscht die oberste Zeile
24	CAN	Setzt den Cursor eine Zeichen nach rechts. Wenn der Bildschirm automatischen Zeilenschalt-Modus ist, wird der Cursor bei Erreichen der letzten Spalte auf die erste Position der nächsten Zeile gesetzt
25	ERASE PAGE	Löscht den Bildschirm und setzt den Cursor an den Anfang der ersten Zeile
26	SUB	Keine Reaktion
27	ESC	Keine Reaktion
28	FS	Setzt den Cursor in die gleiche Spalte der vorhergehenden Zeile
29	GS	Setzt den Cursor auf die erste Stelle der ersten Zeile
30	RS	Keine Reaktion
31	US	Keine Reaktion
32	SP	Leerzeichen
33 126		Der alphanumerische Zeichensatz (94 Zeichen), der auf dem Bildschirm dargestellt wird. Siehe nächste Seite.
127	DEL	Keine Reaktion

Dezi. Code	Zeichen	Dezi Code	Zeichen.
33	!	80	P
34	"	81	Q
35	#	82	R
36	$	83	S
37	%	84	T
38	&	85	U
39	'	86	V
40	(	87	W
41	)	88	X
42	*	89	Y
43	+	90	Z
44	,	91	[
45	-	92	\
46	.	93	]
47	/	94	↑
48	0	95	←
49	1	96	`
50	2	97	a
51	3	98	b
52	4	99	c
53	5	100	d
54	6	101	e
55	7	102	f
56	8	103	g
57	9	104	h
58	:	105	i
59	;	106	j
60	<	107	k
61	=	108	l
62	>	109	m
63	?	110	n
64	@	111	o
65	A	112	p
66	B	113	q
67	C	114	r
68	D	115	s
69	E	116	t
70	F	117	u
71	G	118	v
72	H	119	w
73	I	120	x
74	J	121	y
75	K	122	z
76	L	123	{
77	M	124	:
78	N	125	}
79	O	126	~

ANHANG C: Fehlermeldungen

Schlüssel	Bedeutung
83	Aufgerufene Routine nicht geladen
84	ASCII-Argument 0
86	INCHAR - nur einfache Stringvariable sind erlaubt
87	INPUT - unzulässiges Stringformat
89	MSK - keine ganze Zahl mehr
8A	Unzulässige Maske
8B	SUB - Startposition nicht im zulässigen Bereich
8C	Division durch Null
8D	Benutzerunterbrechung (CTRL/C)
8E	FIXED - unzulässiger String
8F	Unzulässiger Gebrauch von SKIP/FIXED
90	Falsche Parameteranzahl im Funktionsaufruf
91	ENTER - Routine muß mit JMP beginnen
92	NEW - falscher Parameter
93	Quelle kein String
96	EXIT - kein PROC aktiv
97	CALL - zu viele Parameter
98	CALL - zu wenig Parameter
99	CALL -PROC läuft bereits (keine Rekursion!)
9A	Funktionsparameter hat falschen Typ
9B	Keine Rekursion von Funktionen erlaubt
9C	(Intern) Datei kann nicht geschlossen werden
9D	NEW - Parameter keine DISPLAY Variable
9E	FIXED - Satzende während der Eingabe
9F	FIXED - Schlußapostroph fehlt (1. Zeichen war Apostroph!)
A0	FIXED - Länge größer als max. Stringlänge
A1	FIXED - Stringlänge zu groß
A2	Quelldaten nicht dezimal
A3	(Intern) Ende von CI
A4	VAL - String stellt keine Zahl dar
A5	Logische Datei nicht eröffnet
A6	CLOSE - logische Datei war nicht eröffnet
A7	OPEN - logische Datei bereits eröffnet
A8	DISPLAY - unzulässige Cursorposition
A9	DISPLAY - Parameter außerhalb der zulässigen Grenzen

