dietz 621

BASEX

Beschreibung

Heinrich Dietz
Industrie-Elektronik
433 Mülheim a.d. Ruhr
Solinger Str. 9
Tel. (0 21 33) 48 50 24
Telex 8 56 770

DIETZ Computer SYSTEME

Lickfeld

dietz 621

BASEX

Beschreibung

Januar 1975

Heinrich Dietz
Industrie-Elektronik
433 Mülheim a.d. Ruhr
Solinger Str. 9
Tel. (02133) 485024
Telex 856770

DIETZ Computer SYSTEME

Inhalt

Ausgabe 12.74

Vorwort

BASEX ist eine interaktive Programmiersprache für Echtzeit-Anwendungen. Sie ist auf der inzwischen weltweit verbreiteten Dialog-Sprache BASIC aufgebaut und umfaßt außer deren Sprachumfang eine Vielzahl weiterer Sprachelemente zur Beschreibung des Echtzeit-Verhaltens und für die Prozeßdaten-Verarbeitung. Dialogfähigkeit und einfacher Sprachaufbau machen BASEX zu einem leicht erlernbaren, benutzerfreundlichen Programmiersystem.

Die Sprache BASEX wurde am Institut für Physik der Universität Freiburg (Breisgau) entwickelt. Die erste Implementierung erfolgte auf Computer-Systemen vom Typ DIETZ 621.

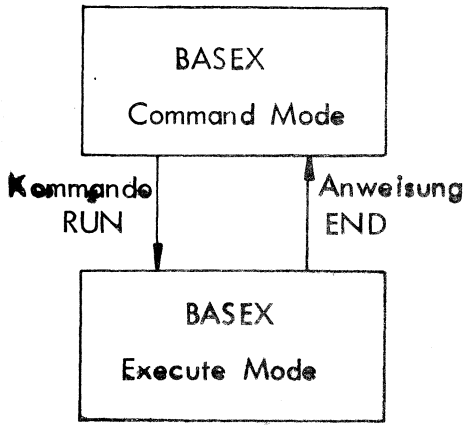
Die vorliegende Benutzer-Anleitung beschreibt BASEX so, wie es auf dem DIETZ 621 implementiert ist. Sie soll Benutzern von DIETZ 621-Systemen helfen, ihren Einsatz zu planen und sie zu programmieren, zu bedienen und erfolgreich zu betreiben.

Dialog mit dem System

BASEX ist eine interaktive Programmiersprache: Programme werden im Dialog zwischen Benutzer und System ein- und ausgegeben, verändert und zur Ausführung gebracht.

Dabei wird zwischen zwei Betriebsarten unterschieden, in denen sich das System befinden kann:

- Bedienungsbetrieb (command mode)
- Ausführungsbetrieb (execute mode)



Im Bedienungsbetrieb nimmt das System Kommandos des Bedieners an, die sofort ausgeführt werden, und speichert Programm-Anweisungen, die später ausgeführt werden sollen.

Im Ausführungsbetrieb wird das gespeicherte BASEX-Programm ausgeführt.

Die Gliederung in zwei Betriebsarten ist nicht streng, da sowohl Kommandos als auch Programme vom gleichen BASEX-System interpretiert werden; zum Verständnis des Systems erweist sie sich jedoch als praktisch.

In Abschnitt 2.1.2.1 sind Grundsätze des Bedienungsbetriebs beschrieben; die restlichen Abschnitte beschäftigen sich mit den im Ausführungsbetrieb abgearbeiteten Elementen vom BASEX-Programm.

Kommandosprache

BASEX-Kommandos sind Befehle, die der Benutzer dem System über das Konsolgerät erteilt. Sie bestehen aus einem Kommandowort, das von einigen Parametern gefolgt sein kann, und werden vom System sofort ausgeführt. Der Zustand des Systems, in dem es Kommandos akzeptiert, wird "Bedienungsbetrieb" (command mode) genannt.

Zum Beispiel bewirkt das Kommando

```
LIST 50,100
```

daß ein Teil des im System befindlichen BASEX-Programms, beginnend mit Anweisung 50 und endend mit Anweisung 100, auf dem Konsolgerät ausgedruckt wird. Der Ausdruck beginnt unmittelbar nach Abschluß der Kommando-Zeile mit "Wagenrücklauf" (CR).

Im Bedienungsbetrieb können Programm-Anweisungen eingegeben werden; zum Beispiel in der Form

```
30 LET A=3
```

Jedoch werden diese vom System nicht sofort ausgeführt, sondern anhand der davorstehenden Zahl als Anweisungen erkannt und gespeichert.

Durch das Kommando

```
RUN
```

wird das gespeicherte Programm zur Ausführung gebracht; das System geht in den "Ausführungsbetrieb" (execute mode) über, in dem es keine Kommandos (oder Eingaben von Anweisungen) mehr zuläßt.

Mit Ende des Programms (END-Anweisung) kehrt das System wieder in den Bedienungsbetrieb zurück, was durch Ausgabe von

```
*READY
```

auf dem Konsolgerät gemeldet wird.

Statt des vollständigen Kommando-Wortes kann auch eine Kurzform eingegeben werden, die aus den drei ersten Buchstaben besteht, z.B.

```
LIS 50,100
```

Fehler bei der Eingabe können auf folgende Weise korrigiert werden:

Eingabe von ← eliminiert 1 vorangehendes Zeichen
Eingabe von ←← eliminiert 2 vorangehende Zeichen
...
Eingabe von "Rubout" eliminiert alle vorangehenden Zeichen der Zeile.

BASEX kennt folgende Kommando-Worte:

Grund-Kommandos:	LIST	Programm listen auf Konsolgerät	(s. 2.2.2.1)
	DELETE	Programmteil löschen	"
	SCRATCH	Gesamtes Programm löschen	"
	RENUMBER	Programm neu numerieren	"
	RUN	Programm starten	"
Peripherie:	READ	Programm einlesen über Streifenleser	(s. 2.2.2.2)
	PUNCH	Programm auslochen über Streifenlocher	"
	LISP	Programm listen auf Schnelldrucker	"
Plattensystem:	LOAD	Programm laden aus Datei	(s. 2.3.2.1)
	SAVE	Programm ablegen auf Datei	"
	KILL	Programm-Datei löschen	"
	INITIALIZE	Programm initialisieren	(s. 2.3.2.3)
	END	Übergang in Basis-Betriebssystem	(s. 2.3.3.1)
Zeitsystem:	TIME	Systemzeit abfragen/eingeben	(s. 2.4.3.2)

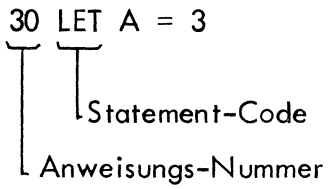
Die BASEX-Kommandos sind in den angegebenen Abschnitten ausführlich beschrieben.

Programmiersprache

BASEX-Programme bestehen aus einer Folge von Anweisungen. Jede Anweisung findet in einer Zeile Platz und besteht aus

- der Anweisungsnummer (1...9999)
- der Anweisung im eigentlichen Sinne (Statement), die stets mit einem für den Anweisungs-Typ spezifischen Kennwort (Statement-Code) beginnt.

Beispiel:



Die Anweisungen werden in aufsteigender Reihenfolge ihrer Nummern abgearbeitet – außer bei Sprüngen (GOTO, GOSUB), ausgeführten Verzweigungen (IF) und Schleifen (NEXT). Kommentare (REM) und Anweisungen beschreibenden Charakters (DIM, CHAR, DEF, DATA, EQUI, EQUO) werden bei der Ausführung übergangen.

Programmanweisungen werden im Bedienungsbetrieb (s. 2.1.2.1) Zeile für Zeile eingegeben; die zeitliche Reihenfolge ist dabei unmaßgeblich.

Fehler bei der Eingabe können durch ← bzw. "Rubout" korrigiert werden (s. 2.1.2.1).

Anweisungen müssen den für die Programmiersprache BASEX gültigen Regeln entsprechen, wie sie im folgenden beschrieben sind. Formale, insbesondere syntaktische Fehler werden unmittelbar nach Eingabe einer Zeile vom System erkannt und durch Ausgabe von

ERR n

auf dem Konsolgerät erkannt; n ist die Fehler-Nummer (s. Fehlerliste 9.1)

Leerzeichen sind in beliebiger Zahl, auch zwischen zusammengehörigen Zeichen, zulässig; jedoch nicht innerhalb von Statement-Codes.

Der Zeichenvorrat von BASEX umfaßt alle 26 Buchstaben des Alphabets, die Ziffern 0...9 sowie eine Reihe von Sonderzeichen:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
+ - * / ↑ = # < > ( ) @ ' " & ! ? . , ; ß % :

Statement-Codes

BASEX kennt folgende Statement-Codes:

BASIC-Statements:	REM	Kommentar	(2.2.3.1)
	DIM	Zahlenfeld	(2.2.3.2)
	CHAR	Stringvariable	(2.2.3.3)
	DEF	Funktion	(2.2.3.4)
	LET	Zuweisung	(2.2.3.5)
	DATA	Konstanten-Liste	(2.2.3.6)
	READ	Konstanten-Zuweisung	"
	RES	Setzen Konstanten-Zeiger	"
	INPUT	Eingabe	(2.2.3.7)
	PRINT	Ausgabe	(2.2.3.8)
	GOTO	Sprung	(2.2.3.9)
	GOSUB	Unterprogramm-Sprung	(2.2.3.10)
	RETURN	Rücksprung	"
	IF	Verzweigung	(2.2.3.11)
	FOR	Schleifen-Anfang	(2.2.3.12)
	NEXT	Schleifen-Ende	"
	CALL	Prozedur-Aufruf	(2.2.3.13)
	END	Programm-Ende	(2.2.3.14)
Plattensystem:	LINK	Aufruf Segment	(2.3.2.4)
	ENDS	Ende Segment	"
Realtime-System:	WAIT	Warten	(2.4.2.1)
	START	Programm-Auftrag	(2.4.2.2)
	STOP	Abschluß Auftragsprogramm	(2.4.2.3)
	AFTER	Zeitauftrag	(2.4.3.4)
	ON INT	Interrupt-Auftrag	(2.4.4.1)
	ENAB	Interrupt zulassen	(2.4.4.2)
	DISAB	Interrupt sperren	"
Prozeß-Ein/Ausgabe:	EQUI	Eingabe-Makro	(2.5.2.1)
	EQUO	Ausgabe-Makro	(2.5.2.2)
	PUT	Makro-Aufruf	(2.5.2.3)

Die BASEX-Statements sind in den angegebenen Abschnitten ausführlich beschrieben.

Namen

Namen sind mnemotechnische Bezeichnungen in BASEX-Programmen. Sie bestehen aus

- einem Buchstaben,
- dem bis zu 3 Buchstaben oder Ziffern folgen können.

Beispiele für Namen:

A
Z
A1
AA1
AAAA
TEMP
X23
N2A8

Namen bezeichnen:

- Variablen
- Systemvariablen
- Systemprozeduren
- Benutzer-Funktionen

Als Namen sollen nicht verwendet werden:

- Statement-Codes, wie REM, LET, ...
- Anweisungs-spezifische Schlüsselwörter, wie OF, THEN, ...
- Operator-Namen, wie NOT, AND, ...
- Standard-Funktionen, wie SQR, CHG, ...

Datentyp Zahl

Numerische Werte werden intern als Gleitkomma-Zahlen ("Real") dargestellt. Dieser Datentyp wird als "Zahl" bezeichnet.

Von jeder Zahl werden im Speicher 4 byte (32 bit) belegt; davon nehmen

- die Mantisse 3 byte
- der Exponent (zur Basis 2) 1 byte

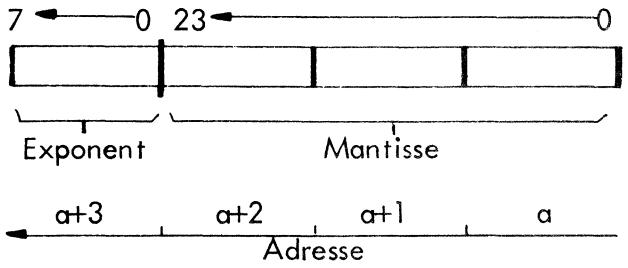
ein.

Mantissen können damit im Bereich $(-2^{23} + 1) \ldots (2^{23} - 1)$ liegen, was einer Genauigkeit der Darstellung von 6 Dezimalstellen entspricht.

Der gesamte Zahlenbereich beträgt absolut $0.5 \cdot 10^{-31} \ldots 0.5 \cdot 10^{45}$.

Ganze Zahlen behalten bei allen arithmetischen Grundoperationen ihren Charakter bei, solange die Mantisse die Bereichsgrenze nicht überschreitet. Dies gilt nicht für Divisionen mit nicht-ganzzahligem Ergebnis.

Im Speicher haben Zahlen folgenden Aufbau:



Mantisse und Exponent sind binäre Zweierkomplement-Zahlen.

Zahlenkonstanten

Numerische Konstanten können im Programm in unterschiedlicher Form geschrieben werden.

Folgende Darstellungsweisen sind möglich:

- Ganzzahlen:
  0
  2
  123
  009999
  -8
  -345678

- mit Dezimalpunkt:
  2.0
  -3.45
  0.7
  .7
  -.00009
  -265.37

- in Gleitkommaform:
  2E3
  2.0E0
  -6E-1
  -6E-01
  0.99E12
  -.234E-10

Bemerkung:

In der Gleitkomma-Form bedeutet die Zahl hinter dem Buchstaben E den Exponenten zur Basis 10.

2E3 hat also den Wert $2 \cdot 10^3 = 2000$.

Zahlenvariablen

Zahlenvariablen bezeichnen Speicherplätze, die einen numerischen Wert (eine Zahl) enthalten.

Es wird unterschieden zwischen

- einfachen Zahlenvariablen und
- indizierten Zahlenvariablen.

Einfache Zahlenvariablen werden durch einen Namen identifiziert, z.B.:

A
OTTO
X2
DAT1

Der für sie benötigte Speicherplatz wird automatisch reserviert, sobald der zugehörige Name im Programm zum ersten Mal erscheint.

Indizierte Zahlenvariablen sind Bestandteil eines ein- oder zweidimensionalen Zahlenfeldes (array). Sie werden durch einen Namen identifiziert, hinter dem in Klammern ein oder zwei Indizes stehen, z.B.:

A(0) A(6*7) Z (X) B15 (S(3)-T)	einfach indizierte Zahlenvariablen
X(1,2) Y(E+4,L) ARRA (A(3),X*8) N2(0,0)	zweifach indizierte Zahlenvariablen

Der Name bezeichnet ein Zahlenfeld, das in einer DIM-Anweisung reserviert werden muß. Der Index bezeichnet die Position innerhalb des Feldes. Bei zweifach indizierten Zahlenvariablen gibt der erste Index die Zeile, der zweite die Spalte des Feldes an.
Indizes beginnen mit 0, 1, 2 ...; d.h. die jeweils erste Position hat den Index 0!

Bemerkung:

Für einfache und indizierte Zahlenvariablen sind die gleichen Namen zulässig; die Variablen X und X(0) sind nicht identisch!

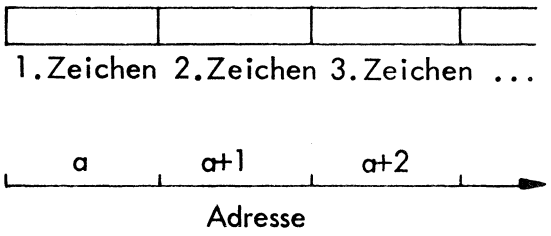
Datentyp String

Zeichenketten werden intern im ASCII-Code dargestellt. Dieser Datentyp wird als "String" bzeichnet.

Für jedes Zeichen wird intern im Speicher 1 byte belegt. Strings haben beliebige Längen zwischen 1 und 1000 byte.

Es sind alle Zeichen des ASCII-Codes in Strings darstellbar, insbesondere die 64 druckbaren Zeichen (einschließlich Leerschritt) sowie Steuerzeichen (Wagenrücklauf, ...). Es können aber auch beliebige Bitmuster in Strings enthalten sein.

Im Speicher haben Strings folgenden Aufbau:



Stringkonstanten

Stringkonstanten können im Programm in unterschiedlicher Form geschrieben werden.

Folgende Darstellungen sind möglich:

- Druckbare ASCII-Zeichen,
  von " eingeschlossen:
  "A"
  "ABC"
  "12"
  "DIES␣IST␣NR.1"

- Hexa-Ziffern,
  von % eingeschlossen:
  %4A%
  %D4C508FF%

Bemerkung:

Jedes Paar von Hexa-Ziffern stellt ein String-Zeichen dar. Damit lassen sich beliebige Bitmuster definieren, insbesondere auch nicht druckbare ASCII-Zeichen.

Stringvariablen

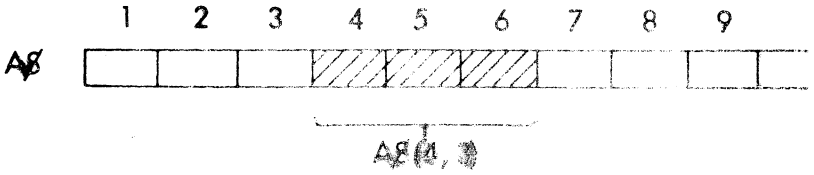
Stringvariablen kennzeichnen Speicherplätze, die Zeichenketten enthalten.

Sie werden durch Namen gekennzeichnet, denen das Dollar-Zeichen $ angehängt wird:

A$
TEXT$
M3$
L5A$

Der für eine Stringvariable benötigte Speicherplatz kann in einer CHAR-Anweisung unter Angabe der Länge (1...32967 Zeichen) reserviert werden. Einer String-Variablen, die in keiner CHAR-Anweisung vorkommt, wird vom Programm automatisch ein Platz von 2 Zeichen reserviert.

Ein String-Name, hinter dem in Klammern zwei Indizes (p,l) stehen, bezeichnet einen Teilstring. Der Index p bezeichnet die Position des ersten Zeichens, der Index l die Länge des Teilstrings. Dadurch kann aus dem Inhalt einer Stringvariablen ein beliebiges Stück herausgegriffen werden:



Beispiele für Teilstring-Definitionen:

A$(4,3)
TEXT$(X,12)
M3$(3*Y,Z+2)

Bemerkung:

Für Zahlen- und Stringvariablen sind die gleichen Namen zulässig, da letztere durch $ gekennzeichnet sind; die Variablen A und A$ sind nicht identisch!

Arithmetische Ausdrücke

Arithmetische Ausdrücke sind verknüpfte Terme, die den Wert einer Zahl haben, mit Hilfe von binären arithmetischen Operatoren. Der Wert eines arithmetischen Ausdrucks ist eine Zahl.

Binäre arithmetische Operatoren sind:

+	Addition	Beispiel: A+3	
-	Subtraktion	6.5-B	
*	Multiplikation	X*Y	
/	Division	C/2	
↑	Potenzierung	Z↑3	(= Z^3)

Es gelten die üblichen Regeln für die Präzedenz der Operatoren (s. 2.1.5.4). Bei gleicher Präzedenz gilt "Klammer vor rechts".

Ferner können folgende binäre Operatoren verwendet werden:

MIN	Minimum	Beispiel: A MIN B	Wert: A wenn A < B
			" B " A ≥ B
MAX	Maximum	A MAX B	Wert: A wenn A ≥ B
			" B " A < B

Als Term können in arithmetischen Ausdrücken vorkommen:

- Zahlenkonstante	Beispiel: 6.5	
- Zahlenvariable	A(0)	
- Systemvariable (Input-Typ)	INW(3)	
- Mathematische Funktionen	SIN(X)	
- Konvertierungsfunktionen (mit String als Argument)	CHG(T$)	
- Logische Verknüpfung	A AND B	
- Vergleichsoperation	X > Y	(auch: A$="MAUS")
- Arithmetischer Ausdruck	C/2	

Bemerkung:

Die Operator-Zeichen + und - treten auch als unitäre arithmetische Operatoren auf (d.h. können allein vor einem Term stehen).
Beispiele:

```
+5
-3.7
-(A+2*B).
```

Logische Ausdrücke

Logische Ausdrücke verknüpfen Terme, die den Wert einer Zahl haben, mit Hilfe von binären logischen Operatoren. Der Wert eines logischen Ausdrucks ist entweder die Zahl 0 oder die Zahl 1.

Binäre logische Operatoren sind:

AND	Konjunktion	Beispiel: A AND B	Wert: 0 wenn A=0 oder B=0 " 1 " A#0 und B#0
OR	Disjunktion	Beispiel: A OR B	Wert: 0 wenn A=0 und B=0 " 1 " A#0 oder B#0

Hierzu gehört auch der unitäre logische Operator

NOT	Negation	Beispiel: NOT A	Wert: 0 für A#0 " 1 " A=0

Beispiele für logische Ausdrücke:

DAT1 AND DAT2
X OR Y
A AND NOT B OR C entspricht: (A AND (NOT B)) OR C

Vergleichsausdrücke

Vergleichsausdrücke (Relationen) verknüpfen Terme, die den Wert einer Zahl haben oder einen String darstellen, mit Hilfe von Vergleichsoperatoren. Der Wert eines Vergleichsausdruckes ist die Zahl 0 oder die Zahl 1.

Vergleichsoperatoren sind:

=	gleich	Beispiel: A = B	Wert: 0 wenn A#B " 1 " A =B
>	größer	A > B	Wert: 0 wenn A≤B " 1 " A>B
>=	größer oder gleich	A >=B	Wert: 0 wenn A< B " 1 " A≥B
<	kleiner	A < B	Wert: 0 wenn A≥B " 1 " A< B
<=	kleiner oder gleich	A<=B	Wert: 0 wenn A> B " 1 " A≤B
#	ungleich	A # B	Wert: 0 wenn A= B " 1 " A#B

Benutzbare Terme: Siehe arithmetische Ausdrücke (2.1.5.1) sowie Strings oder Stringausdrücke (werden mit dem binären Wert der ASCII-Zeichen berechnet).

Beispiele für Vergleichsausdrücke:

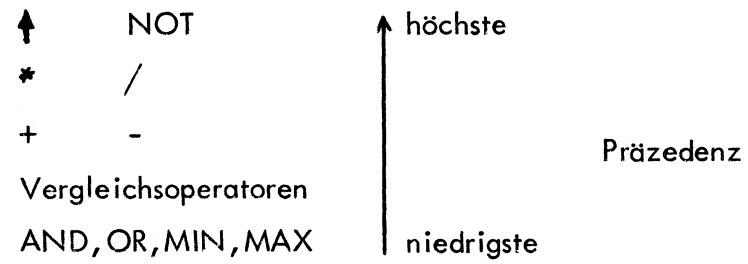
```
X>2
DAT1#DAT2
A$ = "MAUS"
1000<=Y(7)
```

Anweisungsbeispiel:

```
25Ø IF A(X)>= 6.75 THEN 18Ø
```

Präzedenz von Operatoren

Für die Präzedenz von arithmetischen, logischen und Vergleichsoperatoren (d.h. die Rangfolge ihrer Abarbeitung in Ausdrücken, soweit sie nicht durch andere syntaktische Regeln, wie Klammerung, festgelegt ist) gilt folgendes Schema:



Befinden sich Stringvergleiche in einem Ausdruck, so haben die zugehörigen Vergleichsoperatoren die höchste Präzedenz.

Stringausdrücke

Stringausdrücke verknüpfen Terme, die einen String darstellen, mit Hilfe von binären Stringoperatoren. Das Ergebnis ist wiederum ein String.

Binäre Stringoperatoren sind:

AND $	Konjunktion	Beispiel: %1A% AND$ %ØF%	Ergebnis: %ØA%
OR $	Disjunktion	" %1A% OR$ %ØF% "	%1F%

Diese Operatoren führen bitweise UND- bzw. ODER-Verknüpfungen der Strings durch. Bei Strings unterschiedlicher Länge wird die Operation auf die Länge des kürzeren von beiden beschränkt; der andere wird "linksbündig" bearbeitet.

&	Verkettung	Beispiel: "A" & "B"	Ergebnis: "AB"

Mit diesem Operator werden zwei Strings verkettet, d.h. aneinandergesetzt.

Hierzu gehört auch der unitäre Stringoperator

NOT $	Negation	Beispiel: NOT%ØE%	Ergebnis: %F1%

Er bewirkt die bitweise Komplementierung des Strings.

Benutzbare Terme sind:

- Stringkonstanten
- Stringvariable
- Stringausdrücke

Beispiele für Stringausdrücke:

```
A$ AND $ B  $
TX$ OR $ % ØF ØF ØF%
A1$ AND $ NOT $ A2$
```

Bemerkung:

Rechts vom Gleichheitszeichen einer LET-Anweisung darf nur ein Stringoperator stehen.

Mathematische Funktionen

Folgende mathematische Funktionen sind im System implementiert:

ABS(x)	Absolutwert	$\|x\|$
INT(x)	Ganzzahl-Teil	
SGN(x)	Vorzeichen	Wert: 1 wenn $x \geq 0$ " -1 " $x < 0$
LOG(x)	nat. Logarithmus	ln x
EXP(x)	Exponentialfunktion	e^x
SQR	Quadratwurzel	$x^{1/2}$
SIN(x)	Sinus	sin x
COS(x)	Cosinus	cos x
TAN(x)	Tangens	tan x
ATN(x)	Arcustangens	arctan x
RND(0)	Zufallsfunktion	Wertebereich: > 0... < 1

Als Argument dürfen Zahlenkonstanten, Zahlenvariable oder beliebige arithmetische Ausdrücke verwendet werden.
Das Argument von RND ist ohne Wirkung.

Funktion CHG

Die Funktion CHG dient zur Umwandlung des binären Inhalts eines Strings in eine Zahl bzw. umgekehrt.

Umwandlung String → Zahl:

```
CHG (a$)
```

Der binäre Inhalt des Strings a$ (1 oder 2 byte lang) wird in eine positive Zahl verwandelt (0...255 oder 0...65535). Ist der String länger als 2 byte, erfolgt Fehlermeldung.

Umwandlung Zahl → String:

```
CHG (a)
```

Die Zahl a wird in einen String von 2 byte Länge verwandelt, dessen binärer Inhalt (16 bit entsprechend 0...65535) dem Wert der Zahl entspricht. Ist die Zahl größer als 65535, erfolgt Fehlermeldung.

Die Funktion CHG hat vor allem den Zweck, den Übergang zwischen (als Strings definierten) 16-bit-Ganzzahlen (wie sie z.B. an Prozeßschnittstellen auftreten) und dem Datentyp Zahl (der für die Weiterverarbeitung geeignet ist) herzustellen.

Beispiele:

```
10 LET A$ = CHG (B)
25 LET X(0) = CHG (T$(4,2))
50 IF CHG (Q$) > 2500 THEN 150
```

Systemvariablen

Systemvariablen sprechen bestimmte Funktionen des Systems an und werden im Programm wie einfache oder indizierte Zahlenvariablen verwendet.

Es gibt zwei Arten von Systemvariablen:

- Input-Typ
- Output-Typ

Der Input-Typ liefert bei der Abarbeitung einen Zahlenwert; er wird wie eine Zahlenvariable behandelt und kann wie diese in beliebigen Ausdrücken vorkommen. Er ist jedoch links vom Gleichheitszeichen in LET-Anweisungen verboten. Typische Anwendung: Eingabe eines Wertes aus dem Prozeß.

Der Output-Typ ist stets links vom Gleichheitszeichen in LET-Anweisungen zu verwenden; ihm wird bei der Abarbeitung der Wert des rechts vom Gleichheitszeichen stehenden Ausdrucks zugewiesen. Typische Anwendung: Ausgabe eines Wertes an den Prozeß.

Außerdem werden Systemvariable vom Output-Typ in der PUT-Anweisung verwendet; in diesem Falle wird kein Wert ausgewiesen. Typische Anwendung: Ausgabe eines Steuersignals an den Prozeß.

Eine Reihe von Systemvariablen ist Bestandteil des BASEX-Systems bzw. wird bei der Systemgenerierung dem Betriebssystem hinzugefügt. Der Benutzer kann jedoch auch eigene Systemvariablen im BASEX-Programm definieren; dies geschieht von den Anweisungen EQUI und EQUO unter Angabe des Maschinencodes, der das entsprechende "Makro" repräsentiert.

Systemvariable sind ohne oder mit Index definiert:

- einfache Systemvariable: Name
- indizierte Systemvariable: Name (Index)

Beispiele:

```
100 LET A=HOUR
120 IF INB(2)=1 THEN 150
200 LET OUTW(Y+2)=X+5
6000 PUT HOME
```

HOUR = einfache Input-Variable
INB(2) = indizierte Input-Variable
OUTW(Y+2) = indizierte Output-Variable
HOME = einfache Output-Variable

Systemprozeduren

Systemprozeduren erweitern den durch die Statements definierten Sprachumfang. Sie sind Bestandteile des Betriebssystems bzw. werden diesem bei der Systemgenerierung hinzugefügt.

Systemprozeduren

- sind durch einen Namen gekennzeichnet,
- werden durch das Statement CALL unter diesem Namen aufgerufen,
- können mit ein bis vier Parametern behaftet sein, die den Zusammenhang mit dem übrigen Programm herstellen.

Die Parameter werden der Systemprozedur vom Programm übergeben, wobei Anzahl und Typ für jede Prozedur festgelegt ist. Folgende Parameter-Typen sind möglich:

- Zahl: Zahlenkonstante, beliebige Zahlenvariable oder arithmetischer Ausdruck.
  Übergeben wird der numerische Wert.

  Beispiele:

  20
  VAR
  AB (5*I)
  X + SIN (PHI)

- String: Stringkonstante oder einfache Stringvariable.
  Übergeben werden die Anfangsadresse und die Länge des Strings.

  Beispiele:

  "ANTON"
  TX$

- Array: Indizierte Zahlenvariable.
  Übergeben wird die Adresse des entsprechenden Elements im DIM-Feld.

  Beispiele:

  A (0)
  XDAT (2,4)
  AB (5*I)

Eine Vielzahl von Systemprozeduren ist entweder fester oder bei der Systemgenerierung wahlweise einbezogener Bestandteil des vom Hersteller gelieferten BASEX-Systems.

Der Benutzer kann darüberhinaus eigene Systemprozeduren erstellen und in die Systemgenerierung einbeziehen.

Systemvariable ERR

Die Systemvariable ERR enthält einen Fehler-Code, der bei der Abarbeitung bestimmter Systemprozeduren (z.B. Dateibefehle) erzeugt wird. Es handelt sich dabei um Fehler, die nicht zum Abbruch des Programms führen.

Der Benutzer kann ERR im Programm über Anweisungen vom Typ IF, GOTO...OF oder GOSUB...OF abfragen, wann immer er dies für nötig hält. Jedoch sollte die Abfrage erfolgen, bevor (in der gleichen Programmebene) die nächste Systemprozedur mit CALL aufgerufen wird.

Der Fehlercode ist eine Zahl, deren Bedeutung von der jeweiligen Systemprozedur abhängt. Er ist unter den jeweiligen Prozeduren beschrieben.

Beispiel für die Anwendung von ERR:

```
100 CALL OPEN (0,"SIGMA",3)
115 GOSUB 800
...
800 IF ERR=0 THEN 820
810 GOTO ERR-3 OF 830,840,850
820 RETURN
830 ...
```

Bemerkung:

Jede Programmebene besitzt physisch einen eigenen Speicherplatz für ERR, so daß gleichzeitig in mehreren Ebenen ablaufende Systemprozeduren ihre Fehlermeldungen nicht gegenseitig verändern.

BASEX-Befehle aus BASIC

In BASEX ist der Sprachumfang üblicher Implementierungen der Programmiersprache BASIC enthalten.

In den folgenden Abschnitten sind die Kommandos und Programm-Anweisungen beschrieben, die aus BASIC stammen und das Kernstück von BASEX bilden.

In BASIC erfahrene Benutzer werden diese Befehle kennen; jedoch ist zu beachten, daß einige davon erweiterte bzw. auf die Hardware-Konfiguration abgestimmte Funktionen haben.

Die Abschnitte 2.2.2.1 und 2.2.2.2 behandeln Grund- und erweiterte Kommandos; in den restlichen Abschnitten sind Programmanweisungen beschrieben.

Grundkommandos

Unabhängig von der Konfiguration des Systems sind folgende Grund-Kommandos vorgesehen:

LIST	Gesamtes Programm listen
LIST n	Programm Anweisung n listen
LIST,n	Programm bis Anweisung n listen
LISTm,	Programm ab Anweisung m listen
LIST m,n	Programm von Anweisung m bis Anweisung n listen
SCRATCH	Gesamtes Programm löschen
DELETE n	Programm Anweisung n löschen
DELETE,n	Programm bis Anweisung n löschen
DELETE m,	Programm ab Anweisung m löschen
DELETE m,n	Programm von Anweisung m bis Anweisung n löschen
RENUMBER m,n	Programm neu numerieren. Neue erste Anweisungsnummer = m; Schrittweite = n
RUN	Gesamtes Programm ausführen
RUN n	Programm Anweisung n ausführen
RUN,n	Programm bis Anweisung n ausführen
RUN m,	Programm ab Anweisung m ausführen
RUN m,n	Programm von Anweisung m bis Anweisung n ausführen

Bemerkungen:

LIST bewirkt die Ausgabe des Quellprogramms auf dem Konsolgerät.

m,n bezeichnen Anweisungsnummern.

Statement REM

- m REM Bemerkung

REM erlaubt es, in das Programm zum Zweck besserer Verständlichkeit Bemerkungen einzufügen. Bei der Programm-Ausführung werden sie übergangen.

Beispiele:

```
10 REM   DIES IST EINE BEMERKUNG
20 REM *** PROGRAMMTEIL 1 ***
90 REM ENDE PROGRAMM XYZ
```

Hinweis:

Ein Sprung auf eine REM-Anweisung ist erlaubt. Da sie nicht ausführbar ist, wird das Programm mit der nächsthöheren Anweisungs-Nummer fortgesetzt.

Kommandos READ, PUNCH und LISP

Folgende Kommandos sind nur bei Ausrüstung des Systems mit Lochstreifen-Peripherie bzw. Schnelldrucker möglich:

READ	Programm einlesen
PUNCH	Gesamtes Programm lochen
PUNCH n	Programm Anweisung n lochen
PUNCH,n	Programm bis Anweisung n lochen
PUNCH m,	Programm ab Anweisung m lochen
PUNCH m,n	Programm von Anweisung m bis Anweisung n lochen
LISP	Gesamtes Programm auf Schnelldrucker listen Parametrierung siehe LIST

Bemerkungen:

Diese Kommandos bewirken Ein- bzw. Ausgabe des Quellprogramms.

Folgende Hardware-Voraussetzungen müssen erfüllt sein:

- für READ: Lochstreifen-Leser (auf Standard-Adresse, Geräte-Nr. d = 2)
- für PUNCH: Lochstreifen-Stanzer (auf Standard-Adresse, Geräte-Nr. d = 2)
- für LISP: Schnelldrucker (auf Standard-Adresse, Geräte-Nr. d = 3)

m,n bezeichnen Anweisungsnummern.

Statement DIM

- m DIM z (p)
- m DIM z (q,p)
- m DIM z1 (p1), z2 (p2), z3 (q3, p3), ...

Mit DIM wird für ein oder mehrere ein- oder zweidimensionale Zahlenfelder z Platz reserviert. Die Dimensionsangabe erfolgt hinter dem Namen des Feldes; sie muß ganzzahlig sein. Die maximal zulässige Feldgröße hängt vom zur Verfügung stehenden Kernspeicherbereich ab.

Beispiel:

```
10 DIM A(9),B(3,2)
```

Für jede Zahl werden 4 Bytes im Speicher reserviert.

Bei der Feldgrößen-Angabe ist zu beachten, daß ab 0 gezählt wird:

DIM A(9) ist ein Feld von 10 Zahlen und belegt 40 byte
DIM B(3,2) ist ein Feld von 4 x 3 Zahlen und belegt 48 byte

Bei zweidimensionalen Feldern (X,Y) ist X die Zeile und Y die Spalte.

Beispiel:

DIM (1,3) reserviert 4 Spalten und 2 Zeilen:

ØØ Ø1 Ø2 Ø3
1Ø 11 12 13

Achtung: Wenn kein DIM-Feld vorhanden ist und im Programm z.B. eine Variable X(A) angegeben wurde, so wird automatisch ein Feld X(9) reserviert.

Gleiche Variablen-Namen für einfache und indizierte Variablen sind zulässig; z.B. sind die Variablen X und X(Ø) nicht identisch.

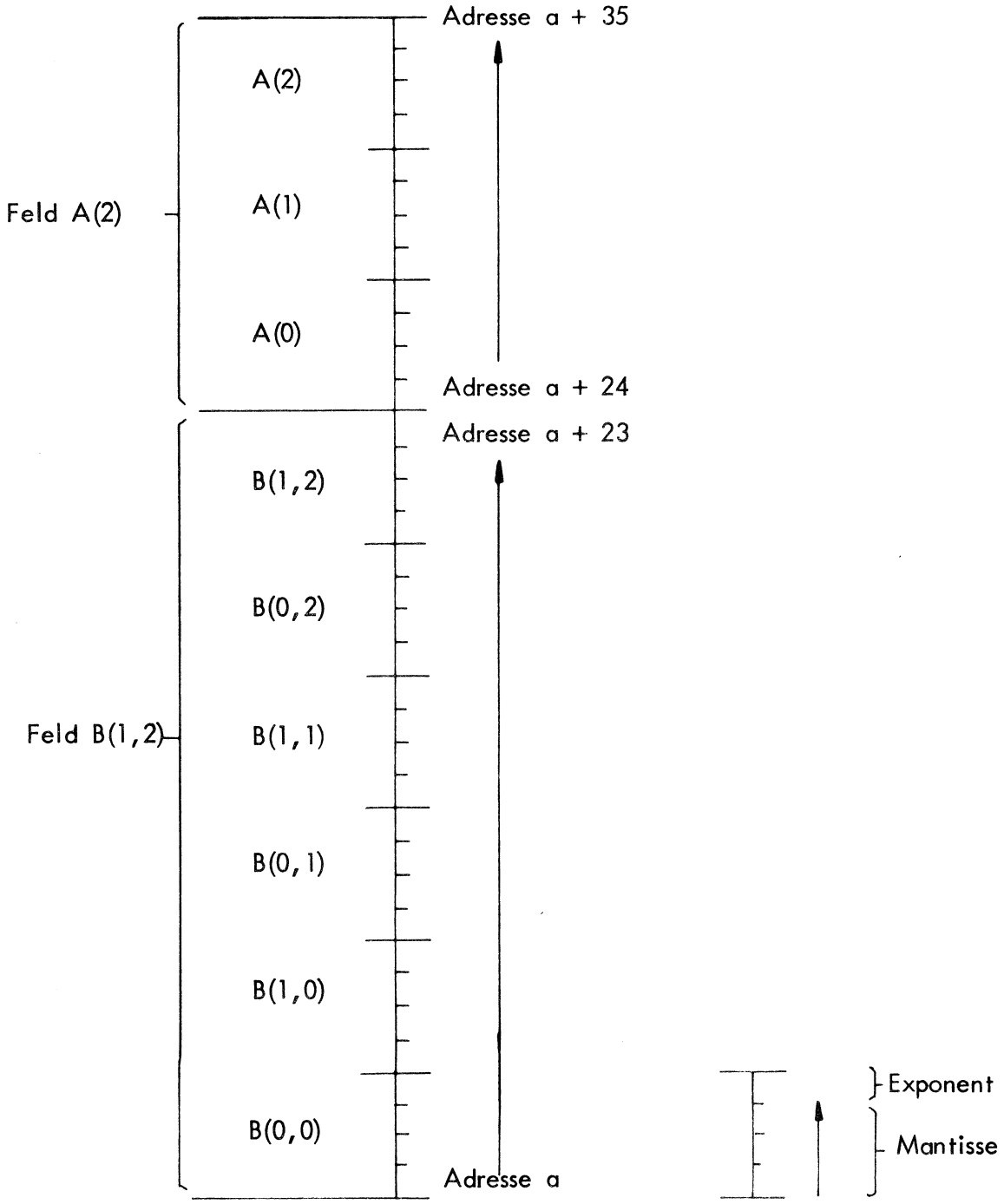
Bereichsüberschreitungen (versuchter Zugriff zu Feldelementen außerhalb der Feldgrenzen durch zu großen Index) werden vom System bei Ausführung des Programms erkannt (Fehler 16).

Speicherbelegung durch DIM:

Die dargestellte Speicherbelegung wird durch beide der folgenden Programmbeispiele bewirkt:

Beispiel 1: 1Ø DIM A(2), B(1,2)

Beispiel 2: 1Ø DIM A(2)
2Ø DIM B (1,2)



Die absolute Speicher-Adresse a ist von System-Konfiguration und Benutzer-Programm abhängig.

Statement CHAR

- m CHAR s(l)
- m CHAR s1 (l1), s2 (l2), ...

Mit CHAR wird für eine oder mehrere Stringvariablen s Platz reserviert. Die Länge l des Strings wird hinter dem Namen der Stringvariablen angegeben; sie entspricht der Anzahl der Zeichen, die in der Variablen Platz finden.

Beispiel:

```
10 CHAR A$(20),R$(10)
```

Für jedes Zeichen wird 1 Byte im Speicher reserviert.

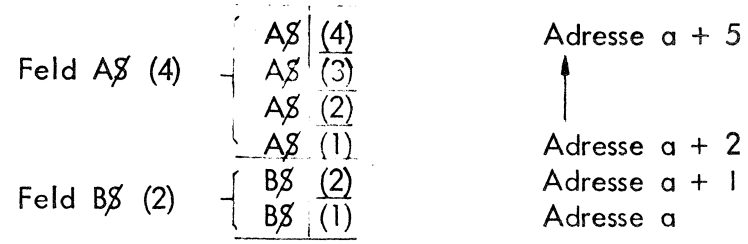
Kommt im Programm eine String-Variable vor, für die nicht mit CHAR Platz reserviert wurde, so werden automatisch 2 Zeichen für sie reserviert.