Schlüssel	Bedeutung
AA	DISPLAY - Variable muß vom Typ String sein
AB	LET - unzulässige linke Seite der Zuweisung
AC	Funktion benötigt Parameter / Feld nicht indiziert
AD	Variable nicht besetzt
AE	IF - unzulässiger Vergleich
AF	IF - Bergleich fehlt
B0	IF - Vergleich zwischen unterschiedlichen Typen
B1	NEXT - arithmetischer Überlauf
B2	FOR / NEXT - Schachtelungsfehler
B3	NEXT - kein zugehöriges FOR
B4	LET - linke Seite falscher Datentyp
B5	RETURN - kein zugehöriges GOSUB
B6	Unbekannte log. Dateibezeichnung
B7	log. Dateibezeichnung fehlt
B8	Wert muß zwischen - 32767 und + 32767 liegen
B9	LOC - unzulässige Cursorposition
BA	STOP (kein Fehler)
BB	LEN - Stringparameter erforderlich
BC	BIN - unzulässiger Wert
BD	ATT - unzulässiger Attributwert
BE	ABS - Parameter keine Dazimalzahl
BF	Undefiniertes Symbol
C0	SIZE ($) - falsche Länge (1 L 255)
C1	PRECISION - Feld bereits besetzt
C2	PRECISION - Variable hat falschen Typ
C3	PRECISION - falsche Dezimalstellenzahl (0 bis SIZE)
C4	Funktionsparameter hat falschen Typ
C5	Unterbrechungsstelle (kein Fehler)
C6	Variable mehrfach definiert
C7	SUB - Startpunkt nicht im zulässigen Bereich
C8	Feldindex max. 32767
C9	Variable bereits mit anderem Datentyp deklariert
CA	STRING - max. Länge ist 255
CB	STRING / DECIMAL - falsche Länge
CC	Falsches Binärliteral
CD	Operand hat falschen Typ
CE	INPUT - falsches Zahlenformat
CF	Anweisung an falscher Stelle
D0	PROCEND fehlt
D1	Anweisung nach PROC-PROCEND unzulässig
D2	END fehlt
D3	Speicherüberlauf - SAVE und Neustart
D4	PROC Name mehrfach definiert
D5	Indizierte Variable ist kein Feld
D6	Komma fehlt
D7	Unzulässiges Symbol im Ausdruck

Schlüssel	Bedeutung
D8	Zuwenig Operatoren im Ausdruck
D9	Zuwenig Operanden im Ausdruck
DA	falscher Feldindex
DB	Klammerfehler im Ausdruck
DC	LIST - unzulässige Zeilengrenzen
DD	LIST - unzulässiges Zeichen
DE	INPUT - max. Länge ist 80 Zeichen
DF	Falsches Schlüsselwort
E0	Schlüsselwort fehlt
E1	Dezimalkonstante fehlt
E2	Benutzersymbol erwartet
E3	Text hinter Anweisungsende
E4	Syntaktischer Fehler
E5	Arithmetischer Überlauf bei Multiplikation
E6	(Intern) Operation noch nicht vorhanden
E7	Keine Zeilenänderung vor ABORT
E8	(Intern) end-of-extent
E9	Unzulässiges Kommando
EA	Variable kein Feld oder keine Funktion
EB	Unbekannte Variable im Ausdruck
EC	Kommando in diesem Zusammenhang unzulässig
ED	Arithmetischer Überlauf bei Addition
EE	Falsche Anzahl von Feldindizes
EF	Falscher Feldindex
F0	Unzulässiger Operand in der Quelle
F1	Unzulässiger Ausdruck
F2	Zeile existiert nicht
F3	Zeile außerhalb des Bereichs
F4	Unzulässige Zeilennummer
F5	Unbekanntes Schlüsselwort
F6	Variable unbesetzt
F7	Keine ganze Zahl mehr
F8	String zu kurz für empfangene Daten
F9	Unzulässiger Variablenname
FA	Variablenname zu lang (6 Zeichen)
FB	Unzulässiges Zeichen in Dezimalkonstante
FC	Dynamischer Speicher voll
FD	Schlußapostroph fehlt
FE	Nicht-darstellbares Zeichen im String
FF	Vorzeitiges Textende