Speicherbelegung durch CHAR:

Die dargestellte Speicherbelegung wird durch beide der folgenden Programmbeispiele bewirkt:

Beispiel 1: 10 CHAR A$ (4), B$ (2)

Beispiel 2: 10 CHAR A$ (4)
20 CHAR B$ (2)



Die absolute Speicher-Adresse a ist von System-Konfiguration und Benutzerprogramm abhängig.

Statement DEF

- m DEF FNf = u

Zusätzlich zu den in BASIC vorhandenen Standardfunktionen erlaubt DEF die Definition von Benutzer-Funktionen u. Sie werden durch das Symbol FN und einen Namen f, z.B.

FNA
FNOTTO

bezeichnet.

Beispiel:

```
10 DEF FNOTTO(X)=LOG(X)/LOG(10)
```

Diese Funktion ermittelt z.B. für ein gegebenes Argument X den Logarithmus zur Basis 10. Im Programm kann diese Funktion später wie folgt aufgerufen werden:

```
50 LET A=FNOTTO(B)
60 LET V1=FNOTTO((Y-C)/H)
70 PRINT FNOTTO(25)-C,A
```

Die Definition einer Funktion kann auch den Aufruf einer anderen Funktion beinhalten.

Beispiel:

```
10 LET P=3.14159
20 DEF FND(X)=X*P/180
30 DEF FNS(X)=SIN(FND(X))
40 DEF FNC(X)=COS(FND(X))
50 PRINT P,FNC(1),FNS(1),FND(1)
```

Die Funktion FNS (X) ermittelt den Sinus für ein Argument, dessen Wert im Bogenmaß vorliegt. FND (X) besorgt die Umwandlung vom Bogenmaß in Grad.

Statement LET

- m LET v = e
- m LET v1 = e1, v2 = e2, ...

Im Statement LET wird einer Variablen v ein Wert zugewiesen.

Der Wert wird auf der rechten Seite des Gleichheitszeichens durch einen Ausdruck e angegeben. Die rechte Seite wird zunächst berechnet und das Resultat der linken Seite zugewiesen.

LET kann mehrere, durch Kommata getrennte Zuweisungen enthalten.

Beispiele:

```
 10 LET A=5.01
 20 LET DAT1=3/CAE2*N/(AT1+SIN(1.1))
 30 LET OTTO(1,2)=X
 40 LET TEXT$=B$ AND$ C$
 50 LET A$=NOT$ %F6%
 60 LET X1$="AB"
 70 LET X((A$<B$)+5,INT(SIN(A$<B$)))=100
 80 LETX6(Y)=1,A=1,AZ(X,Y)=1
 90 LET A=A$>C$
 95 LET A=1-(2+(3-(4+(5-(6+7)))))
100 LET X$=X$ AND$ %0F%,X$=X$ OR$ %30%
```

Hinweis:

Die Variable v und der Ausdruck e, bzw. der Wert des Ausdruckes müssen vom gleichen Typ sein (Zahl oder String)

Statements DATA, READ und RES

- m DATA c
- m DATA c1, c2, ...

DATA eröffnet eine Liste von Zahlen- oder String-Konstanten c, die später durch READ-Anweisungen bestimmten Variablen zugewiesen werden.

DATA-Statements können an beliebigen Stellen des Programms vorkommen; sie müssen insbesondere nicht vor den READ-Anweisungen stehen.

Die Argumentlisten mehrerer DATA-Anweisungen in einem Programm werden als eine zusammenhängende DATA-Liste behandelt.

Beispiel:

```
10 DATA 1,2,3
20 DATA 4 ,5,6
```

entspricht:

```
10 DATA 1,2,3,4,5,6
```

- m READ v
- m READ v1, v2, ...

READ weist Konstanten der DATA-Liste sequentiell den Variablen v der Argumentliste zu.

Beispiel:

```
10 DATA 8,7,"JA"
15 DATA 3,4,5
20 READ A,B,C$,E,F
30 READ I
```

Wertzuweisung:

A = 8, B = 7, C$ = "JA"
E = 3, F = 4, I = 5

- m RES
- m RES n

Die Anweisung RES (Restore) erlaubt mehrmaliges bzw. gezieltes Lesen von Konstanten aus der DATA-Liste.

Zu Beginn des Programms steht der DATA-Zeiger auf der ersten Konstante der DATA-Liste; bei jeder Wertzuweisung wird er um 1 erhöht.

RES setzt den Zeiger auf die erste Konstante der gesamten DATA-Liste zurück, während RES n ihn auf die erste Konstante der Anweisung n DATA... setzt.

Beispiel:

```
  1 DATA 1,2,3
160 RES 210
170 READ A,B,C
180 RES
190 READ D
210 DATA 4,5,6
```

Wertzuweisung:

A = 4, B = 5, C = 6

D = 1

Statement INPUT

- m INPUT v
- m INPUT v1, v2, ...
- m INPUT "Text", v1, v2, ...

INPUT dient dazu, den Variablen v einer Argumentliste während des Programm-Ablaufs über ein Eingabegerät Werte zuzuweisen, die je nach Variablentyp Zahlen- oder String-Konstanten sind.

Findet das Programm im Programmverlauf ein INPUT-Statement, so meldet es sich mit Fragezeichen (?) auf dem Eingabegerät und wartet auf die Eingabe.

Der Benutzer gibt eine der Variablenliste entsprechende Anzahl von Zahlen (in beliebigem Format) bzw. Text-Strings, getrennt durch Komma, ein. Da eingegebene Text-Strings ebenfalls durch Komma voneinander zu trennen sind, ist Komma (,) als String-Zeichen nicht eingebbar. Hat er sich verschrieben, so kann er wie üblich korrigieren (RUBOUT,←). Die Eingabe wird mit "Wagenrücklauf" (CR) beendet.

Die eingegebenen Werte werden der Reihe nach den Variablen zugeordnet. Gibt der Benutzer eine nicht ausreichende Zahl von Werten ein (vorzeitiges CR), schreibt das Programm "??" und wartet auf Vervollständigung der Eingabe. Zuviel eingegebene Werte dagegen werden nicht berücksichtigt.

Vor der Eingabe von Zahlen oder Strings kann ein Text ausgegeben werden, der hinter INPUT in Anführungszeichen ("...") anzugeben ist. Dies dient als Zusammenfassung der häufig vorkommenden Befehlsfolge PRINT "Text" INPUT Variablenliste.

Durch Eingabe von Control-S (Tasten CTRL und S gleichzeitig, auch X-OFF genannt) statt eines Wertes kann eine Programmbeendigung erreicht werden.

Beispiele:

Das Programm

```
10 INPUT "GIB EIN A1,B1,C1 ",A,B,C
20 LET D=A+B+C
30 GOTO 10
40 END
```

führt bei Ausführung zu folgendem Dialog (Eingaben unterstrichen):

RUN

```
GIB EIN A1,B1,C1 ?1,2⊣
??3⊣
GIB EIN A1,B1,C1 ?⊕
*READY
```

⊣ = Eingabe CR (ASCII-Code: '8D)
⊕ = Eingabe X-OFF (ASCII-Code: '93)

• m INPUT DEV(d), ...

Die Eingabe erfolgt im Normalfall über das Konsolgerät (Geräte-Nummer d = 0). Soll ein anderes Eingabegerät benutzt werden, so ist dessen Geräte-Nummer d durch das Steuerwort DEV(d) in der INPUT-Anweisung zu spezifizieren. Diese Zuordnung bleibt (in der betreffenden Programmebene) für alle folgenden INPUT- (und PRINT-) Anweisungen gültig, bis ein neues Gerät spezifiziert wird.

Das Steuerwort DEV(d) muß in der INPUT-Anweisung vor den weiteren Anweisungen stehen.

Beispiel:

```
10 INPUT "SOLLWERT ", SLWT        ——— Gerät 0
20 INPUT DEV(1), A                —┐  Gerät 1
30 INPUT B, C, X                  —┘
40 INPUT DEV(0), Y                —┐  Gerät 0
50 INPUT Z                        —┘
```

Häufig vorkommende Nummern von Eingabe-Geräten sind:

d = 0	Tastatur am Konsolgerät
d = 1	Streifenleser am Konsolgerät (bei Teletype ASR 33)
d = 2	Schneller Streifenleser

Der Zusammenhang zwischen Geräte-Nummer d und Gerät ist durch das Betriebssystem festgelegt. Für d kann eine Zahlenkonstante, eine Zahlenvariable oder ein beliebiger arithmetischer Ausdruck eingesetzt werden.

Es ist darauf zu achten, daß

- das Steuerwort DEV(d) auch in PRINT-Anweisungen verwendet werden kann, wodurch die in einer vorangegangenen INPUT-Anweisung vorgenommene Geräte-Zuordnung verändert wird,

- bestimmte Nummern d Eingabe- und Ausgabefunktionen desselben Gerätes bzw. derselben Gerätegruppe spezifizieren (z.B. d = 0 für Konsolgerät-Ein/-Ausgabe; d = 2 für schnellen Streifenleser/Streifenlocher).

Statement PRINT

- m PRINT a
- m PRINT a1, a2, ...
- m PRINT

Das Statement PRINT dient zur Ausgabe von Zahlenwerten und Texten auf einem Ausgabegerät. Die auszugebenden Daten a stehen in einer Argumentliste hinter PRINT.

In der Argumentliste können, beliebig gemischt, folgende Arten vorkommen:

Zahlenkonstanten	Beispiel:	2.5
Zahlenvariablen	"	DAT1
Zahlenvariablen, indiziert	"	OTTO(1,2)
Arithmetische Ausdrücke	"	A+B * SIN(C)
Logische Ausdrücke	"	X AND NOT B
Vergleichsausdrücke	"	HIGH = LOW
String-Konstanten (Text)	"	"BERICHT"
String-Konstanten (Steuerzeichen)	"	%8DØA%
String-Variablen	"	TX2Ø

Die Ausgabe erfolgt in der Reihenfolge der Argumente. Diese sind in der Liste durch Komma oder Semikolon voneinander zu trennen.

Komma (,) als Trennzeichen bewirkt, daß die Ausgabe des nächsten Datums an der nächsten Tabulatorspalte beginnt.

Semikolon (;) als Trennzeichen bewirkt dagegen, daß das nächste Datum unmittelbar anschließend ausgegeben wird.

Normalerweise wird nach Ausgabe aller Argumente eine neue Zeile begonnen. Dies kann verhindert werden, indem man hinter das letzte Argument ein Semikolon oder Komma setzt.

PRINT ohne weitere Angaben bewirkt Wagenrücklauf und Zeilenvorschub.

Beispiel:

Das Programm

```
10 LET A=6.4, B=5.1
20 PRINT "WERT A =";A,"WERT B =";B,"ENDE"
30 END
```

bewirkt folgende Ausgabe:

```
RUN
WERT A =  6.4          WERT B =  5.1                    ENDE
*READY
```

Tabulator-Spalten

- m PRINT DEV(d);
- m PRINT DEV(d); ...
- m PRINT ... DEV(d); ...

Die Ausgabe erfolgt im Normalfall über das Konsolgerät (Geräte-Nummer d = 0). Soll ein anderes Ausgabegerät benutzt werden, so ist dessen Geräte-Nummer d durch das Steuerwort DEV(d) in der PRINT-Anweisung zu spezifizieren. Diese Zuordnung bleibt (in der betreffenden Programmebene) für alle folgenden PRINT- (und INPUT-) Anweisungen gültig, bis ein neues Gerät spezifiziert wird.

Das Steuerwort DEV(d) kann in der PRINT-Anweisung öfter vorkommen oder an beliebiger Stelle der Argumentliste stehen. Seine Wirkung beginnt mit dem Augenblick der Abarbeitung in der Liste.

Beispiele:

```
10 PRINT "START"                    - Gerät 0
20 PRINT DEV(2);"AA";DEV(3);        - Gerät 2
30 PRINT 1, 2, 3, 4                 - Gerät 3
40 PRINT DEV(0);
50 PRINT 1, 2, 3, 4                 - Gerät 0
```

Häufig vorkommende Nummern von Ausgabe-Geräten sind

d = 0	Druckwerk bzw. Bildschirm am Konsolgerät
d = 2	Schneller Streifenlocher
d = 3	Schnelldrucker

Der Zusammenhang zwischen Geräte-Nummer d und Gerät ist durch das Betriebssystem festgelegt. Für d kann eine Zahlenkonstante, eine Zahlenvariable oder ein beliebiger arithmetischer Ausdruck eingesetzt werden.

Es ist darauf zu achten, daß

- das Steuerwort DEV(d) auch in INPUT-Anweisungen verwendet werden kann, wodurch die in einer vorangegangenen PRINT-Anweisung vorgenommene Zuordnung verändert wird,

- bestimmte Nummern d Eingabe- und Ausgabefunktionen desselben Gerätes bzw. derselben Gerätegruppe spezifizieren (z.B. d = 0 für Konsolgerät-Ein/-Ausgabe; d = 2 für schnellen Streifenlocher/Streifenleser).

- m PRINT TAB(e);
- m PRINT TAB(e); ...
- m PRINT ... TAB(e); ...

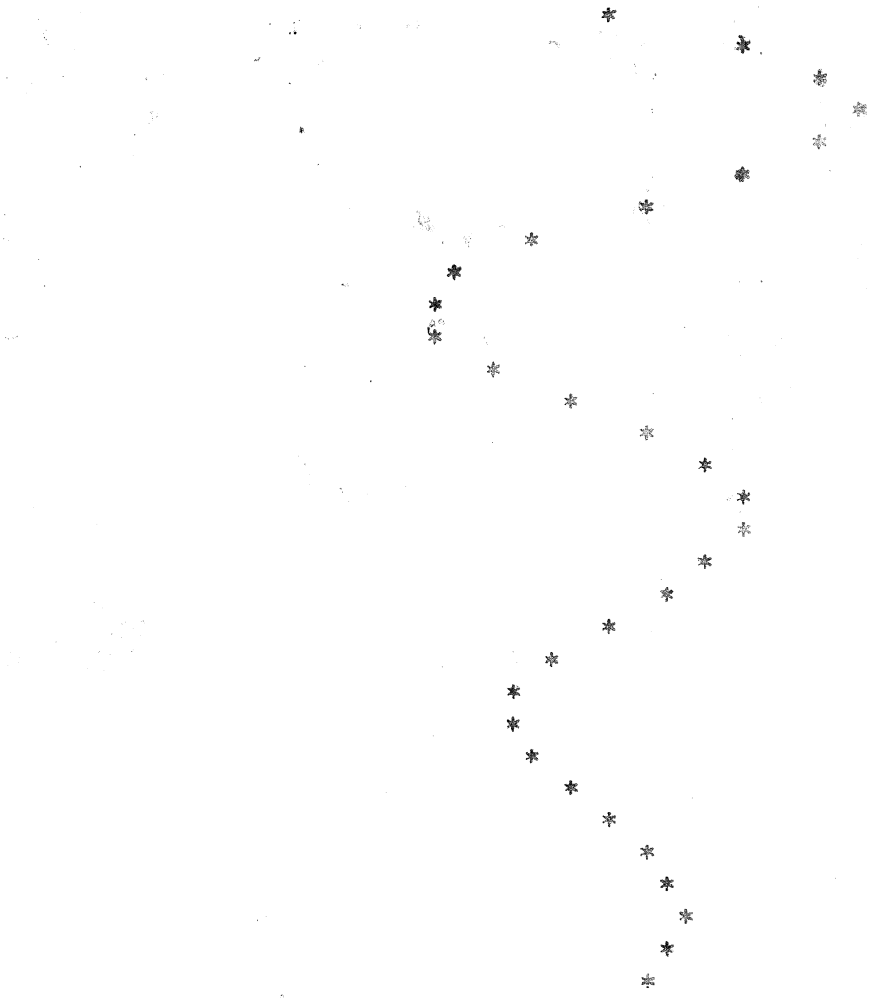
Das Steuerwort TAB in einer PRINT-Anweisung bewirkt, daß der folgende Wert um so viel Spalten weiter rechts ausgegeben wird, wie der Wert des arithmetischen Ausdrucks e ergibt. Dadurch kann z.B. der Schreibkopf eines Druckers beliebig positioniert werden. Die Positionierung geht stets von der augenblicklichen Stellung aus nach rechts.

Das Steuerwort TAB(e) kann in der PRINT-Anweisung an beliebiger Stelle der Argumentliste stehen.

Beispiel (graphische Ausgabe einer Funktion mittels Schreibkopf-Positionierung):

```
100 REM    BEISPIEL TAB(X)
110 DEF FNF(X)=SIN(X)*EXP(-.1*X)
120 FOR I=0 TO 15 STEP .5
130 PRINT TAB(30.5+15*FNF(I));"*"
140 NEXT I
150 END

*READY
RUN
```



```
*READY
```

- m PRINT FMT(f);
- m PRINT FMT(f); ...
- m PRINT ... FMT(f); ...

Das Steuerwort FMT(f) spezifiziert ein Format f für die Ausgabe von Zahlen durch PRINT-Anweisungen.

Eine Format-Spezifikation bleibt (in der betreffenden Programmebene) so lange gültig, bis sie durch eine andere ersetzt wird.

Das Steuerwort FMT(f) kann in der PRINT-Anweisung allein vorkommen oder an beliebiger Stelle der Argumentliste stehen. Seine Wirkung beginnt mit dem Augenblick der Abarbeitung in der Liste.

Es gibt drei verschiedene Typen von Format-Spezifikationen:

- Automatisches Format: FMT(A)
- Festes F-Format: FMT(Fw.d)
- Festes E-Format: FMT(Ew.d)

Das automatische Format ist die Standard-Ausgabeform für Zahlen; es ist vom Augenblick des Programmstarts durch RUN wirksam, bis eine andere Format-Spezifikation FMT(...) im Programm erkannt und abgearbeitet wird. Danach kann es jederzeit durch FMT(A) wiederhergestellt werden. Beim automatischen Format paßt sich die Darstellung der Art und Größe des auszugebenden Zahlenwertes an:

- Ganzzahlige Werte mit Beträgen kleiner als 10^6 werden 1- bis 6-stellig eingegeben; vor negativen Zahlen steht ein Minuszeichen (-). Beispiele:

  0
  1
  -1
  325
  -471.1
  999999
  -99999

  Zu beachten ist hierbei, daß die Ausgabe "linksbündig" erfolgt, d.h. an der Stelle beginnt, wo der Schreibkopf im Augenblick steht. Die "Feldweite", d.h. die Anzahl der ausgegebenen Zeichen, ist variabel, da sie von der Größe der Zahl und ihrem Vorzeichen abhängt.

- Nicht-ganzzahlige Werte im Bereich zwischen 10^{-6} und 10^{+5} werden in einer Dezimalpunkt-Darstellung (variables F-Format) ausgegeben. Beispiele:

  .5
  .123456
  -.000001
  99999.9

  Auch hier handelt es sich um "linksbündige" Ausgabe mit variabler Feldweite.

- Alle übrigen Zahlenwerte werden in einer Gleitkomma-Darstellung (Standard-E-Format) ausgegeben. Beispiele:

.100000E 10	(= 10^9)
.123456E-05	(= 0.0000012345)
-.500000E-09	(= -0.0000000005)

Das feste F-Format ist eine Dezimalpunkt-Darstellung, die durch das Steuerwort FMT(Fw.d) spezifiziert wird. Darin bedeutet w die Feldweite, d.h. die Gesamt-Zahl der auszugebenden Zeichen einschließlich führender Leerzeichen (␣), Vorzeichen und Dezimalpunkt, während d die Anzahl der Stellen nach dem Dezimalpunkt angibt. Die Spezifikatoren w und d können nach Bedarf gewählt werden; jedoch sind folgende Einschränkungen zu beachten:

$2 \leq w \leq 15$	(Feldweite minimal 2, maximal 15 Zeichen)
$0 \leq d \leq 6$	(maximal 6 Stellen nach Dezimalpunkt)
$w \geq d+2$	(Feldweite wenigstens 2 Zeichen größer als d)

Ausgabebeispiele für die Formatspezifikation FMT(F8.3):

```
-999.999
␣␣␣-.123
␣␣␣␣.001
␣␣20.000
```

Paßt eine Zahl nicht in das angegebene Format, so wird sie im automatischen Format ausgegeben.

Das feste E-Format ist eine Gleitkomma-Darstellung mit Mantisse und Exponent zur Basis 10; sie wird durch das Steuerwort FMT(Ew.d) spezifiziert. Darin bedeutet w die Feldweite einschließlich führender Leerstellen, Vorzeichen der Mantisse, Dezimalpunkt, Buchstabe "E" und Vorzeichen des Exponenten, während d die Anzahl der Stellen nach dem Dezimalpunkt angibt. Für w und d sind folgende Einschränkungen zu beachten:

$7 \leq w \leq 15$	(Feldweite minimal 7, maximal 15 Zeichen)
$1 \leq d \leq 6$	(minimal 1, maximal 6 Stellen nach Dezimalpunkt)
$w \geq d + 6$	(Feldweite wenigstens 6 Zeichen größer als d)

Ausgabe-Beispiele für die Formatspezifikation FMT(E10.3):

```
␣␣.100E 02
␣-.123E-09
```

Programm-Beispiel mit verschiedenen Format-Spezifikationen für die Ausgabe der Zahl -1.234:

```
10 LET A=-1.234
20 PRINT "FX.0-FORMAT"
30 PRINT "*";FMT(F2.0);A;"*"
40 PRINT "*";FMT(F3.0);A;"*"
50 PRINT "*";FMT(F4.0);A;"*"
60 PRINT "*";FMT(F5.0);A;"*"
70 PRINT "*";FMT(F6.0);A;"*"
80 PRINT "*";FMT(F15.0);A;"*"
90 PRINT
100 PRINT "FX.Y-FORMAT"
110 PRINT "*";FMT(F3.1);A;"*"
120 PRINT "*";FMT(F4.1);A;"*"
130 PRINT "*";FMT(F5.1);A;"*"
140 PRINT "*";FMT(F15.1);A;"*"
150 PRINT "*";FMT(F15.2);A;"*"
160 PRINT "*";FMT(F15.3);A;"*"
170 PRINT "*";FMT(F15.4);A;"*"
180 PRINT "*";FMT(F15.5);A;"*"
190 PRINT
200 PRINT "EX.Y-FORMAT"
210 PRINT "*";FMT(E7.1);A;"*"
220 PRINT "*";FMT(E8.1);A;"*"
230 PRINT "*";FMT(E9.1);A;"*"
240 PRINT "*";FMT(E15.1);A;"*"
250 PRINT "*";FMT(E15.2);A;"*"
260 PRINT "*";FMT(E15.3);A;"*"
270 PRINT "*";FMT(E15.4);A;"*"
280 PRINT "*";FMT(E15.5);A;"*"
290 PRINT "*";FMT(E15.6);A;"*"
300 END

*READY
RUN
FX.0-FORMAT
*-1*
* -1*
*  -1*
*   -1*
*    -1*
*                -1*

FX.Y-FORMAT
*-1.234 *
*-1.2*
* -1.2*
*             -1.2*
*            -1.23*
*           -1.234*
*          -1.2340*
*         -1.23400*

EX.Y-FORMAT
*-.1E 01*
* -.1E 01*
*  -.1E 01*
*         -.1E 01*
*        -.12E 01*
*       -.123E 01*
*      -.1234E 01*
*     -.12340E 01*
*    -.123400E 01*

*READY
```

Statement GOTO

- m GOTO n
- m GOTO e OF n1, n2, ...

Mit GOTO wird der normale Programmablauf verändert, indem das Programm mit der Anweisung n fortgesetzt wird (statt mit der auf m folgenden Nummer).

Beispiel:

```
10 LET A=6
20 LET B=7
30 GOTO 60
40 PRINT A, B, C
50 END
60 LET C=A+B
70 GOTO 40
```

entspricht:

```
10 LET A=6
20 LET B=7
30 LET C=A+B
40 PRINT A, B, C
50 END
```

In der Form GOTO...OF wird anstelle eines festen ein berechneter Sprung im Programmablauf erzeugt.

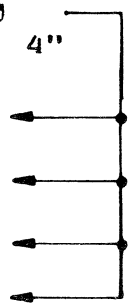
Je nachdem, ob der Wert des Ausdrucks e gleich 1, 2, ... ist, wird das Programm an der Stelle n1, n2, ... fortgesetzt.

Ist der Wert des Ausdrucks kleiner als 1 oder größer als die Anzahl der hinter OF angegebenen Statement-Nummern, so wird die auf m folgende Anweisung ausgeführt.

Für e kann eine Zahlenvariable oder ein beliebiger arithmetischer Ausdruck eingesetzt werden. Bei nicht ganzzahligen Werten wird der Ganzzahl-Anteil für die Berechnung des Sprungzieles herangezogen. Wegen eventueller Rundungsfehler sollten jedoch solche Ausdrücke vermieden werden, bei denen nicht-ganzzahlige Werte zu erwarten sind.

Beispiel:

```
10 INPUT A
20 GOTO A OF 100, 120, 140, 160
30 PRINT "A < 1    ODER    A > 4"
40 GOTO 10
100 PRINT "1", A
110 GOTO 10
120 PRINT "2", A
130 GOTO 10
140 PRINT "3", A
150 GOTO 10
160 PRINT "4", A
170 GOTO 10
```



(A<1, A>4)

(A = 1)

(A = 2)

(A = 3)

(A = 4)

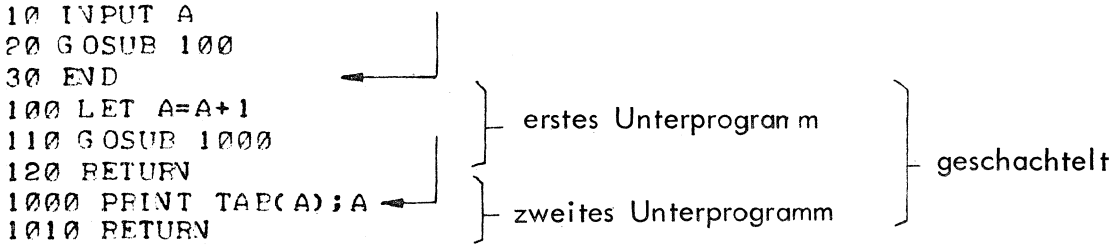
Statements GOSUB und RETURN

- m GOSUB n
- m GOSUB e OF n1, n2, ...

Die Anweisung GOSUB bewirkt den Sprung in ein Unterprogramm, das mit der Anweisungsnummer n beginnt. Nach dem Rücksprung aus dem Unterprogramm (durch RETURN) wird das Programm mit der auf m folgenden Anweisung fortgesetzt.

Unterprogramme können mehrfach und von beliebigen Stellen des Programms aufgerufen werden. Von Unterprogrammen können weitere Unterprogramme aufgerufen werden; sie dürfen beliebig tief geschachtelt sein. (Eine Grenze setzt lediglich die vorhandene Speicherkapazität.

Beispiel:



Die Form GOSUB...OF beschreibt einen berechneten Unterprogramm-Sprung; die Berechnung des Sprungzieles ist dieselbe wie bei GOTO...OF.

Beispiel:

```
30 GOSUB XY OF 110,120,130,140
```

- m RETURN

RETURN bewirkt den Rücksprung aus dem Unterprogramm. Ein Unterprogramm kann mehrere, muß jedoch mindestens eine Rücksprung-Anweisung enthalten.

Statement IF

- m IF e THEN n

Das Statement IF beschreibt eine bedingte Verzweigung.

Ist der Wert des Ausdrucks e ungleich Null, so wird das Programm mit der Anweisung n fortgesetzt, deren Nummer n hinter THEN steht; andernfalls mit der auf m folgenden Anweisung.

Als Ausdruck e ist jede Variable sowie jeder arithmetische und logische Ausdruck zulässig.

Besonders gebräuchlich sind auch Vergleichs-Ausdrücke mit den Operatoren

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

in der Form:

m IF a op b THEN n

Die Verzweigung zu n erfolgt hier, wenn die Vergleichsbedingung zwischen a und b erfüllt ist.

Beispiele:

```
10 IF A=6 THEN 100
```
Wenn A = 6, ist Sprung nach 100

```
10 IF A+6<=7 THEN 100
```
Wenn A+6 oder = 7 ist Sprung nach 100

```
10 IF A$="Y" AND A=7 THEN 100
```
Wenn die Stringvariable A$ gleich "Y" ist und die Variable A den Wert 7 hat, dann Sprung nach 100

```
10 IF SIN(X)<=COS(Y) AND SIN(Z)#.5 THEN 100
```

Sprung nach 100, wenn SIN(x) < = COS(y) und SIN(z) # .50

```
10 INPUT A, B
20 IF A=B OR A<6 OR B>6 THEN 50
30 PRINT "30", A, B, A=B, A<6, B>6
40 GOTO 10
50 PRINT "50", A, B, A=B, A<6, B>6
60 GOTO 10
```

Statements FOR und NEXT

- m FOR x = a TO b
- m FOR x = a TO b STEP c

- n NEXT x

Die Statements FOR und NEXT beschreiben eine Programmschleife.

Sie wird eröffnet mit dem Statement FOR, in dem einem Laufindex x ein Anfangswert a zugewiesen und dessen Endwert b bestimmt wird. Außerdem ist darin die Angabe der Schrittweite c möglich; fehlt diese Angabe, so wird mit der Schrittweite 1 gearbeitet.

Abgeschlossen wird die Schleife durch das Statement NEXT mit Angabe des Laufindex' x.

Der Laufindex x ist eine einfache Zahlenvariable, die nur für die Schleife definiert ist. Als Anfangswert, Endwert und Schrittweite können Zahlenkonstanten, -Variablen oder beliebige arithmetische Ausdrücke verwendet werden.

Zu Beginn des ersten Durchlaufs wird dem Laufindex x der Anfangswert a zugewiesen. Nach jedem Durchlauf wird x um die Schrittweite c (bzw. um 1) erhöht. Dies geschieht so oft, bis x größer als der Endwert b geworden ist; damit wird die Schleife beendet und die auf NEXT folgende Anweisung ausgeführt.

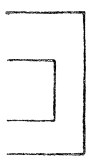
Programmschleifen dürfen beliebig ineinander verschachtelt sein; jedoch ist überkreuzte Schachtelung verboten.

Zu jedem FOR x darf nur ein NEXT x gehören; dies muß eine höhere Anweisungs-Nummer haben als das FOR x.

FOR-Schleifen müssen vom Programm durch Ausführung des Statements NEXT beendet werden; Herausspringen aus der Schleife ohne Rücksprung in die Schleife ist nicht erlaubt.

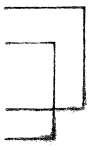
Beispiele:

```
10 FOR I=1 TO 10 STEP 2
20 FOR K=A TO 4*B+1
30 LET A=A+M(I,K)
40 NEXT K
50 NEXT I
```



erlaubte Schachtelung

```
10 FOR I=1 TO 10 STEP 2
20 FOR K=A TO 4*B+1
30 LET A=A+M(I,K)
40 NEXT I
50 NEXT K
```



verbotene Schachtelung

Statement CALL

- m CALL z
- m CALL z (y)
- m CALL z (y1, y2, ...)

Das Statement CALL ruft eine Systemprozedur auf, die den Namen z hat. Die Prozedur ist Bestandteil des Betriebssystems bzw. wird bei der Systemgenerierung diesem hinzugefügt. Sie wird nach dem Aufruf wie ein Unterprogramm ausgeführt; danach wird das Programm mit der auf m folgenden Anweisung fortgesetzt.

Es dürfen nur solche Prozeduren gerufen werden, die im Betriebssystem unter dem betreffenden Namen enthalten sind.

In Klammern können ein bis vier Parameter y angegeben sein. Sie werden der Prozedur vom Programm übergeben und dienen zum Transfer von Werten aus dem Programm zur Prozedur bzw. in umgekehrter Richtung. Anzahl und Typ der Parameter sind für jede Prozedur festgelegt (siehe Abschnitt "Systemprozeduren").

Die verfügbaren Systemprozeduren sind in späteren Abschnitten angegeben.

Beispiele:

```
10 CALL HOME
20 CALL SYMB(A,B$(6,7))
30 CALL CREA(U,"NAME",LAEN)
40 CALL GFB(ARB,A(X),SEKA,SEKL)
```

Statement END

- m END

Die Anweisung END kennzeichnet das logische Ende des gesamten Programms auf beliebiger Ebene.

Das System kehrt bei Ausführung von END in die Bedienungs-Betriebsart (command mode) zurück und meldet sich auf dem Konsolgerät mit der Nachricht

```
* READY
```

Ein neuer Start des Programms erfordert Eingabe des Kommandos RUN.

Die Anweisung END kann beliebig oft im Programm vorkommen. Es ist üblich, aber nicht notwendig, sie außerdem stets als letztes Statement anzugeben.

Beispiel:

```
1250 END
...
9999 END
```

Platten als Hintergrundspeicher

Plattenspeicher erhöhen die Leistungsführung eines Computer-Systems beträchtlich, da Programme und Daten in großem Umfang für das System auf der Platte bereitgehalten werden können.

Im folgenden sind die Funktionen beschrieben, die BASEX bei plattenorientierten Systemen mit dem Computer DIETZ 621 enthält. Dabei handelt es sich um die DIETZsysteme 621 C, D und E, die unter einem plattenorientierten Betriebssystem (z.B. DBOS) laufen.

Jedes Plattenlaufwerk hat eine Nummer u:

- DIETZsystem 621 C	u = 0	DIETZdisk	0.25 MB (Systemplatte)
	u = 4	DIETZdisk	0.25 MB
- DIETZsystem 621 D	u = 0	Wechselplatte	2.4 MB (Systemplatte)
	u = 4	DIETZdisk	0.25 MB
- DIETZsystem 621 E	u = 0	Wechsel-+ Festplatte	9.6 MB (Systemplatte)
	u = 4	DIETZdisk	0.25 MB

Bis zu 3 weitere Laufwerke vom Typ der Systemplatte sind anschließbar; die Nummern u lauten dann 1, 2 und 3.

Im folgenden sind die BASEX-Funktionen geschildert, die sich im Zusammenhang mit dem Plattensystem ergeben (Ablegen, Laden und Segmentieren von Programmen; Verwaltung und Zugriff zu Dateien).

Bemerkung:

Mit SAVE und INITIALIZE abgelegte Programme sind stets auf der Systemplatte enthalten!

Kommandos SAVE, LOAD und KILL

Bei plattenorientierten Systemen können das gesamte BASEX-Programm oder Teile davon, insbesondere auch die Teile segmentierter Programme, durch die Kommandos SAVE bzw. LOAD auf dem Plattenspeicher abgelegt bzw. aus ihm in den Kernspeicher geladen werden. SAVE und LOAD entsprechen den Kommandos PUNCH und READ bei Lochstreifen-orientierten Systemen.

Jeder so behandelte Programmteil ist mit einem Namen zu versehen (Buchstabe, u.U. gefolgt von bis zu 3 Buchstaben oder Ziffern), durch den er identifiziert wird; der Programmteil wird als Datei geführt.

Die Kommandos SAVE und LOAD können wie folgt angewandt werden:

SAVE Name	Ablegen gesamtes Programm
SAVE Name n	Ablegen Anweisung n
SAVE Name,n	Ablegen bis Anweisung n
SAVE Name m,	Ablegen ab Anweisung m
SAVE Name m,n	Ablegen von Anweisung m bis Anweisung n
LOAD Name	Laden Programm

Die Programme werden beim Ablegen (SAVE) von Zwischen- in Quellsprache, beim Laden (LOAD) von Quell- in Zwischensprache übersetzt.

Ein mit LOAD geladenes Programm kann wie üblich in Dialog verändert, mit LIST gelistet, durch RUN gestartet oder mit SAVE wieder abgelegt werden.

Bemerkung:

- Bei Ablegen von Namen auf der Platte werden die Namen im Namensverzeichnis mit B-Name abgelegt

- Führt das Kommando SAVE Name zu der Fehlermeldung 40, so kann dieser Name durch die Kommandos KILL Name und erneutem SAVE Name abgelegt werden.

Beispiel (Eingaben unterstrichen):

```
10 INPUT A
20 PRINT SQR(A)
30 END
SAVE BSP1

*READY
LOAD BSP1

*READY

5 PRINT "QUADRATWURZEL"
LIST
5 PRINT "QUADRATWURZEL"
10 INPUT A
20 PRINT SQR(A)
30 END

*READY

RUN

QUADRATWURZEL
?4
2

*READY
SAVE BSP1

*READY

 KILL BSP1
 * READY
```

Mit SAVE wird auf dem Plattenspeicher unter dem eingegebenen Namen eine Datei eröffnet, in der das Programm abgelegt ist. Durch das Kommando

KILL Name Programm-Datei löschen

wird der Programm-Name im Datei-Inhaltsverzeichnis gelöscht; das Programm kann nicht mehr geladen werden.

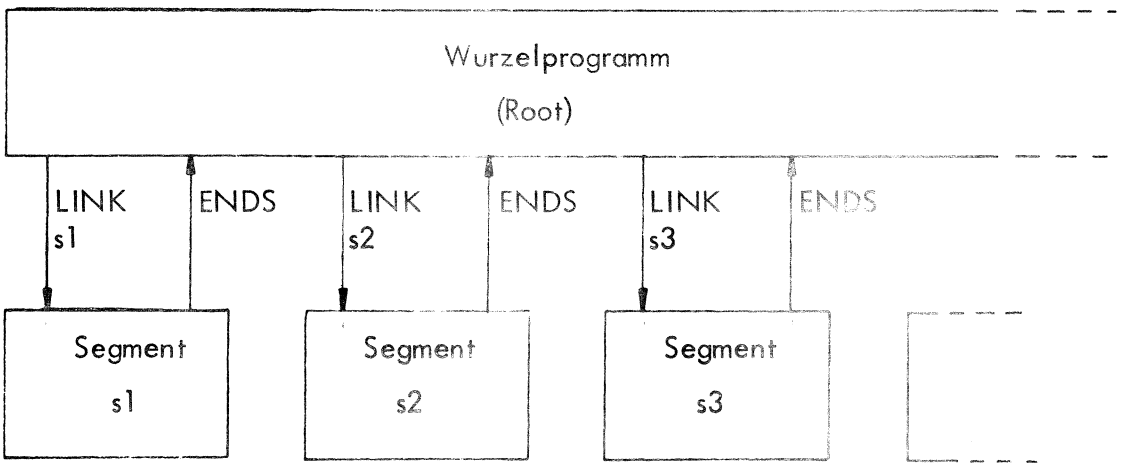
Segmentierung von Programmen

BASEX-Programme, deren Größe die verfügbare Kernspeicher-Kapazität übersteigt, können segmentiert, d.h. in einzelne nacheinander in den Speicher geladene Teile zerlegt werden. Voraussetzung ist, daß das System einen Plattenspeicher besitzt und von einem plattenorientierten Betriebssystem (z.B. DBOS) unterstützt wird.

Zu beachten ist, daß nur ein Overlay-Bereich im Kernspeicher existiert, in den die Programmsegmente nacheinander geladen werden. Zur gleichen Zeit kann sich daher nur ein Segment dort befinden und abgearbeitet werden. Daraus ergibt sich:

- Segmentierte Programme haben eine einstufige Baumstruktur; sie bestehen aus einem Kernspeicher-residenten Wurzelprogramm (Root) und mehreren, von ihr nacheinander aufgerufenen Segmenten.
- Bei Systemen, die im Multiprogramming laufen (typisch für BASEX-Anwendungen), ist im allgemeinen nur das Programm einer Benutzer-Ebene segmentierbar. Laufen in mehreren Ebenen segmentierte Programme, so muß der Overlay-Bereich vom Benutzer-Programm verwaltet werden.
- Insbesondere die Programme, die in den Zeitauftrags- und Interrupt-Ebenen laufen, dürfen nicht segmentiert werden, sondern müssen Bestandteil der Root sein.

Segmentierungs-Struktur:

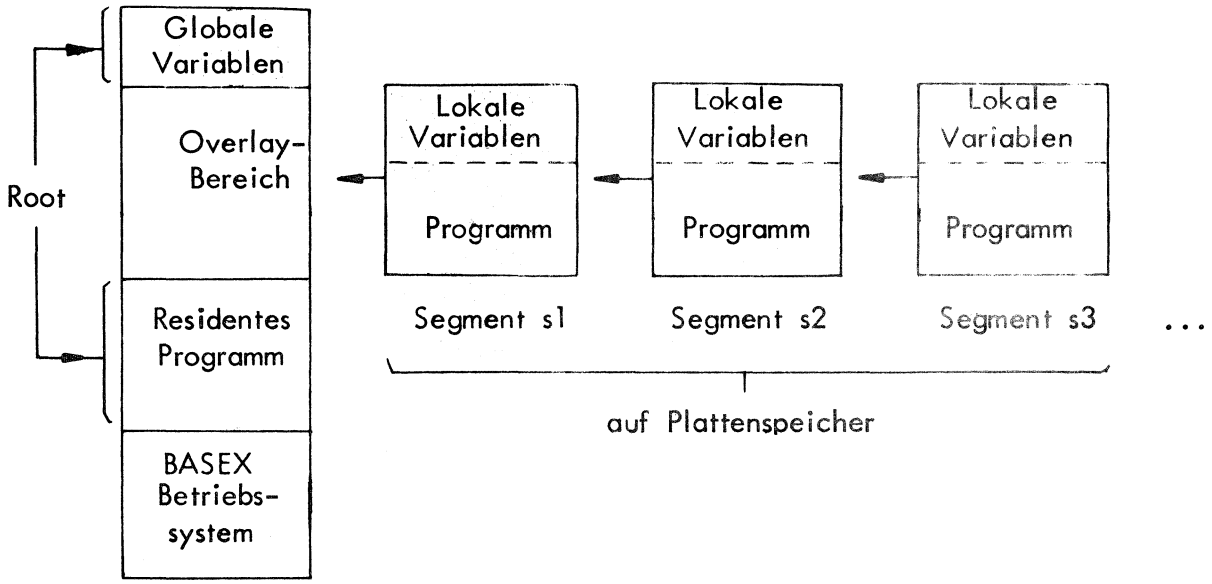


Jedes Segment wird durch einen Namen identifiziert (Buchstabe, u.U. gefolgt von bis zu 3 Buchstaben oder Ziffern). Es wird im Wurzelprogramm durch eine LINK-Anweisung mit Angabe des Namens aufgerufen; die Rückkehr aus dem Segment in die Wurzel bewirkt die ENDS-Anweisung.