ANHANG D : Beispielprogramm

```
0010      STRING (1) t$(1:100)
0020      STRING (7) tab$(1:100)
0030      STRING (80)in$,cust$
0040      STRING (60) x$
0050      PRINT ,CLR,LOC(1,1),"Kunde:"
0060      DISPLAY cust$,1,10,70,8
0070      LET cust$=NEW(cust$)
0080      OPEN si, "file2"
0090      INPUT si,n
0100      IF n   100 THEN 870
0110      FOR i=1 TO n
0120      INPUT si, tab$(i)
0130      LET t$(i)="n"
0140      NEXT i
0150      CLOSE si
0160      PRINT ,LOC(3,1),"Teil-Nr. Beschreibung "
0165      PRINT ,LOC(3,60),"Menge Preis"
0170      PRINT ,LOC(25,40),"Gesamtwert Auftragsnr."
0180      DISPLAY tot$,25,60,7,8
0190      LET tot$="0"
0200      LET line=4
0210      PRINT ,LOC(line,1)," "
0220      DISPLAY in$,line,2,3,8
0230      LET in$=NEW(in$)
0240      IF in$="end" THEN 800
0250      PRINT ,LOC(line,1)," ",LOC(line,5)," "
0255      LET in$=MSK("999",VAL(in$))
0260      FOR i=1 TO n
0270      IF SUB (tab$(i),1,3)=in$ THEN 310
0280      NEXT i
0290      PRINT ,LOC(line,10),"Ungültige Teil-Nr."
0300      GOTO 210
0310      OPEN si,"file1"
0320      FOR j=1 TO i
0330      INPUT si,x$
0340      NEXT j
0350      CLOSE si
0360      LET ord$=SUB(x$,1,3)
0370      LET cost$=SUB(x$,LEN(x$)-4,5)
0380      LET lim=VAL(SUB(x$,LEN(x$)-8,4))
0390      LET x$=SUB(x$,4,LEN(x$)-12)
0400      IF ord$  in$ THEN 700
0410      LET v=VAL(SUB(tab$(i),4,4))
0420      PRINT,LOC(line,10),x$
0430      IFv =lim THEN 450
0440      GOSUB 5000
0450      PRINT ,LOC(line,70)," ",MSK("zz9.99",VAL(cost$))
0455      PRINT ,LOC(line,62)," "
0460      DISPLAY q$,line,63,4,4
```

```
0470        LET q=VAL(NEW(q$))
0480        PRINT ,LOC(line,62),"  ",LOC(line,67),"  "
0490        IF q =v THEN 500
0494        PRINT ,LOC(line,62),ATT(49),STR(v),NOR
0498        LET q=v
0500        LET t=q*VAL(cost )
0510        PRINT ,LOC(line,71),MSK("zz9.99",t)
0520        LET tot$ =MSK("zzz9.99",VAL(tot$)+t)
0530        LET tab$(i)=JNC(SUB(tab$(i),1,3),STR(v-q))
0540        IF v-q  = lim THEN 560
0550        GOSUB 5000
0560        LET line=line+1
0570        IF line   24 THEN 210
0580        LET line=4
0590        FOR i=4 TO 23
0600        PRINT,LOC(i,1),BIN(4)
0610        NEXT i
0620        GOTO 210
0700        PRINT,LOC(line+1,1),ATT(48),"Dateien passen nicht ",CRLF
0710        EXIT
0800        KILL file2
0810        OPEN so,"file2"
0820        PRINT so,n,CRLF
0830        FOR i=1 TO n
0840        PRINT so,QUOTE,tab$(i),QUOTE,CRLF
0850        NEXT i
0860        PRINT,LOC(line+2,1),UND,"PROGRAMMENDE",NOR
0870        EXIT
5000        IF t$(i)="y" THEN 5030
5010        PRINT :lp:,"Teil",in$,"Bestand niedrig",CRLF
5020        Let t$(i)="y"
5030        RETURN
9999        END
```

ANHANG F : Begriffserläuterungen

Aktueller Parameter: Aktuelle Parameter sind die Werte und Bezüge auf Variable, die beim Funktions-/Prozeduraufruf an die Funktion/Prozedur übergeben werden. Aktuelle Parameter können Konstante, Variable oder Ausdrücke sein.

Assemblerroutine: Eine Assemblerroutine ist ein Programm, das aus Assemblerbefehlen für den Mikroprozessor besteht, assembliert und dann vom BS 610 geladen wurde. Sie wird von BASIC aus durch die ENTER-Anweisung gestartet.