Alle Variablen, deren Namen in der Wurzel vorkommen, sind global, d.h. sowohl von der Wurzel als auch von allen Segmenten aus zugreifbar. Dagegen sind alle die Variablen lokal, d.h. nur in einem Segment definiert und innerhalb dieses zugreifbar, die nicht mit Variablen-Namen der Wurzel übereinstimmen. Für lokale Variablen können in verschiedenen Segmenten die gleichen Namen verwendet werden; jedoch haben sie dann keinerlei Bezug zueinander.

Zu beachten ist, daß die Anweisungsnummern der Segmente größer sein müssen als die höchste Anweisungsnummer der Wurzel; jedoch können für verschiedene Segmente die gleichen Anweisungsnummern verwendet werden.

Kernspeicher-Aufteilung bei segmentierten Programmen (vereinfacht):



Alle Segmente und die Wurzel werden unter deren Namen zusammenhängend auf der Platte abgelegt und sind direkt vom DBOS her ausführbar.

Kommando INITIALIZE

Die einzelnen Teile eines segmentierten Programms sind, bevor sie zum ersten Mal komplett zur Ausführung gebracht werden, im Dialog zu verketten. Dabei ist vorausgesetzt, daß die Programmteile durch SAVE-Kommandos unter ihren Namen auf dem Plattenspeicher abgelegt sind.

Die Verkettung beginnt mit dem Kommando

INITIALIZE

worauf das System mit

ROOT?

nach dem Namen des Wurzelprogramms fragt. Der Bediener gibt den Namen ein (mit Wagenrücklauf danach). Der entsprechende Programmteil wird geladen und bleibt im folgenden im Kernspeicher.

Danach fragt das System mit

SEGMENT?

nach dem Namen des ersten Segmentes, den der Bediener eingibt (mit Wagenrücklauf danach); das Segment wird geladen, in Zwischensprache auf einer besonderen Datei des Plattenspeichers abgelegt und mit seiner Platten-Adresse im System vermerkt.

Es folgt wieder die Frage

SEGMENT?

mit der Beantwortung durch den nächsten Segment-Namen. Dies wiederholt sich für jedes weitere Segment, bis der Bediener mit Eingabe des Zeichens "**#** " den Verkettungs-Dialog beendet. Das System kehrt in das Basis-Betriebssystem (z.B. DBOS) zurück.

Beispiel (Eingaben unterstrichen):

```
INITIALIZE

ROOT? ADAM
SEGMENT? SEG 1
SEGMENT? SEG 2
SEGMENT? SADA
SEGMENT? A5A
SEGMENT? #

*DBOS
```

Hinweis:

Auch unsegmentierte Programme sind, wenn sie in Zwischensprache abgelegt und später vom Basis-Betriebssystem (z.B. DBOS) unter ihren Namen aufgerufen werden sollen (s. Kommandos SAVE und LOAD), dem INITIALIZE-Dialog zu unterwerfen. Der Programm-Name wird nach der Frage ROOT? eingegeben; nach der ersten Frage SEGMENT? wird der Dialog durch Eingabe von **#** beendet.

Statements LINK und ENDS

- m LINK Name

Die Anweisung LINK ruft das Programmsegment mit dem eingegebenen Namen auf.

Das Segment wird vom Plattenspeicher in den Overlay-Bereich des Kernspeichers geladen, und das Programm wird mit der ersten Anweisung des Segmentes fortgesetzt.

LINK-Anweisungen dürfen nur im Wurzelprogramm (Root) stehen, nicht dagegen in Segmenten.

Beispiel:

```
200 LINK ASE2
```

- m ENDS

Die Anweisung ENDS (End Segment) beendet den Ablauf des Programms im Segment.

Die Kontrolle wird an das Wurzelprogramm (Root) zurückgegeben, das mit der Anweisung fortgesetzt wird, die auf den Segment-Aufruf durch LINK folgt.

ENDS-Anweisungen dürfen nur in Segmenten vorkommen.

Beispiel:

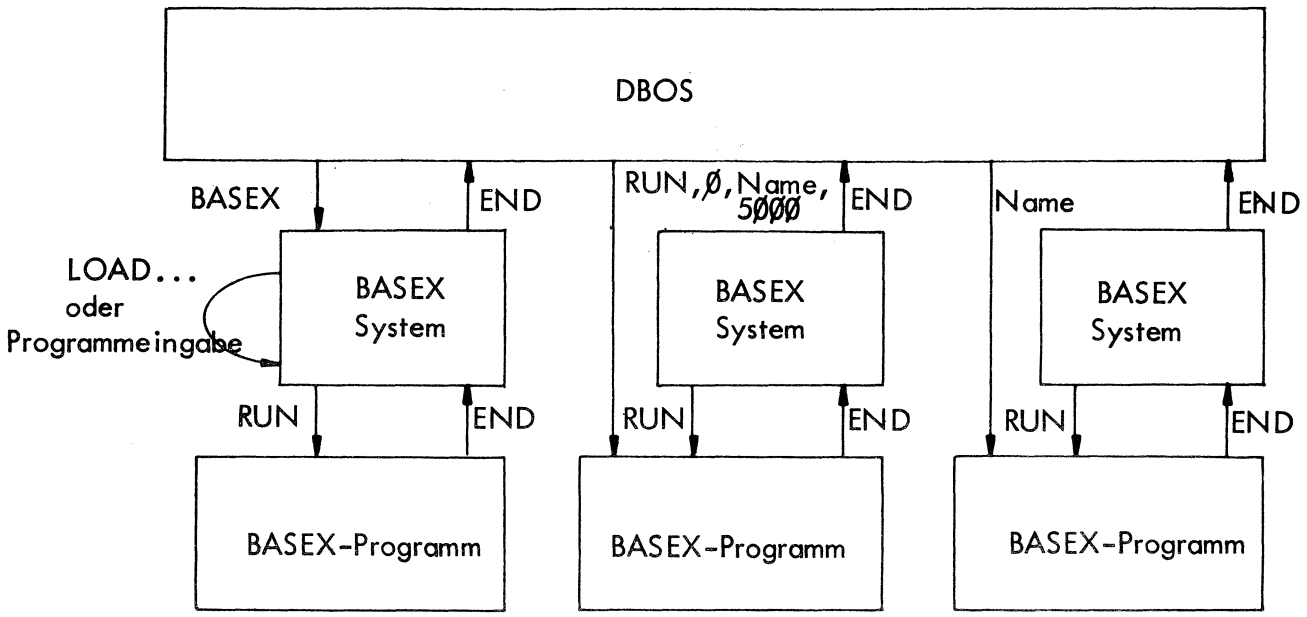
```
855 END S
```

DBOS in BASEX

DBOS (Disk Based Operating System) ist ein Betriebssystem für plattenorientierte Systeme mit dem Computer DIETZ 621 (DIETZsystem 621).

Seine Hauptfunktionen sind die Verwaltung von Plattenspeicher-Dateien und der Zugriff zu ihnen. DBOS-Dateien bestehen aus einem oder mehreren Sektoren von je 128 byte Länge.

Der Übergang von der DBOS-Kontrolle in die von BASEX bzw. BASEX-Programmen geschieht durch DBOS-Kommandos:



Fall 1:
Aufruf BASEX-System

Fall 2:
Aufruf BASEX-Programm über RUN und Namen

Fall 3:
Aufruf BASEX-Programm über Namen direkt

Fall 1: Der Bediener gibt BASEX ein. Der Interpreter wird geladen, und das System befindet sich im BASEX-Bedienungsbetrieb. Nun kann über das Kommando LOAD... ein vorhandenes BASEX-Quellprogramm geladen oder ein Programm eingegeben werden. Auch alle übrigen Möglichkeiten des BASEX-Bedienungsbetriebes bestehen.

Durch das Kommando RUN wird das Programm zur Ausführung gebracht.

Bei Ausführung der END-Anweisung kehrt das System in den BASEX-Bedienungsbetrieb zurück.

Das Kommando END gibt die Kontrolle an DBOS zurück.

Fall 2: Der Bediener ruft mit RUN, Ø, Name, 5ØØØ ein vorhandenes (bereits vorher im INITIALIZE-Dialog in Zwischensprache abgelegtes) BASEX-Programm auf, das einschließlich Interpreter geladen wird. Das Programm wird sofort zur Ausführung gebracht.

Alle übrigen Funktionen entsprechen Fall 1.

Als Name ist der Programmname, bei segmentierten Programmen der Name des Wurzelprogramms (Root) einzugeben.

Fall 3: Wie Fall 2, jedoch wird das BASEX-Programm einschließlich Interpreter direkt, d.h. nur durch Eingabe seines Namens aufgerufen und zur Ausführung gebracht.

Voraussetzung ist, daß das Programm vorher bereits im INITIALIZE-Dialog abgelegt und durch das DBOS-Kommando

PROT,Ø,Name,11

zum Direktaufruf freigegeben (und gleichzeitig geschützt) worden war.

Die Dateiverwaltungs- und Dateizugriffsfunktionen von DBOS werden in BASEX von Systemprozeduren durchgeführt, die mit CALL aufgerufen werden:

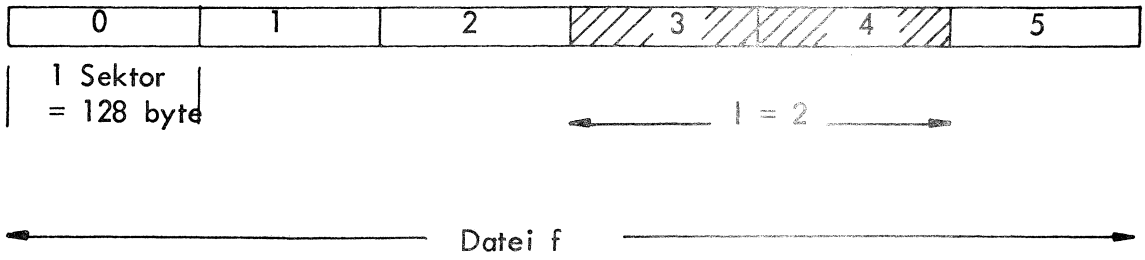
- Dateiverwaltung:

```
CALL CREA (u,f,l)
CALL KILL (u,f)
CALL ALTR (u,f1,f2)
CALL LENG(u,f,l)
CALL PROT (u,f,p)
```

- Dateizugriff:

```
CALL OPEN(u,f,w)
CALL CLSE (w)
CALL GFB  (w,a,s,l)
CALL PFB  (w,a,s,l)
CALL GFBS (w,a,s,l)
CALL PFBS (w,a,s,l)
```

Beispiel: Datei mit 6 Sektoren Länge. Zugriff auf Sektoren 3 und 4 (s = 3, l = 2).



Als Optionen sind folgende Systemprozeduren vorgesehen, die Lesen und Schreiben von Feldern erlauben, die nicht mit Sektoren übereinstimmen müssen:

```
CALL GF  (w,a,i,l)
CALL PF  (w,a,i,l)
CALL GFS(w,a,i,l)
CALL PFS (w,a,i,l)
```

Die DBOS-Prozeduren sind in den folgenden Abschnitten beschrieben.

Im übrigen wird auf die Beschreibung des Betriebssystems DBOS verwiesen.

Dateiverwaltung unter DBOS

Die Systemprozeduren

CREA (u,f,l)	Datei eröffnen
KILL (u,f)	Datei löschen
ALTR (u,f1,f2)	Dateinamen ändern
LENG(u,f,l)	Dateilänge kürzen
PROT (u,f,p)	Datei schützen

dienen zur Verwaltung der Plattenspeicher-Dateien im Rahmen des plattenorientierten Basis-Betriebssystems DBOS. Sie werden mit CALL aufgerufen und verändern das Datei-Inhaltsverzeichnis.

Die Nummer des Plattenlaufwerks (Einheit) wird durch u spezifiziert; f bezeichnet den Dateinamen (String mit max. 6 Zeichen); l gibt die Länge der Datei in Sektoren zu je 128 Bytes an. Der Dateischutz-Code p ist unter CALL PROT erklärt.

- m CALL CREA (u,f,l)

Auf der Einheit u wird unter dem Namen f eine Datei mit der Länge l eröffnet.

Beispiele:

```
100 CALL CREA(0,"SIGMA",25)
```

Eröffnen einer Datei SIGMA von 25 x 128 byte Länge auf Einheit 0

```
500 CHAR F$(6)
...
550 LET F$(1,4)="FIL3"
560 LET U=4,SCTS=100
...
600 CALL CREA(U,F$,SCTS)
```

Eröffnen einer Datei FIL3 von 100 x 128 byte Länge auf Einheit 4

- m CALL KILL (u,f)

Auf der Einheit u wird die Datei mit dem Namen f gelöscht.

Beispiel:

```
200 CALL KILL(0,"SIGMA")
```

Löschen der Datei "SIGMA" auf Einheit 0

- m CALL ALTR (u,f1,f2)

Auf der Einheit u wird der Name der Datei f1 durch den Namen f2 ersetzt. Alle übrigen Parameter bleiben erhalten.

Beispiel:

```
650 CALL ALTR(4,"FIL3","FIL7")
```
Neuer Dateiname: FIL7

- m CALL LENG (u,f,l)

Auf der Einheit u wird die Länge der Datei mit dem Namen f auf l Sektoren verkürzt.

Beispiel:

```
660 CALL LENG(4,"FIL7",50)
```
Neue Dateilänge: 50 x 128 byte

- m CALL PROT (u,f,p)

Auf der Einheit u erhält die Datei mit dem Namen f den Schutzcode p. Der Schutzcode p hat folgende Bedeutung:

1 Schreib-/Lösch-Schutz
2 feste Startadressen (ab Systemende)
4 nicht ausführbarer Code (kein Maschinencode)
8 automatischer Start bei Namen-Eingabe auf Unit Ø (direkter Aufruf durch Programmname)

Für p können Kombinationen dieser Codes eingesetzt werden.
(z.B. 11 ≙ 8 und 2 und 1)

Beispiele:

```
700 CALL PROT(4,"FIL7",5)
750 CALL PROT(4,"FIL7",4)
800 CALL PROT(4,"FIL7",0)
```
Schreib-/Lösch-Schutz; nicht ausführbarer Code
Aufheben des Schreib-/Lösch-Schutzes
Aufheben des Eintrags "nicht ausführbarer Code"

Hinweis:

Bestimmte Programm- und System-Fehler werden bei Ausführung dieser Systemprozeduren erkannt und sind durch die Systemvariable ERR abfragbar.

Siehe hierzu Abschnitt "DBOS Fehlermeldungen".

Dateizugriff unter DBOS

Mit den Systemprozeduren

OPEN(u,f,w)	Datei öffnen
CLSE (w)	Datei schließen
GFB (w,a,s,l)	Datei in Zahlenfeld lesen
PFB (w,a,s,l)	Datei aus Zahlenfeld schreiben
GFBS (w,a,s,l)	Datei in String lesen
PFBS (w,a,s,l)	Datei aus String schreiben

wird zum Inhalt von Plattenspeicher-Dateien im Rahmen des plattenorientierten Basis-Betriebssystems DBOS zugegriffen. Die Prozeduren werden mit CALL aufgerufen.

Die Nummer des Plattenlaufwerks (Einheit) wird durch u spezifiziert; f bezeichnet den Dateinamen, w die Arbeitsnummer (0...7) der Datei nach Öffnung, a das Kernspeicherfeld, s die Nummer des ersten gelesenen bzw. beschriebenen Sektors der Datei und l die Anzahl der zu lesenden bzw. zu schreibenden Sektoren (1 Sektor = 128 Bytes).

• m CALL OPEN (u,f,w)

Auf der Einheit u wird die Datei f für den Lese- oder Schreib-Zugriff geöffnet. Ihr wird die Arbeitsnummer w zugeteilt.

Beispiele:

```
110 CALL OPEN(0,"SIGMA",3)
```

Öffnen der Datei SIGMA auf Einheit 0 mit Arbeitsnummer 3

```
610 LET WKNR=7
...
640 CALL OPEN(4,"FIL3",WKNR)
```

Öffnen der Datei FIL3 auf Einheit 4 mit Arbeitsnummer 7

• m CALL CLSE (w)

Die Datei mit der Arbeitsnummer w wird für den Lese- oder Schreib-Zugriff geschlossen. Die Arbeitsnummer w wird frei und kann neu vergeben werden.

Beispiel:

```
190 CALL CLSE(3)
```

Datei mit der Arbeitsnummer 3 (in diesem Falle die Datei SIGMA auf Einheit 0) schließen

• m CALL GFB (w,a,s,l)

Der Inhalt der Datei mit Arbeitsnummer w wird, beginnend mit Sektor s und einer Länge von l aufeinanderfolgenden Sektoren, in ein Zahlenfeld gelesen, dessen erstes Element durch a bestimmt ist.

Beispiele:

```
10 DIM A(4,63)
120 CALL GFB(3,A(0,1),0,1)   Lesen 1 Sektor ab Sektor 0 in Feld A ab Element (0,1)
125 CALL GFB(3,A(1,0),10,2)  Lesen 2   "   "   "   10   "   A ab   "      (1,0)
```

• m CALL PFB (w,a,s,l)

Die Datei mit der Arbeitsnummer w wird, beginnend mit Sektor s und in einer Länge von l Sektoren, mit dem Inhalt eines Zahlenfeldes beschrieben, dessen erstes Element durch a bestimmt ist.

Beispiele:

```
130 CALL PFB(3,A(0,0),0,1)   Schreiben 1 Sektor ab Sektor 0 aus Feld A ab El. (0,0)
135 CALL PFB(3,A(2,0),5,2)   Schreiben 2   "   "   "   5   "   "   "  ab El. (2,0)
```

• m CALL GFBS (w,a,s,l)

Der Inhalt der Datei mit der Arbeitsnummer w wird, beginnend mit Sektor s und in einer Länge von l aufeinanderfolgenden Sektoren, in eine durch a bezeichnete Stringvariable gelesen.

Beispiel:

```
20 CHAR F$(256)
140 CALL GFBS(3,F$,20,2)     Lesen 2 Sektoren ab Sektor 20 in String F$
```

• m CALL PFBS (w,a,s,l)

Die Datei mit der Arbeitsnummer w wird, beginnend mit Sektor s und in einer Länge von l aufeinanderfolgenden Sektoren, mit dem Inhalt der Stringvariablen beschrieben.

Beispiel:

```
150 CALL PFBS(3,F$,21,1)     Schreiben 1 Sekt.ab Sektor 21 aus String F$
                             (erste 128 Zeichen)
```

Hinweis:

Bestimmte Programm- und Systemfehler werden bei Ausführung dieser Systemprozeduren erkannt und sind durch die Systemvariable ERR abfragbar.

Siehe hierzu Abschnitt "DBOS Fehlermeldungen".

DBOS Fehlermeldungen

Fehler bei der Ausführung von Systemprozeduren, die der Datei-Verwaltung oder dem Datei-Zugriff unter dem Betriebssystem DBOS dienen, werden vom System erkannt und sind mit Hilfe der Systemvariablen

ERR

abfragbar. ERR ist die allgemeine Fehlervariable von BASEX; sie muß abgefragt werden, bevor (in der gleichen Programmebene) eine neue Systemprozedur aufgerufen wird, die eine Fehlermeldung liefern kann.

Die Systemvariable ERR liefert den Wert 0, wenn kein Fehler aufgetreten ist; andernfalls liefert sie folgende Werte:

Parameter-Fehler:	ERR = 4	Datei-Name bei CREA oder ALTR schon vorhanden. Arbeitsnummer bei OPEN schon verwendet.
	= 5	Zu große Datei-Länge bei CREA, LENG, GFB, PFB, GFBS oder PFBS.
	= 6	Datei-Name nicht vorhanden oder nicht geöffnet.
	= 7	Index bzw. Anzahl < 0 bei GF, PF, GFS, PFS
System-Fehler:	ERR = 129	Laufwerk nicht bereit
	= 130	Schreibschutz für Laufwerk oder Datei
	= 131	CRC-Fehler (Lesefehler nach 5 Versuchen)
	= 132	Plattenadresse zu groß
	= 133	Laufwerk nicht angeschlossen
	= 134	Kernspeicheradresse zu klein
	= 135	Falsche Betriebsart
	= 136	Spur nicht gefunden (WP)

Beispiel für Abfrage von Parameter-Fehlern:

```
110 CALL OPEN(0,"SIGMA",3)
115 GOSUB 800
...
800 IF ERR=0 THEN 820
810 GOTO ERR-3 OF 830,840,850
820 RETURN
830 PRINT "NAME DOPPELT/ARB.-NR. BELEGT"
835 RETURN
840 PRINT"DATEILAENGE ZU GROSS"
845 RETURN
850 PRINT "NAME UNBEKANNT/DATEISCHUTZ";
855 PRINT " VERLETZT"
860 RETURN
```

Systemprozeduren GF, PF GFS und PFS

Als Option kann auch zu einzelnen Zahlen oder Zeichen in Plattenspeicher-Dateien zugegriffen werden sowie zu Zahlen oder Stringfeldern beliebiger Länge, in welche die Datei strukturiert ist. Hierzu dienen die Systemprozeduren

GF (w,a,i,l)	Zahlen lesen
PF (w,a,i,l)	Zahlen schreiben
GFS(w,a,i,l)	String lesen
PFS (w,a,i,l)	String schreiben

Sie erweitern den im Betriebssystem DBOS vorgesehenen sektorweisen Zugriff (mit GFB, PFB, GFBS und PFBS). Auch hier sind die Dateien mit OPEN zu eröffnen und mit CLSE zu schließen; mit OPEN wird die Arbeitsnummer w der Datei vergeben.

Bei dieser Zugriffsart wird angenommen, daß eine Datei aus Einzel-Elementen oder aus ein- oder zweidimensionalen Feldern besteht, wobei ein Element entweder einer Zahl (Länge 4 byte) oder einem Zeichen (Länge 1 byte) entspricht.

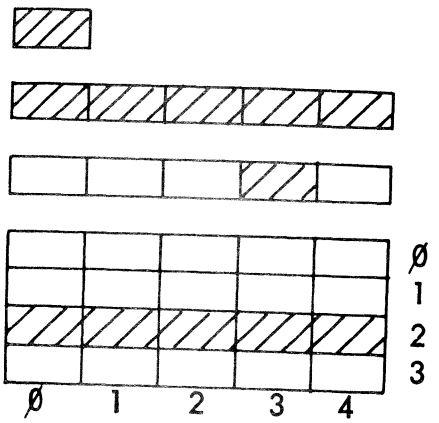
Die Angabe a bezeichnet das Kernspeicher-Feld, mit dem der Austausch stattfindet; es ist mit DIM (für Zahlen) oder CHAR (für Strings) zu reservieren.

Der Parameter l gibt die Anzahl der übertragenen Zahlen bzw. Zeichen an (Einheit 1 bzw. 4 byte). Mit i wird ein Index angegeben, der bei eindimensionalen Dateien ein Element, bei zweidimensionalen eine Zeile der Länge l spezifiziert. Ist i = 0, so erfolgt keine Indizierung. Die Parameter i und l definieren somit eine Position mit der Lage i * l in der Datei.

Beispiele:

l = 1	i = 0
l = 5	i = 0
l = 1	i = 3
l = 5	i = 2

1 Element = 4 byte (Zahl) oder
1 byte (String)



• m CALL GF (w,a,i,l)

Lesen von l Zahlen ab Position i*l. der Datei w in Zahlenfeld a.

• m CALL PF (w,a,i,l)

Schreiben von l Zahlen aus Zahlenfeld a auf Datei w ab Position i*l.

• m CALL GFS (w,a,i,l)

Lesen von l Zeichen ab Position i*l der Datei w in String a.

• m CALL PFS (w,a,i,l)

Schreiben von l Zeichen aus String a auf Datei w ab Position i*l.

Der Parameter a ist beim Zugriff zu Zahlendateien der Name einer ein- oder zweifach indizierten Variablen. Er muß nicht mit dem Feldumfang übereinstimmen; Lesen bzw. Schreiben erfolgt von dieser Variablen aus in Richtung steigender Indizes (Spalten, Zeilen).

Der Parameter a ist beim Zugriff zu Stringdateien der Name einer String-Variablen. Ihre Länge kann größer sein als die Länge l des tatsächlich übertragenen Strings; Lesen bzw. Schreiben beginnt mit dem ersten Zeichen der String-Variablen.

Für die Parameter w, l, i können Zahlen-Konstanten, -variablen oder beliebige arithmetische Ausdrücke eingesetzt werden.

Beispiele:

```
200 CALL GF (0,A(0),0,1)
220 CALL PF (3,ZF(3,2),5,2)
230 CALL GFS (X,B$,A+2,10)
270 CALL PFS (7,TEXT$,0,128)
```

Parameter und Systemfehler sind über die Systemvariable ERR abfragbar (siehe DBOS Fehlermeldungen).

Realtime-Funktionen

BASEX arbeitet in DIETZ 621-Systemen mit einer in die BASEX-Software integrierten Version des Echtzeit-Betriebssystems RTOS.

Die wichtigsten Aufgaben von RTOS sind folgende Realtime-Funktionen:

- Multiprogramming: Simultane Abarbeitung mehrerer Benutzer-Programme mit gegenseitiger Beauftragung und mit Verwaltung der Aufträge.
- Zeitverwaltung: Laufende Führung der Absolutzeit; Verwaltung und Ausführung von zeitgebundenen Programmaufträgen.
- Interrupts: Verwaltung und Ausführung von Programmaufträgen, die an spontane äußere Ereignisse (Interrupts) gebunden sind.

Die einzelnen Programmfunktionen laufen unabhängig voneinander in "Programmebenen" ab, die von der Hardware-Struktur des DIETZ 621 vorgegeben sind und von ihr stark unterstützt werden.

Die Programme in den einzelnen Ebenen haben eine hierarchische Priorität. Ein Programm, das in einer höheren Ebene läuft, unterbricht die Programme aller niedrigen Ebenen und blockiert ihren Ablauf, solange es arbeitet. Allerdings gilt dies nur, solange das prioritäre Programm die CPU des Computers benutzt; insbesondere bei Ein-/Ausgabe-intensiven Programmen entstehen so geringe Belegungszeiten der CPU, daß die Programme auf niedrigeren Ebenen, u.U. mit verminderter Geschwindigkeit, ihren Ablauf fortsetzen können.

Entsprechend der Hardware-Konfiguration sind 2 RTOS-Versionen verfügbar:

- für insgesamt 8 Ebenen
- für insgesamt 16 Ebenen.

Die Programmebenen sind aufgeteilt in:

- Systemebenen
- Benutzerebenen.

Die Systemebenen haben höchste Priorität. In ihnen laufen zur Betriebssystem-Funktionen; der Benutzer hat keinen Zugriff zu ihnen. Sie sind aufgeteilt in

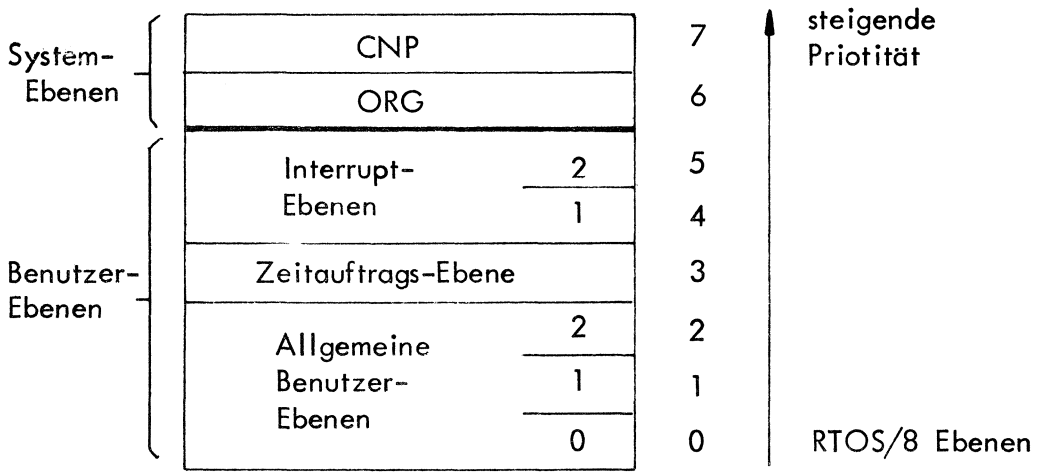
- CNP-Ebene: Führt die Absolutzeit, überwacht den jeweils aktuellsten Zeitauftrag und erkennt Hardware-Systemfehler.
- ORG-Ebene: Verwaltet den Plattenspeicher sowie die Geräte-Peripherie.
- Freie Systemebenen: Für spezielle Betriebssystem-Implementierungen (nur bei 16-Ebenen-Version).

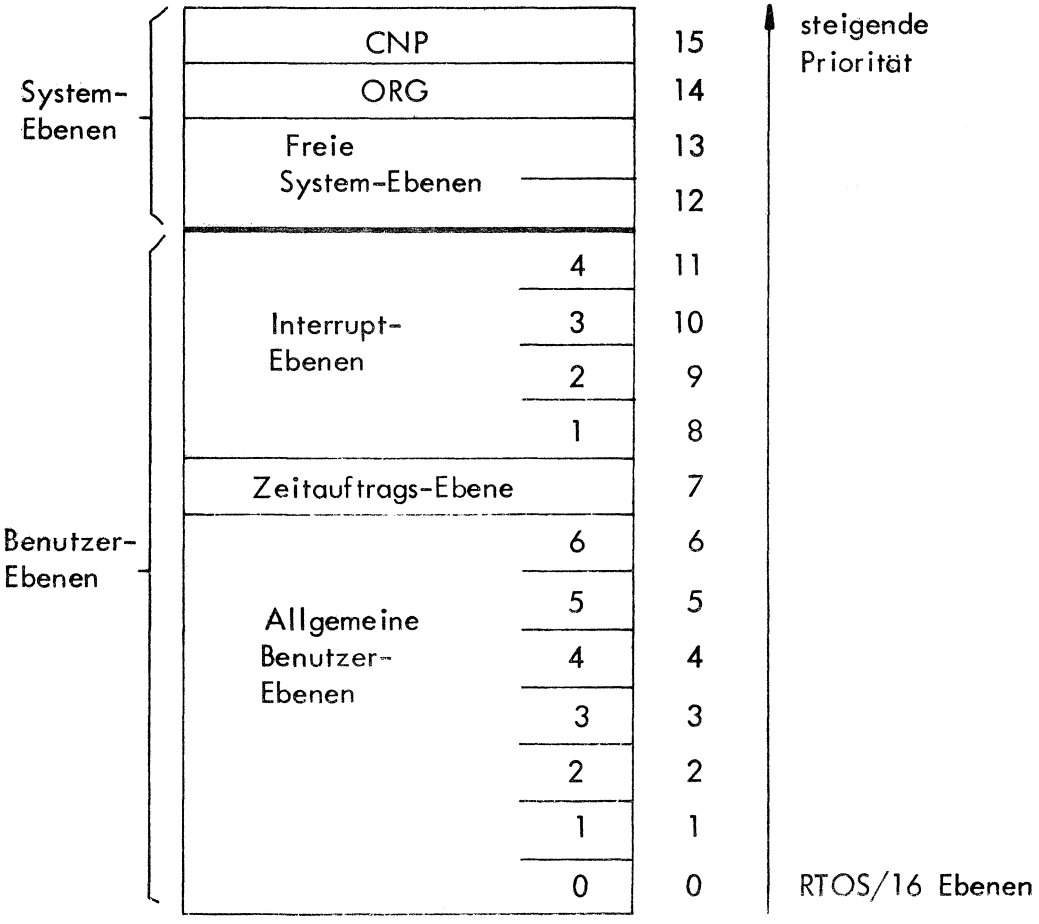
Hierarchisch darunter liegen die Benutzer-Ebenen. In ihnen laufen BASEX-Programme; der Benutzer hat ihre Verwaltung voll in der Hand.

Sie sind aufgeteilt in

- Interrupt-Ebenen: Hier laufen die von externen Ereignissen (Interrupts) aktivierten Programme.
- Zeitauftrags-Ebene : Hier laufen von der Zeitauftrags-Überwachung aktivierte Programme.
- Allgemeine Benutzer-Ebenen: Hier laufen die übrigen Benutzer-Programme. Die Ebene 0 (niedrigste Priorität) wird als "Hauptebene" bezeichnet.

Zu beachten ist, daß zur gleichen Zeit in einer Benutzer-Ebene nur ein BASEX-Programm aktiviert sein bzw. ablaufen kann.


System-
Ebenen
Benutzer-
Ebenen
CNP
ORG
Interrupt-
Ebenen
2
1
Zeitauftrags-Ebene
Allgemeine
Benutzer-
Ebenen
2
1
0
7
6
5
4
3
2
1
0
steigende
Priotität
RTOS/8 Ebenen


System-
Ebenen
Benutzer-
Ebenen
CNP
ORG
Freie
System-Ebenen
Interrupt-
Ebenen
4
3
2
1
Zeitauftrags-Ebene
Allgemeine
Benutzer-
Ebenen
6
5
4
3
2
1
0
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
steigende
Priorität
RTOS/16 Ebenen

Die Realtime-Funktionen von BASEX sind in den folgenden Abschnitten beschrieben.

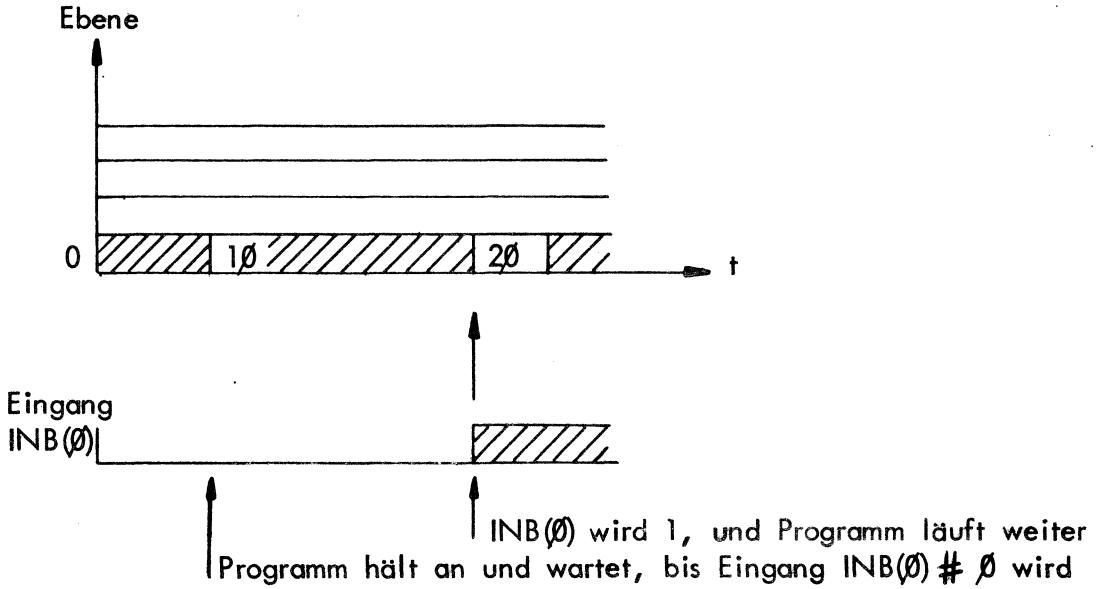
Die wichtigsten Realtime-Befehle sind die Anweisungen

- WAIT	Warten
- START	Programmauftrag
- STOP	Ende Auftrags-Programm
- AFTER	Zeitauftrag
- ON INT	Interrupt-Auftrag

Ihre Wirkung ist an folgenden Programm-Beispielen bzw. Zeitdiagrammen abzulesen:

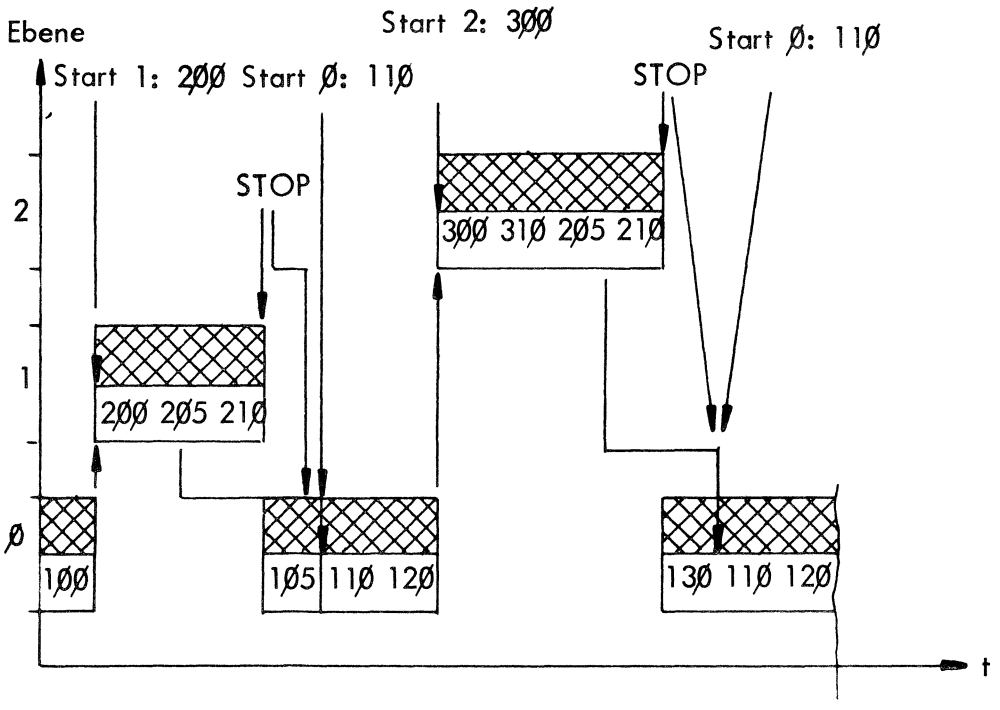
Beispiel für WAIT:

```
10 WAIT INB(0)
20 ...
```



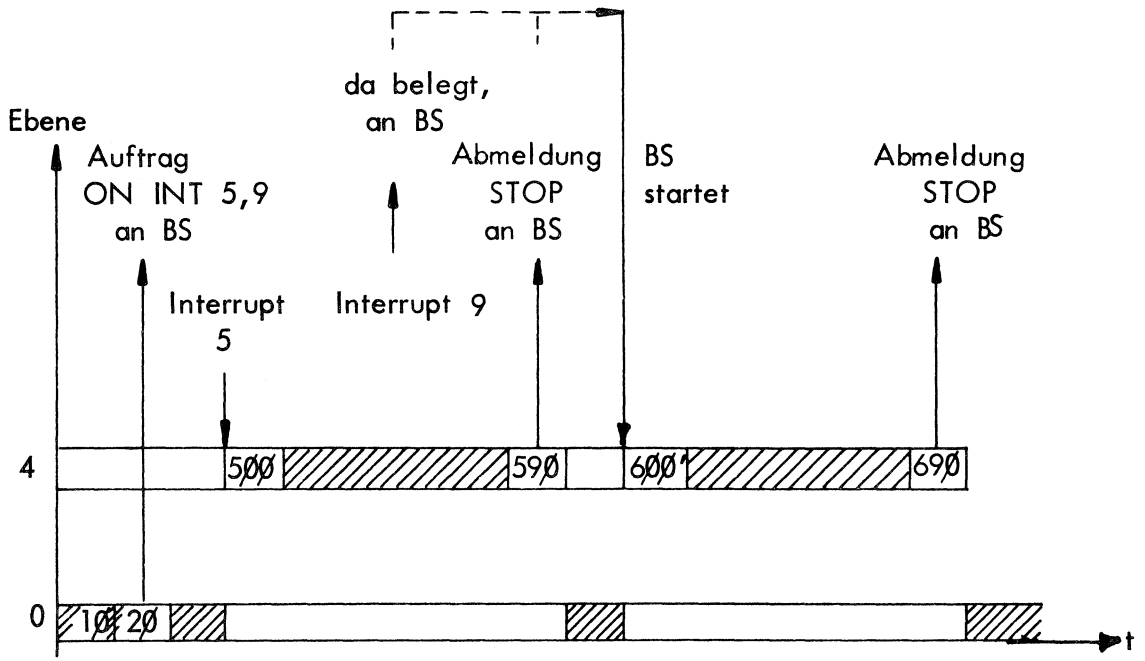
Beispiel für START:

```
100 START 1:200
105 STOP
110 PRINT A
120 START 2:300
130 STOP
200 LET A=0
205 START 0:110
210 STOP
300 LET A=A+1
310 IF A<100 THEN 205
320 END
```



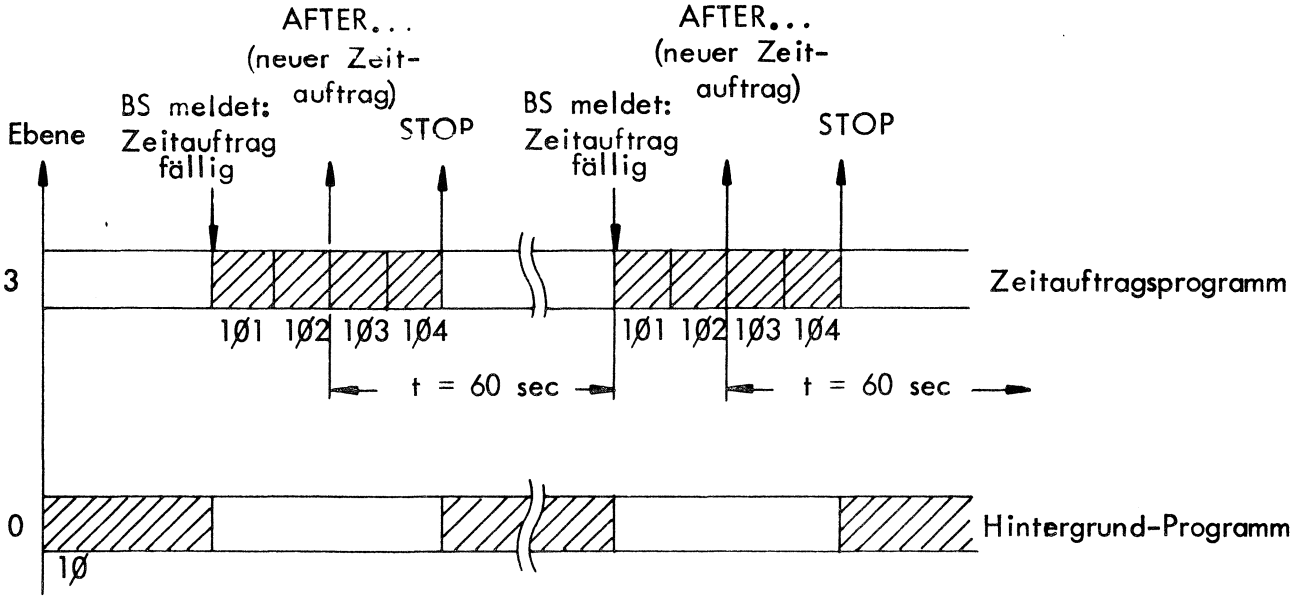
Beispiel für ON INT:

```
10 REM PROGRAMM IN HAUPTEBENE
20 ON INT  5:GOTO 500
30 ON INT  9:GOTO 600
40 ENAB 5,9
...
500 REM SERVICE-ROUTINE ZU INT 5
...
590 STOP
600 REM SERVICE-ROUTINE ZU INT 9
...
690 STOP
```



Beispiel für AFTER:

```
10 REM HINTERGRUNDPROGRAMM
...
100 REM PEGELM.ABLESUNG JEDE MINUTE
101 IF I=IMAX THEN 103
102 AFTER 60000: GOTO 101
103 LET X(I)=INP(I),I=I+1
104 STOP
```



Statement WAIT

- m WAIT e

Das Statement WAIT ist eine bedingte Halt-Anweisung.