Ausdruck: Durch einen Ausdruck wird die Berechnung eines Wertes festgelegt.

BS 610: BS 610 ist das Betriebssystem für das Siemens System 6.610

Byte: Ein Byte ist die Grundeinheit des Speichers des Mikroprozessors. Es kann 256 verschiedene Werte beinhalten, die Zeichen, Dezimalziffern oder Befehle darstellen können.

Datei: Eine Datei ist eine bezeichnete Folge von auf der Floppy-Disk/ Kassette gepeicherten Bytes. Der Dateiname kann bis zu 9 (INTEL), bzw. 6 (IBM) Zeichen lang sein und muß mit einem Buchstaben beginnen.

Dezimalvariable: Eine Dezimalvariable ist eine Variable, die eine Dezimalzahl enthält.

Dezimalzahl: Eine Dezimalzahl ist eine Zahl, die aus einer oder mehreren Dezimalziffern (0, ... 9) besteht, die eventuell von einem Dezimalpunkt und weiteren Dezimalziffern gefolgt werden.

einfache Variable: Eine einfache Variable ist eine einzelne Größe, deren Wert während der Ausführung geändert werden kann. Sie wird durch eine Bezeichnung angezeigt, die eindeutig ist für diese Variable.

Feldvariable: Eine Feldvariable ist eine geordnete Menge von Größen (Elementen) desselben Typs. Auf die gesamte Menge wird durch eine einzige Bezeichnung Bezug genommen; ein einzelnes Element wird durch einen zusätzlich angegebenen Index ausgewählt.

Formaler Parameter: Formale Parameter sind in Funktions-/Prozedurdefinitionen verwendete Variable. Wenn die Funktion/Prozedur aufgerufen wird, werden die aktuellen Parameter für die formalen Parameter eingesetzt.

Floppy-Disk: Eine Floppy-Disk ist eine flexible magnetisierbare Scheibe zum Speichern von Daten.

Funktion: Eine Funktion führt eine festgelegte Berechnung aus und liefert ein Ergebnis, wobei die an sie übergebenen Parameterwerte benutzt werden. In BASIC gibt es Standard-(oder systemdefinierte) und vom Benutzer definierte Funktionen.

ganze Zahl: Eine ganze Zahl ist eine Dezimalzahl ohne Dezimalpunkt und Dezimalstellen.

Genauigkeit: Die Genauigkeit einer Dezimalvariable ist die Anzahl der Dezimalstellen (d. h. die Anzahl der Stellen nach dem Dezimalpunkt), die diese Variable enthält.

Achtung: Dies ist nicht die Anzahl der signifikanten Stellen. Einige oder alle Dezimalstellen dieser Variablen können Null sein.

Index: Ein Index ist ein Wert, der angibt, welches Element eines Feldes ausgewählt wird. Die Werte, die der Index eines Feldes annehmen kann, werden durch die Ober- und Untergrenze des Feldes begrenzt. Die Untergrenze muß 0 oder 1 sein. Falls ein Indexwert außerhalb der Grenzen verwendet wird, wird ein Fehler gemeldet. Die Grenzen werden bei der Deklaration des Feldes durch eine STRING- oder DEZIMAL-Anweisung definiert.

Kassette: Eine Kassette ist ein 1/4 Zoll breites Magnetband in einem Plastikgehäuse.

Kommando-Modus: Als Kommando-Modus wird der Status des Systems bezeichnet, in dem sich das System befindet, bevor irgendein Programm ausgeführt wird oder nachdem ein Programm ordnungsgemäß beendet wurde (durch EXIT oder END Anweisungen oder das ABORT-Kommandos).

lokale Variable: Eine lokale Variable ist eine Variable, die nur innerhalb des Programmteils benutzt werden kann, in dem sie deklariert wurde. Alle in Prozeduren deklarierten und benutzten Variablen sind lokal innerhalb dieser Prozedur.

Operator: In BASIC gibt es keine Stringoperatoren. Die arithmetischen Operatoren sind:

+ Addition
- Subtraktion
* Multiplikation
/ Division

Programm: Ein Programm ist ein BASIC-Quellprogramm und besteht aus einer Folge von Quellzeilen. Eine Quellzeile besteht aus einer Zeilennummer und einer BASIC-Quellsprachanweisung.

Prozedur: Eine Prozedur ist ein Programmteil, der einmal definiert und dann überall, wo nötig, benutzt wird. Alle innerhalb einer solchen Prozedur deklarierten und benutzten

	Variablen sind lokal für diese Prozedur. Falls auf außerhalb dieser Prozedur gelegene Variable zugegriffen werden muß, müssen sie als Parameter übergeben werden.
Rekursion:	Rekursion ist der Vorgang, bei dem während der Ausführung einer Prozedur diese Prozedur erneut aufgerufen wird. Dies kann direkt (Prozedur A ruft Prozedur A auf) oder indirekt (Prozedur A ruft Prozedur B auf, die wiederum A aufruft) geschehen. Im Siemens Commercial BASIC sind Rekursionen nicht erlaubt.
Run-Modus:	Als Run-Modus wird der Status des Systems bezeichnet, in dem ein Programm ausgeführt wird.
Schleife:	Eine Schleife ist ein Programmteil, der wiederholt ausgeführt wird. Dies kann entweder explizit durch die Verwendung von GOTO- und IF ... THEN-Anweisungen oder für die Bildung von FOR-Schleifen durch FOR- und NEXT-Anweisungen erreicht werden.
Stop-Modus:	Als Stop-Modus wird der Status des Systems nach einer zeitweiligen Unterbrechung der Programmausführung bezeichnet.
String:	Ein String ist eine Folge von Zeichen. Er wird durch die in Apostrophe (") eingeschlossene Folge dargestellt.
UART	Universal Asynchronous Receiver-Transmitter: Schnittstelle zum asynchron gekoppelten Rechner
Unterprogramm:	Ein Unterprogramm ist ein Programmteil, der einmal definiert und überall wo nötig aufgerufen werden kann (durch die GOSUB-Anweisung). Im Gegensatz zur Prozedur werden beim Aufruf keine Parameter übergeben und die Variablen des Unterprogramms sind nicht nur lokat verwendbar.

Zahl: Eine Zahl ist eine Dezimalzahl, d. h. sie besteht aus einer oder mehreren Dezimalziffern, eventuell gefolgt von einem Dezimalpunkt und weiteren Ziffern und kann eventuell ein vorangestelltes Vorzeichen haben.

ANHANG G : Interpreterroutinen

Es gibt eine Anzahl von Routinen im Interpreter, deren Benutzung durch Assemblerroutinen, die durch die ENTER-Anweisung aufgerufen werden, möglich ist. Nach der Ausführung der ENTER-Anweisung zeigt das Registerpaar DE auf eine Adresstabelle für diese Routinen.

Tabellen-position		
0	TAVDV	Test des Gültigkeitsbits Eingabe: HL zeigt auf Attributsvektor Ausgabe: Carry gelöscht, falls Gültigkeitsbit gesetzt. Falls Carry gesetzt, enthält A einen Fehler-Code.
1	XAVTP	Hole Typbyte Eingabe: HL zeiht auf Attributsvektor Ausgabe: A enthält Typbyte des Attributvektors
2	XAVDP	Hole Datenzeiger Eingabe: HL zeigt auf Attributvekor Ausgabe: DE enthält Datenzeiger-Wort des Attributvektors
3	XAVPR	Hole Genauigkeitsbyte Eingabe: HL zeigt auf Attributsvektor Ausgabe: A enthält Genauigkeitsbyte des Attributvektors
4	XAVSZ	Hole Längenbyte (SIZE) Eingabe: HL zeigt auf Attributvektor Ausgabe: A enthält das Längenbyte des Attributvektors
5	GPTST	Hole Zeiger auf Stringdatenbereich Eingabe: HL zeigt auf Datenzeiger-Wort des Attributvektors Ausgabe: DE zeigt auf das erste Byte des Stringdatenbereichs (das Längenbyte)
6	SAVVD	Setze Gültigkeitsbit Eingabe: HL zeigt auf Attributvektor Ausgabe: Standard
15	CVID	Konvertiert eine ganze Zahl in eine Dezimalzahl Eingabe: HL zeigt auf Attributsvektor des dezimalen Ergebnisses DE enthält den zu konvertierenden Binärwert Ausgabe: Carry gelöscht, falls erfolgreich