Das Programm hält an, bis der Wert des Ausdruckes e ungleich Null geworden ist.

Beispiele:

```
 10 WAIT MSEC>=500
300 WAIT INB(5)
500 WAIT X15 OR INB(2) AND X16
```

Hinweis:

Bei WAIT befindet sich das System in einer Programmschleife, die fortlaufend den Wert des Ausdrucks e ermittelt. Sobald infolge eines äußeren Ereignisses (z.B. ein Prozeßeingang), der Systemzeit oder der Veränderung des Inhalts einer Variablen (durch ein Programm in einer höheren Ebene) der Wert des Ausdrucks e ungleich Null wird, geht das Programm mit der nächsten Anweisung weiter.

WAIT-Anweisungen blockieren die Programmausführung in allen niedrigeren Ebenen; sie sollten daher möglichst nur in der niedrigsten Benutzer-Ebene (Ebene 0) laufen.

Statement START

- m START l : n

Das Statement START erteilt der Programmebene l den Auftrag, ein Programm auszuführen, das mit der Anweisung n beginnt.

Läuft auf dieser Ebene kein Programm, so wird mit der Ausführung sofort begonnen; andernfalls wird die Ausführung so lange zurückgestellt, bis das laufende Programm beendet ist bzw. alle weiteren dieser Ebene durch START erteilten Aufträge abgewickelt sind.

Programmaufträge mit START können von jeder Programmebene aus erteilt werden. Sie dienen vor allem zur Auslösung von Programmabläufen in anderen Benutzerebenen und bewirken so deren Synchronisation mit Ereignissen anderer Art, z.B. eintreffenden Interrupts, abgelaufenen Zeiten oder erreichten Programmzuständen.

Jeder über START beauftragte Programmteil ist durch die Anweisung STOP abzuschließen; das Programm hält in dieser Ebene an, es sei denn, es liegen weitere Programmaufträge vor.

Programmaufträge werden in der zeitlichen Reihenfolge ihres Eintreffens abgewickelt.

Beispiele:

```
100 AFTER 30000 : START 1: 700
...
700 LET AMES=INA(0),OUTB(1)=0
710 STOP
200 ON INT 6:START 0 : 60
...
60 PRINT "ALARM 6"
65 STOP
900 START 2: 1500
...
1500 GOSUB 310
1510 STOP
```

nach 30 sec

wenn Interrupt 6

sofortiger Auftrag

Bemerkung: Kommando RUN startet das Gesamtprogramm in der Hauptebene (Ebenen-Nummer 0).

Statement STOP

- m STOP

Die Anweisung STOP beendet einen Programmteil, der auf einer der Programmebenen läuft. Das Programm in dieser Ebene wird abgeschlossen; liegt bereits ein neuerer Auftrag für diese Ebene vor (und ist dieser aktuell), so wird der neue Programmteil unverzüglich begonnen.

STOP ist in folgenden Fällen zu verwenden:

- zum Abschluß des durch das Kommando RUN gestarteten Programms in der Hauptebene,
- zum Abschluß eines mit START ausgelösten Programmteils in der Haupt- oder den Zusatzebenen,
- am Ende eines mit AFTER t : GOTO n oder ON INT i : GOTO n formulierten Zeit- oder Interrupt-Auftrags.

Beispiele:

```
1 ON INT Ø : START Ø : 2Ø
5 ENAB Ø
1Ø STOP
2Ø ...
```

Weitere Beispiele siehe unter AFTER, ON INT und START.

Bemerkung:

Auch wenn das Programm in allen Ebenen ruht bzw. durch STOP beendet ist, geht das System nicht in den Bedienungsbetrieb über; diese Wirkung hat allein das Statement END.

Daher ist in jedem Programm mindestens eine END-Anweisung vorzusehen.

Systemvariable LEV

Für die Zuordnung von Datenfeldern zur jeweils laufenden Programmebene sowie für Ebenen-abhängige Modifikationen von Programm-Abläufen ist die Systemvariable

LEV Level

vorgesehen.

Sie wird im Programm wie eine einfache Variable behandelt und vor allem als Index verwendet. Sie liefert im Augenblick der Abarbeitung die Nummer der gerade laufenden Programmebene.

Mit LEV ist es möglich, in verschiedenen Ebenen das gleiche Programm laufen zu lassen. Jeder Ebene ordnet man z.B. durch den Index LEV verschiedene Teile des gleichen Datenfeldes zu; jede Ebene arbeitet dann mit den eigenen Daten, ohne die der anderen zu zerstören.

Außerdem können mit LEV ebenenabhängige Unterschiede im Programmverlauf eingeführt werden.

- LEV

Liefert die Nummer der laufenden Programmebene (0, 1, ...).

Beispiel 1:

```
100 DIM ABA(2,5)
200 FOR I=0 TO 5
210 INPUT ABA (6*LEV,I)
215 NEXT I
```

I =	0	1	2	3	4	5	
ABA							LEV = 0
							= 1
							= 2

Die eingegebenen Daten gehen in Zeile 0, 1 oder 2 des Feldes ABA je nachdem, ob das Programm in Ebene 0, 1 oder 2 läuft.

Beispiel 2:

```
555 IF LEV=1 THEN 575
```

Ebenenabhängige Modifikation eines Programms: Nur wenn es in Ebene 1 läuft, erfolgt ein Sprung nach 575.

Systemvariable STNU

Zur Abfrage von aktuellen Anweisungsnummern dient die Systemvariable

STNU(I) Statement Number

Die Systemvariable liefert die Nummer der BASEX-Anweisung, die im Augenblick der Abfrage in der Ebene I abgearbeitet wird. Damit kann von jeder Ebene aus der aktuelle Programmstand jeder anderen Benutzer-Ebene abgefragt werden.

Hinweis:

Wird die Anweisungsnummer einer Ebene abgefragt, in der kein Programm läuft, so erhält man eine undefinierte hohe Zahl.

STNU(I)

Liefert die Nummer der aktuellen BASEX-Anweisung des in Ebene I laufenden Programms.

Beispiele:

```
100 PRINT STNU(0)
220 IF STNU(7)>800 THEN 250
```

Systemvariablen MSEC, SEC, MIN und HOUR

Die Systemvariablen

MSEC	Millisekunden
SEC	Sekunden
MIN	Minuten
HOUR	Stunden

liefern die Absolutzeit des Systems.

Sie werden im Programm wie einfache Variablen behandelt. Ihr Inhalt kann jederzeit abgefragt werden. Die Variablen werden vom System laufend geführt.

Die Systemvariablen SEC, MIN und HOUR können durch das Kommando TIME sowie durch die Systemprozedur STIM auf Ausgangswerte der Absolutzeit gesetzt werden.

- MSEC

  Liefert die Absolutzeit in Millisekunden (Ø...3599999).

- SEC

  Liefert die Absolutzeit in Sekunden (Ø...59).

- MIN

  Liefert die Absolutzeit in Minuten (Ø...59).

- HOUR

  Liefert die Absolutzeit in Stunden (Ø...23).

Beispiele:

```
300 AFTER 10000-MSEC: LET OUTB(6)=1
400 IF HOUR>=5 THEN 450
450 PRINT HOUR,MIN,SEC
```

Bei 10 s Absolutzeit Bit-Ausgang 6 setzen.
Ab 5 h Absolutzeit Sprung nach 45Ø:
Stunden, Minuten und Sekunden drucken

Bemerkung:

Je nach Systemkonfiguration wird die Zeitvariable MSEC mit einer Auflösung von 1, 10 oder 100 ms vom System geführt.

Kommando TIME

Die Absolutzeit des Systems kann im BASEX-Bedienungsbetrieb durch das Kommando TIME abgefragt bzw. neu eingestellt werden. Dabei werden die Zeitvariablen HOUR, MIN und SEC ausgedruckt bzw. auf neue Werte gesetzt.

Abfragen: TIME eingeben, anschließend Wagenrücklauf.
Die Absolutzeit in Stunden, Minuten und Sekunden wird ausgedruckt.

Beispiel (Eingaben unterstrichen):

```
TIME
23:59:59
```

Eingeben: TIME sowie die neue Absolutzeit (Stunden:Minuten:Sekunden) wird eingegeben, anschließend Wagenrücklauf.

Beispiel (Eingaben unterstrichen):

```
TIME 15:08:45
```

Systemprozedur STIM

Den Zeitvarialben HOUR, MIN und SEC werden durch die Systemprozedur

STIM (h,m,s) Zeit setzen

neue Werte zugewiesen. Dadurch wird die Absolutzeit des Systems neu eingestellt.

Die Systemprozedur wird mit CALL aufgerufen; als Parameter können Zahlenkonstanten, Zahlenvariablen oder arithmetische Ausdrücke eingesetzt werden. Der für die Zeitvariablen gültige Zahlenbereich ($0 \leq h \leq 23$; $0 \leq m \leq 59$; $0 \leq s \leq 59$) ist zu beachten.

- CALL STIM (h,m,s)

  Die Absolutzeit wird auf h Stunden, m Minuten und s Sekunden gesetzt.

Beispiele:

```
400 CALL STIM (0,0,0)
500 CALL STIM (HOUR,12,MIN,5EC)
650 CALL STIM (23,59,5 )
```

Bemerkung: Falsche Angaben (z.B. HOUR = 25) führen zum Setzen der Variablen ERR auf 1

Statement AFTER

- m AFTER t : Auftragsanweisung
- m AFTER t : GOTO n

Durch AFTER wird dem System der Auftrag erteilt, nach einer Zeit t die Auftragsanweisung durchzuführen.

Die Zeit t beginnt mit der Abarbeitung des AFTER-Statements. Sie kann als Zahlenkonstante, -variable oder arithmetischer Ausdruck angegeben werden; ihr Wert entspricht der Relativzeit in Millisekunden.

AFTER-Aufträge können von jeder Programmebene aus erteilt werden. Die Ausführung des zeitgebundenen Programms (der Auftragsanweisung) selbst erfolgt jedoch stets in der dafür reservierten prioritären Zeitebene.

Auftragsanweisungen können einzelne Statements vom Typ LET, START, PUT, CALL oder END sein. Sollen längere Folgen von Anweisungen als Zeitauftrag ablaufen, so sind diese als getrennter Programmteil zu formulieren, mit GOTO n als Auftragsanweisung anzuspringen und mit dem Statement STOP abzuschließen. Weitere AFTER-Anweisungen können darin enthalten sein.

Beispiele:

```
100 AFTER 200: LET A=INW(3)
200 AFTER T2: START 0:500
300 AFTER X+50: CALL HLTC(1)
500 AFTER 1000: GOTO 550
...
550 LET OUTB(1)=1
555 AFTER 1000: END
560 STOP
```

Abfrage INW(3) nach 200 ms
Programmstart Ebene Ø nach T2 ms
Anhalten Zähler 1 nach X + 50 ms
nach 1000 ms:

Setzen OUTB(1)
Nach weiteren 1000 ms Programmende

Statement ON INT

- m ON INT i : Auftragsanweisung
- m ON INT i : GOTO n

Durch ON INT wird dem System der Auftrag erteilt, bei Auftreten des Interrupts i die Auftragsanweisung durchzuführen.

Für i ist die Nummer des Interrupt-Eingangs einzutragen.

ON INT-Aufträge können von jeder Programmebene aus erteilt werden. Die Ausführung des durch den Interrupt ausgelösten Programms (Auftragsanweisung) erfolgt jedoch stets in einer der hierfür reservierten prioritären Interrupt-Ebenen, die jeweils Gruppen von Interrupt-Nummern zugeordnet sind.

Demselben Interrupt i können im Laufe des Programms über ON INT verschiedene Aufträge zugewiesen werden. Auftragsanweisungen können einzelne Statements vom Typ PUT, LET, START, CALL oder END sein. Sollen längere Folgen von Anweisungen als Interrupt-Programm ablaufen, so sind diese als getrennter Programmteil zu formulieren, mit GOTO n als Auftragsanweisung anzuspringen und mit dem Statement STOP abzuschließen.

Beispiele:

```
140 ON INT 3: LET OUTD(6)=DIS
150 ON INT 31: START 1:1050
160 ENAB 3,31
170 ON INT 17: END
180 ON INT 2: GOTO 600
190 ENAB 17,2
600 LET OUTB(2)=0
610 FOR X=0 TO 7
615 LET OUTW(X)=0
620 NEXT X
625 STOP
```

Ausgabe OUTD(6) wenn Interrupt 3
Start Ebene 1 wenn Interrupt 31

Programmende wenn Interrupt 17
wenn Interrupt 2:

Nullsetzen OUTB(2)
Nullsetzen OUTW(0...7)

Interrupts werden nur vom System akzeptiert, wenn sie durch das Statement ENAB zugelassen sind; durch DISAB kann man sie wieder sperren.

Statements ENAB und DISAB

- m ENAB i
- m ENAB i1, i2, ...

Das Statement ENAB bewirkt, daß die Interrupteingänge i geöffnet werden. Eintreffende Interrupts lösen die mit ON INT zugewiesenen Abläufe aus.

Beispiel:

```
100 ON INT 0 : GOTO 1000
110 ON INT 1 : GOTO 2000
120 ENAB 0,1
130 LET OUTB(0)=1
140 ON INT 7 : END
150 ENAB 7
```

- m DISAB i
- m DISAB i1, i2, ...

Durch das Statement DISAB werden die Interrupteingänge i wieder gesperrt. Eintreffende Interrupts bewirken nichts.

Beispiele:

```
100 DISAB 0
200 DISAB 0,1,2,3,4,5,6,7
```

Bemerkung:

Bei Programmbeginn sind alle Interrupteingänge gesperrt; jeder zu benutzende Eingang muß daher ausdrücklich mit ENAB geöffnet werden.

Auch im gesperrten Zustand wird ein eintreffender Interrupt gespeichert. Sobald durch ENAB zugelassen, wird der so gespeicherte Interrupt wirksam.

Prozeß-Ein-/-Ausgabe

Mit BASEX können Prozeßanschlüsse des DIETZ 621-Systems behandelt werden.

Für die Standard-Prozeßperipherie sind in BASEX Systemvariablen und -prozeduren enthalten, von denen sie voll unterstützt wird. Sie sind speziell auf diese Prozeßanschlüsse zugeschnitten und mit mnemotechnischen Bezeichnungen verfügbar (z.B. INW, ...), so daß der Benutzer einen hohen Programmierkomfort zur Verfügung hat.

Im Falle von Sonder-Prozeßperipherie oder vom Benutzer selbst angefertigten Interfaces, die von BASEX angesprochen werden sollen, gibt es universelle Möglichkeiten zur Formulierung geeigneter Unterprogramme und für deren Aufruf (Statements EQUI, EQUO und PUT).

Im folgenden sind die Funktionen für beide Formen der Prozeßperipherie-Behandlung besprochen. Die Systemprozeduren und -variablen für Standard-Prozeßanschlüsse werden konfigurationsabhängig implementiert.

Bemerkung:

Interrupt-Eingänge (Einkarten-Interfaces vom Typ PDSE) gehören zwar zur Prozeßperipherie, sind jedoch wegen ihrer besonderen Funktion unter den Statements ON INT bzw. ENAB und DISAB beschrieben (siehe 2.4.4.1/2.4.4.2).

Statement EQUI

- m EQUI Name = % Maschinencode %
- m EQUI Name (Index) = % Maschinencode %

Mit Hilfe von EQUI kann der Benutzer Unterprogramme (Makros) zu eigenen Systemvariablen vom INPUT-Typ formulieren.

EQUI definiert einfache oder indizierte Systemvariablen, die in arithmetischen Ausdrücken an beliebiger Stelle vorkommen dürfen. Sie werden insbesondere bei speziellen Prozeß-Eingaben benutzt, für die in BASEX keine Standard-Systemvariablen vorgesehen sind.

Das zur Systemvariablen gehörende Makro wird zwischen %...% als Hexa-String-Konstante angegeben; diese stellt den Maschinencode des Makros dar und ist stets mit dem Rücksprung-Befehl F27C abzuschließen. Ist die Hexa-String-Konstante länger als eine Zeile, so wird sie - ohne neue Anweisungsnummern - in der bzw. den nächsten Zeilen fortgesetzt.

Für EQUI-Makros gelten folgende Vereinbarungen:

- Datenübergabe: Der an das BASEX-Programm zu übergende binäre Zahlenwert muß nach Ablauf des Makros in folgenden Registern stehen:

  'Ø2 Mantisse niedrige Stellen
  'Ø3 "
  'Ø4 " hohe Stellen
  'Ø5 Exponent

- Index (wenn vorhanden): Der vom BASEX-Programm übergebene Wert des Index' steht vor Ablauf des Makros als binäre Ganzzahl in folgenden Registern:

  'Ø9 niedrige Stellen
  'ØA hohe Stellen

- Rücksprung: Durch Maschinenbefehl F27C am Ende des Makros.

- Benutzbare Register: 'Ø2...'1F

Beispiel:

```
100 EQUI TEMP = %........F27C%
150 EQUI XYZ(X) = %......F27C%
...
200 LET A=3.5*TEMP
210 IF XYZ(X) = 0 THEN 300
```

Statement EQUO

- m EQUO Name = % Maschinencode %
- m EQUO Name (Index) = % Maschinencode %

Mit Hilfe von EQUO kann der Benutzer eigene Unterprogramme (Makros) formulieren, die

a) Systemvariablen vom OUTPUT-Typ darstellen und die in LET-Anweisungen verwendet werden,

b) durch PUT-Anweisungen als selbständige Abläufe aufgerufen werden.

Sie werden vor allem bei speziellen Prozeß-Eingaben benutzt, für die in BASEX keine Standard-Makros vorgesehen sind.

Der Name des Makros kann einfach oder indiziert sein. Im Falle a) steht der Makro-Name auf der linken Seite des Gleichheitszeichens in LET-Anweisungen; dem Makro wird vom BASEX-Programm der Wert des Ausdrucks auf der rechten Seite zugewiesen.
Im Fall b) wird das Makro ohne Wertzuweisung durch PUT aufgerufen.

Das Makro wird zwischen %...% als Hexa-String-Konstante angegeben; diese stellt den Maschinencode des Makros dar und ist stets mit dem Rücksprung-Befehl F27C abzuschließen. Ist die Hexa-String-Konstante länger als eine Zeile, so wird sie - ohne neue Anweisungsnummern - in der bzw. den nächsten Zeilen fortgesetzt.

Für EQUO-Makros gelten folgende Vereinbarungen:

- Datenübergabe (nur für a): Der vom BASEX-Programm übergebene binäre Zahlenwert steht vor Ablauf des Makros in folgenden Registern:

  'Ø2 Mantisse niedrige Stellen
  'Ø3 "
  'Ø4 " hohe Stellen
  'Ø5 Exponent

- Index (wenn vorhanden): Der vom BASEX-Programm übergebene Wert des Index' steht vor Ablauf des Makros als binäre Ganzzahl in folgenden Registern:

  'Ø9 niedrige Stellen
  'ØA hohe Stellen

- Rücksprung: Durch Maschinenbefehl F27C am Ende des Makros.