Tabellen-position		
16	CVDI	Konvertiert eine Dezimalzahl in eine ganze Zahl Eingabe: HL zeigt auf Attributsvektor der Dezimalzahl Ausgabe: Carry gelöscht, falls erfolgreich HL enthält Binärwert. Falls Carry gesetzt, enthält A einen Fehlerschlüssel und HL enthält null
17	TAKE	Fordert Bereich im dynamischen Speicher Eingabe: A enthält benötigte Größe in Byte (1 bis 255) Ausgabe: Carry gelöscht, falls erfolgreich HL zeigt auf erstes Byte des Speichers für den Benutzer Falls Carry gesetzt, enthält A Fehler-Code, HL enthält null
18	GIVE	Gibt einen Bereich des dynamischen Speichers frei Eingabe: HL zeigt auf erstes Byte des freizugebenden Speicherbereichs Ausgabe: HL gleich null, Carry gelöscht
19	CBFSP	Kreiert Pufferbereich im Stack Eingabe: HL enthält die Anzahl der anzulegenden Bytes Ausgabe: Carry gelöscht, falls erfolgreich HL zeigt auf das niedrigstwertige Byte des Stacks Der Stackpointer SP steht auf HL-2; das obere Ende des Stacks enthält den alten Wert von HL. Falls Carry gesetzt, enthält A einen Fehler-Code
20	RBFSP	Gibt durch CBFSP angelegten Stack frei Eingabe: SP besetzt wie beim Ende von CBFSP Ausgabe: SP besetzt wie beim Eintritt in CBFSP

INDEX

	SEITE
A	
ABORT-Kommando	42
ABS-Funktion	56
Ändern eines Programms	83
Ändern von Quellzeilen	83
Anweisungen:	
CALL	42
CLOSE	49
DECIMAL	25
Deklaration	24
Ein/Ausgabe	48
DEF	28
DEF$	29
DISPLAY	54
END	47
ENTER	43
EXIT	47
FOR	35
GOSUB	38
GOTO	33
IF ... THEN	34
INCHAR	51
INPUT	50
INPUTC	51
Kommentar	24
LET	32
NEXT	37
ON ... GOTO	34
OPEN	48
PRECISION	53
PRINT	52
PROC	40
PROCEND	41
REM	24
RETURN	39
Steueranweisungen	32
START	48
STOP	37
STRING	27
Zuweisung	31
Aktueller Parameter	28 f
Arithmetischer Ausdruck	20
Operator	20
ASCII-Funktion	65
Assemblerroutine	43 ff
ATT-Funktion	63
Auflisten eines Programms	76 ff

	SEITE
Ausdruck, arithmetischer	20
String-	21
Ausführen eines Programms	80
Ausgabeanweisungen	48 ff
B	
Beenden des Dialogs	71
Benutzer-definierte Funktion	23/27
BIN-Funktion	62
BYE-Kommando	71
C	
CALL-Anweisung	42
CLOSE-Anweisung	49
CLR-Funktion	64
CONTINUE-Kommando	81
CRLF-Funktion	61
D	
Dateibezeichnung, logische	48
Daten	17
DECIMAL-Anweisung	25
DEF-Anweisung	28
DEF$-Anweisung	29
Deklaration: Daten	24 ff
Feld	25/27
Funktion	27f
DELETE-Kommando	83
Dezimalzahl	II 2.1.1
Dezimalvariable	17/25
DISPLAY-Anweisung	54
E	
Ein/Ausgabeanweisungen	48
Einfügen von Quellzeilen	83
Eingaberoutinen: FIXED	66
SKIP	66
END-Anweisung	47
ENTER-Anweisung	43
EXIT-Anweisung	47

	SEITE
F	
Feld	19
FIXED-Eingabefunktion	66
FOR-Anweisung	35
Formaler Parameter	28f
Funktion: Deklaration	27
Standard-	22/56
vom Benutzer definiert	23/27
G	
GET-Kommando	72
GOSUB-Anweisung	38
GOTO-Anweisung	33
H	
I	
IDX-Funktion	60
IF ... THEN-Anweisung	34
INCHAR-Anweisung	51
Index	19/25
INPUT-Anweisung	50
INPUTC-Anweisung	51
INT-Funktion	56
Interpreter-Einstellungen	78
J	
JNC-Funktion	58
K	
KILL-Kommando	74
Kommandomodus	69
Kommandos:	
ABORT	82
BYE	71
CONTINUE	81
DELETE	83
GET	72
KILL	74
LIST	76
NEW	74
REPLACE	75

	SEITE
RUN	81
SAVE	73
SIZE	78
SIZE$	78
START	90
Kommentar	24
Konstante: Dezimalzahl	19
String	18

L

Laden eines Programms	72
LEN-Funktion	60
LET-Anweisung	32
LIST-Kommando	76
LOC-Funktion	63

M

MSK-Funktion	59

N

NEW-Kommando	74
NEW-Funktion	65
NEXT-Anweisung	37
NOR-Funktion	64

O

ON ... GOTO-Anweisung	34
OPEN-Anweisung	48

P

PRECISION-Anweisung	53
PRINT-Anweisung	52
PROC-Anweisung	40
PROCEND-Anweisung	41
Produktionsmodus	90
Prozedur: Definition	40
Aufruf	42

SEITE

Q

QUOTE-Funktion 61

R

REM-Anweisung 24
REPLACE-Funktion 75
RETURN-Anweisung 39
RUN-Kommando 81
Run-Modus 69

S

SAVE-Kommando 73
SGN-Funktion 57
SIZE-Kommando 78
SIZE$-Kommando 90
SKIP-Eingabefunktion 66
Speichern eines Programms 73
Standardeingabefunktion: FIXED 66
SKIP 66
Standardfunktionen: 56 ff
ABS 56
ASCII 65
ATT 63
BIN 62
CLR 64
CRLF 61
IDX 60
INT 56
JNC 58
LEN 60
LOC 63
MSK 59
NEW 65
NOR 64
QUOTE 61
SGN 57
STR 57
SUB 58
TAB 62
UND 64
VAL 60
START-Anweisung 48
-Kommando 90
Steueranweisungen 32
STOP-Kommando 37
STR-Funktion 57

	SEITE
String: Ausdruck	21
Feld	19/27
Konstante	18
Variable	27
STRING-Anweisung	27
SUB-Funktion	58
Systemkommandos: s. Kommados	
T	
TAB-Funktion	62
Testen eines Programms	74
U	
UND-Funktion	64
Unterprogramm	38
V	
VAL-Funktion	60
Variable	18
Vergleich	33
Vergleichsoperator	33
Z	
Zuweisung	31

NACHTRAG

1) NEW-Kommando

Die hexadezimale Adresse im NEW-Kommando darf aus maximal fünf Ziffern bestehen und nicht kleiner als 9D18h sein. Ist sie niedriger, so tritt der Fehler 92 (NEW-unerlaubter Parameter) auf.

2) BYE

Das Kommando BYE zum Verlassen des Interpreters kann sowohl im Kommando- wie auch im Stop-Modus gegeben werden.

Vorschläge

Korrekturen

An

SIEMENS AKTIENGESELLSCHAFT

Bereich Daten- und Informationssysteme
D ÖA Technische Druckschriften

Postfach 83 29 40

8000 München 83

für Druckschrift:
Basis-Datensystem 6·000
AMBOSS 1
Commercial BASIC
Programmieranleitung

Bestell-Nr. D 41/1088–02
Ausgabe Juli 1979

Der Druckfehlerteufel läßt sich nicht immer überlisten; sollten Sie beim Lesen dieser Unterlage auf Druckfehler gestoßen sein, bitten wir Sie, uns diese mit diesem Vordruck mitzuteilen. Ebenso dankbar sind wir für Anregungen und Verbesserungsvorschläge.

Absender:

Name

Firma/Dienststelle

Anschrift

Telefon /

Seite:	Kapitel:	Abschnitt:	Vorschläge und/oder Korrekturen:

Herausgegeben vom Bereich Basisinformationssysteme
Postfach 83 29 40, D–8000 München 83

SIEMENS AKTIENGESELLSCHAFT

Bestell-Nr.: D 41/1088–02
Printed in the Federal Republic of Germany
785 AG 8800.6 (1335)