- Benutzbare Register: 'Ø2...'1F

Beispiel:

```
100 EQUO ALRM = %.....F27C%
125 EQUO OLY(Y) = %....F27C%
...
250 PUT ALRM
285 LET OLY(A+3)=Y
```

Statement PUT

- m PUT Name
- m PUT Name (e)

Mit PUT wird ein Unterprogramm (Makro) aufgerufen, das entweder in BASEX als Standard-Makro enthalten ist oder das der Benutzer durch eine EQUO-Anweisung definiert hat.

Der Name des Makros ist einfach oder indiziert; als Index e kann jeder beliebige arithmetische Ausdruck verwendet werden.

Mit PUT aufgerufene Makros bewirken im allgemeinen Prozeß-Ausgabefunktionen ohne Übergabe eines Zahlenwertes.

Beispiele:

```
400 PUT ALRM
685 PUT REL(2)
```

Statische digitale Eingänge (PSSE)

Diese Eingänge werden durch die Systemvariablen

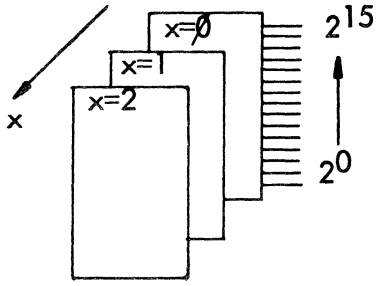
INW(x)	In-Word
IND (x)	In-Decimal
INB (x)	In-Bit

beschrieben. Sie werden im Programm wie einfach indizierte Zahlen-Variablen behandelt und liefern im Augenblick der Abarbeitung die am Eingang x anstehende digitale Information in Form eines Zahlenwertes.

Diese Systemvariablen sind in der Standard-Version den Einkarten-Interfaces vom Typ PSSE 16 zugeordnet (je 16 Eingangsleitungen). Sie interpretieren die externe Information auf unterschiedliche Weise.

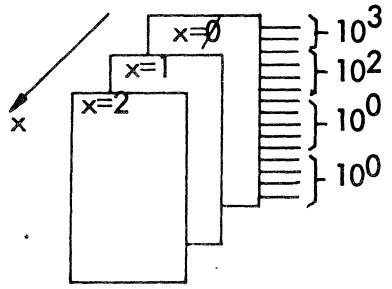
- INW(x)

Die externe Information wird als binäres 16 bit-Wort ($2^0 \ldots 2^{15}$) abgefragt und als positive Zahl (0...65535) ausgewertet. x ist die Nummer des Eingangs (des Interfaces). Sie läuft von 0 bis 63.



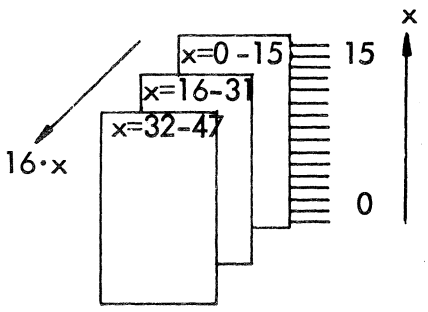
- IND(x)

Die externe Information wird als binär codiertes dezimales Wort mit 4 BCD-Stellen ($10^0 \ldots 10^3$) abgefragt und als positive Zahl (0...9999) ausgewertet. x ist die Nummer des Eingangs (wie oben).



- INB(x)

Die Information an einer Eingangsleitung eines Interfaces wird abgefragt und als Zahl (0 oder 1) ausgewertet. x ist die Nummer der Eingangsleitung (0...1023).



Beispiele:

```
100 LETA=INW(Ø)
150 LET DECS=IND(B)
170 IF INB(Y+5)=Ø THEN 200
```

Binäres Wort am Eingang Ø nach A
BCD-Information am Eingang B nach DECS
Sprung nach 200 wenn Bit-Eingang Y+5 gleich Null

Eine zweite Version der Systemvariablen INW(x), IND(x) und INB(x) ist Einkarten-Interfaces vom Typ PSSE 32 zugeordnet (je 32 Eingangsleitungen). Funktionell stimmt sie mit der Standard-Version überein, jedoch ist zu beachten, daß je Interface 2 Gruppen von 16-bit-Eingängen vorhanden sind:

INW(x):	Je Interface 2 16 bit-Worte mit aufeinanderfolgenden Nummern (z.B. x = 0...1)
IND(x):	Je Interface 2 mal 4 BCD-Stellen mit aufeinanderfolgenden Nummern (z.B. x = 0...1)
INB(x):	Je Interface 32 binäre Eingänge mit aufeinanderfolgenden Nummern (z.B. x = 0...31).

Speichernde digitale Ausgänge (PSSA)

Diese Ausgänge werden durch die Systemvariablen

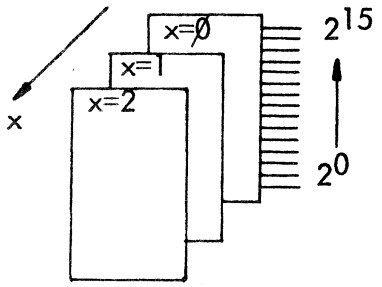
OUTW(x)	Out-Word
OUTD (x)	Out-Decimal
OUTB (x)	Out-Bit

beschrieben. Sie werden im Programm wie einfach indizierte Zahlenvariablen behandelt und setzen im Augenblick der Abarbeitung den Ausgang x so, wie er dem zugewiesenen Wert entspricht.

Diese Systemvariablen sind in einer Standard-Version den Einkarten-Interfaces vom Typ PSSA 16 zugeordnet (je 16 Ausgangsleitungen). Sie interpretieren den Wert, der dem Ausgang durch eine LET-Anweisung zugewiesen wird, auf unterschiedliche Weise.

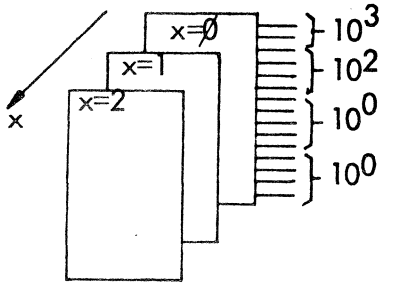
- OUTW(x)

  Der zugewiesene Wert wird als positive Zahl (0...65535) bewertet und als binäres 16 bit-Wort ($2^0 \ldots 2^{15}$) im Ausgang gespeichert. x ist die Nummer des Ausgangs (des Interfaces). Sie läuft von 0 bis 63.



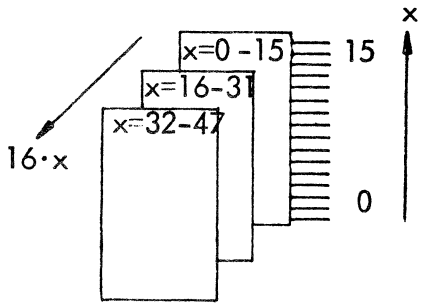
- OUTD(x)

  Der zugewiesene Wert wird als positive Zahl (0. .9999) bewertet und als binär codiertes dezimales Wort mit 4 BCD-Stellen ($10^0 \ldots 10^3$) im Ausgang gespeichert. x ist die Nummer des Eingangs (wie oben).



- OUTB(x)

  Eine Ausgangsleitung wird entsprechend dem zugewiesenen Wert (0 oder 1) gesetzt. Die übrigen Ausgänge des Interfaces bleiben unverändert. x ist die Nummer der Eingangsleitung (0...1023).



Beispiele:

```
200 LET OUTW(X2)=1000+V2
250 LET OUTD(63)=9999
755 LET OUTB(5*X)=K1 OR K2
```

1000 + V2 nach Ausgang X2
9999 in BCD nach Ausgang 63
Bit-Ausgang 5·X entsprechend (k1vk2) setzen

Eine zweite Version der Systemvariablen OUTW(x), OUTD(x) und OUTB(x) ist Einkarten-Interfaces vom Typ PSSA 32 zugeordnet (je 32 Ausgangsleitungen). Funktionell stimmt sie mit der Standard-Version überein, jedoch ist zu beachten, daß je Interface 2 Gruppen von 16-bit-Ausgängen vorhanden sind:

OUTW(x):	Je Interface 2 16 bit-Worte mit aufeinanderfolgenden Nummern (z.B. x = 0...1)
OUTD(x):	Je Interface 2 mal 4 BCD-Stellen mit aufeinanderfolgenden Nummern (z.B. x = 0...1)
OUTB(x):	Je Interface 32 binäre Ausgänge mit aufeinanderfolgenden Nummern (z.B. x = 0...31).

Zähleingänge (PIZE)

Für Zähleingänge sind vorgesehen:

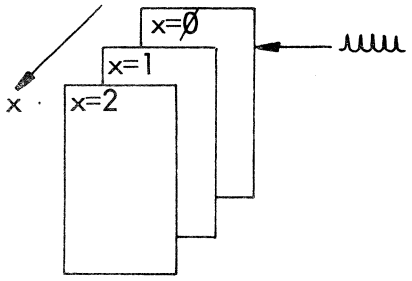
die Systemvariablen	OUTC(x)	Out Counter
	INC(x)	In Counter
die Systemprozeduren	ACTC(x)	Activate Counter
	HLTC(x)	Halt Counter

OUTC(x) und INC(x) werden im Programm wie einfache indizierte Variablen benutzt, während ACTC(x) und HLTC(x) durch das Statement CALL aufgerufen werden. x ist die Nummer des Zähleingangs; sie läuft von 0 bis 15.

Damit können bis zu 16 Zähleingänge (Einkarten-Interfaces vom Typ PIZE 16) bedient werden, die jeweils einen 16 bit-Zähler enthalten (Kapazität 0...65535 Impulse).

- OUTC(x)

  Der Zähler wird auf den Wert gesetzt, der ihm über LET zugewiesen wird.



- INC(x)

  Liefert den Inhalt des Zählers, der dabei unverändert bleibt.

- m CALL ACTC(x)

  Öffnet den Zähleingang für von außen kommende Impulse.

- m CALL HLTC(x)

  Schließt den Zähleingang.

Beispiel:

```
10 LET OUTC(2)=0          Zähler 2 nullstellen
20 CALL ACTC(2)           Eingang öffnen
30 AFTER 1000 : GOTO 100  Nach 1000 ms:
...
100 CALL HLTC(2)          Eingang schließen
110 LET ZLR=INC(2)        Zählinhalt nach ZLR
120 STOP
```

Bemerkungen:
Bei Überlauf ($\geq 2^{16}$ Impulse) kann ein Interrupt ausgelöst werden, der mit ON INT zu verarbeiten ist. Auf diese Weise lassen sich größere Impulsmengen zählen. Die Zuordnung des Interrupts zum Zähleingang wird systemabhängig vorgenommen (Option).
Abfrage des Zählinhalts über INC(x) stets bei geschlossenem Zähleingang empfohlen.

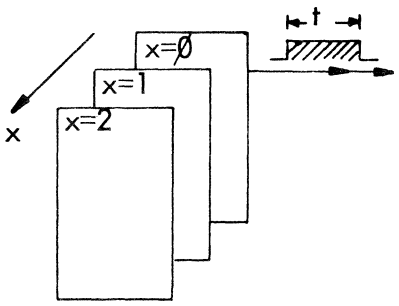
Zeitausgänge (PISA)

Für Zeitausgänge sind vorgesehen

die Systemvariable OUTT(x)	Out Timer
die Systemprozedur ACTT(x)	Activate Timer

OUTT(x) wird im Programm wie eine einfach indizierte Variable benutzt, während ACTT(x) durch des Statement CALL aufgerufen wird. x ist die Nummer des Zeitausgangs; sie läuft von 0 bis 15.

Damit können bis zu 16 Zeitausgänge (Einkarten-Interfaces vom Typ PISA 16) bedient werden, die jeweils einen Quarz und einen 16-bit-Zähler enthalten. Sie dienen zur Ausgabe eines Steuersignals, dessen Zeitdauer zwischen 0 und 65535 Zeiteinheiten liegen kann (Auflösung 0.1 oder 1 µs).



- OUTT(c)

  Setzt den Zeitausgang auf die gewünschte Dauer t; der Wert wird mit LET zugewiesen.

- m CALL ACTT(c)

  Löst die Ausgabe des Steuersignals aus.

Beispiel:

```
5Ø LET OUTT(Ø)=TØ+2ØØØ
6Ø CALL ACTT(Ø)
```

Zeitdauer 0 auf TØ + 2ØØØ setzen
Auslösung des Signals

Bemerkungen:

Mit Ablauf der Zeit (Ende des Signals) kann ein Interrupt ausgelöst werden, der mit ON INT zu verarbeiten ist. Die Zuordnung des Interrupts zum Zeitausgang wird systemabhängig vorgenommen (Option).

Die Anzahl der Zeitausgänge und Zähleingänge pro System ist auf insgesamt 16 begrenzt; sie werden von 0 bis 15 numeriert und können in beliebiger Kombination verwendet werden.

Einkanal-Analog-Eingänge (ADE)

Diese Eingänge werden durch die Systemvariable

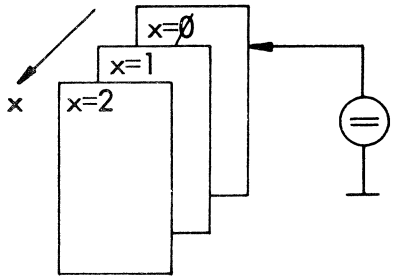
INA(x) In Analog

beschrieben. Sie wird im Programm wie eine einfach indizierte Variable behandelt und liefert im Augenblick der Abarbeitung die am Eingang x anstehende Spannung als Zahlenwert.

Sie bezieht sich auf einen von bis zu 16 Analog-Digital-Umsetzern (Einkarten-Interfaces vom Typ ADE 12... mit 12 bit Ausgang). Der digitalisierte Meßwert liegt unabhängig vom Meßbereich zwischen 0 und 4095.

- INA(x)

Die anliegende Spannung wird digitalisiert und als Zahlenwert übergeben.
x ist die Nummer des Eingangs; sie läuft von 0 bis 15.



Beispiele:

```
400 LET MES2=INA(Ø)/Ø.4Ø96
1Ø5 IF INA(2)>511 THEN 22Ø
62Ø PRINT INA(1)
```

Meßeingang 0 in mV nach MES2
Sprung nach 220 wenn Eingang z > 511 Teile
Meßeingang 1 ausdrucken

Bemerkung:

Der über INA(x) abgefragte Wert ist ggfs. zu skalieren, um den Spannungswert in V oder mV zu erhalten. Im ersten Beispiel ist ein ADU mit 0...+10 V Meßbereich angenommen; der Meßwert wird auf mV skaliert.

Einkanal-Analog-Ausgänge (DAU/DAI)

Diese Ausgänge werden durch die Systemvariable

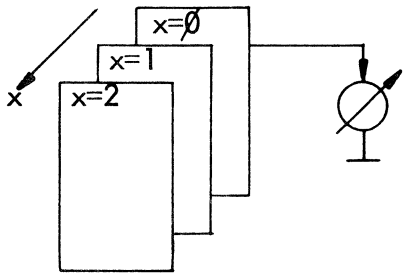
OUTA(x) Out Analog

beschrieben. Sie wird im Programm wie eine einfach indizierte Zahlenvariable behandelt und setzt im Augenblick der Abarbeitung den Ausgang x auf den mit LET zugewiesenen Wert.

Sie bezieht sich auf einen von bis zu 16 Digital-Analog-Umsetzern (Einkarten-Interfaces vom Typ DAU10... bzw. DAI 10... mit 10 bit Auflösung). Der auszugebende Wert liegt unabhängig vom Ausgangsbereich zwischen 0 und 1023.

- OUTA(x)

Der zugewiesene Wert wird in den DAU übertragen und dessen Ausgangssignal entsprechend gesetzt. x ist die Nummer des Ausgangs; sie läuft von 0 bis 15.



Beispiel:

```
1Ø55 LET OUTA(Ø)=ANA*1Ø2.4
```

Ausgang Ø auf ANA (in V) setzen

Bemerkung:

Der über OUTA(x) auszugebende Wert ist ggfs. zu skalieren, um Spannungs- oder Stromwerte, die intern in V, mV oder mA vorliegen, in die richtige Ausgangsgröße zu verwandeln. Im obigen Beispiel ist ein Ausgangsbereich von 0...+10 V angenommen; die Variable ANA enthält einen Spannungswert in V.

Analog-Meßsystem (ADM 621)

Zur Bedienung des mittelschnellen Analog-Meßsystems vom Typ ADM 621 mit eingebautem Multiplexer dienen die Systemprozeduren

ADCS (a,k,q)	A/D-Converter	Single
ADCD (a,k,q)	"	Double
ADCM(a,k,q)	"	Multiple

Sie werden mit CALL aufgerufen, lösen q aufeinanderfolgende Messungen aus und legen die q Meßwerte in einem Datenfeld ab, das bei a beginnt. k ist die Nummer des benutzten bzw. des ersten benutzten Meßkanals.

Die übernommenen Zahlenwerte entsprechen der Auflösung des Meßsystems (z.B. 0...4095 bei 12 bit).

- m CALL ADCS(a,k,q)

  Der Kanal k wird q-mal nacheinander gemessen; die q Werte stehen nachher im Feld a.

- m CALL ADCD(a,k,q)

  Die Kanäle k und k+1 werden abwechselnd insgesamt q-mal gemessen; die q Werte stehen nachher im Feld a. Die Nummer k des Basis-Kanals muß ganzzahlig sein.

- m CALL ADCM(a,k,q)

  Die Kanäle k, k+1, ..., k+q-1 werden nacheinander gemessen; die q Werte stehen nachher im Feld a.

Beispiele:

```
100 DIM MESS(99)
200 CALL ADCS(MESS(0),5,100)
300 CALL ADCD(MESS(0),20,N)
420 CALL ADCM(MESS(50),10,50)
```

Feld MESS mit 100 Plätzen
Kanal 5 100mal messen
Kanäle 20 und 21 N-mal abwechselnd messen
Kanäle 10...59 messen und ab Platz 50 ablegen

Bemerkung zu den Parametern:

a = Feldname (Array); ist mit DIM zu reservieren.
k = Zahl (0...63))
q = Zahl (1...256)) als Konstante, Variable oder arithmetischer Ausdruck angebbar.

Integrierendes Meßsystem (ADI/ADA)

Zur Bedienung der integrierenden Analog-Meßsysteme vom Typ ADI 200, ADI 210, ADA 203 und ADA 213 sowie des vorgeschalteten Meßstellenumschalters MUI dienen die Systemprozeduren

ADIS (a,k,q,s)	Einkanal-Messung
ADID (a,k,q,s)	Zweikanal-Messung
ADIM(a,k,q,s)	Mehrkanal-Messung

Sie werden mit CALL aufgerufen, lösen q aufeinanderfolgende Messungen aus und legen die q Meßwerte in einem Zahlenfeld ab, das bei a beginnt. k ist die Nummer des benutzten bzw. des ersten benutzten Meßkanals.

Der Parameter s definiert den Anfang eines Zahlenfeldes, in dessen 4 Plätzen folgende Steuerparameter stehen:

s(0) Meßgröße:	1 = Gleichspannung (Volt) 2 = Widerstand (Ohm) 3 = Wechselspannung (Volt)
s(1) Meßbereichs-Vorwahl:	0 = 1000 V≃ / 10 MOhm 1 = 100 " / 1 " 2 = 10 " / 100 kOhm 3 = 1 " / 10 " 4 = 0.1 " / 1 " 5 = 0.01 V=
s(2) Meßfolge:	1 = 25 Messungen/s 2 = 100 " 3 = 10 "
s(3) Meßgeschwindigkeit:	Ø = langsam 1 = schnell

- m CALL ADIS (a,k,q,s)

  Der Kanal k wird q-mal nacheinander gemessen.

- m CALL ADID (a,k,q,s)

  Die Kanäle k und k+1 werden abwechselnd insgesamt q-mal gemessen.

- m CALL ADIM (a,k,q,s)

  Die Kanäle k, k+1, ... k+q-1 werden nacheinander gemessen.

Beispiele:

```
125 CALL ADIS (A(Ø),13,1Ø,S(Ø))
25Ø CALL ADID (MESS(R,Ø),KAN,ZAHL,STF(Ø))
375 CALL ADIM (B(2),X,7,R(Ø))
```

Peripheral Ein-/Ausgabe

BASEX bedient periphere Ein-/Ausgabegeräte, wie z.B. das Konsol-Terminal, Schnelldrucker usw.

Die Anweisungen INPUT und PRINT (siehe 2.2.3.7/2.2.3.8) dienen zur Ein- bzw. Ausgabe über diese Geräte.

Darüberhinaus gibt es in BASEX eine Reihe von Systemprozeduren, die diese Geräte in anderer Weise behandeln oder Geräte mit nicht zeichenweisem Datentransfer bedienen.

Sie werden konfigurationsabhängig implementiert und sind in den folgenden Abschnitten beschrieben.

Systemprozedur READ

Mit der Systemprozedur

READ (d, q$)

wird von einem Gerät mit der Nummer d ein Zeichenstring in die String-Variable q$ gelesen. Die Anzahl der Zeichen entspricht der Länge von q$ laut Definition in der CHAR-Anweisung. Die Zeichen werden ohne Parity im String abgelegt.

- m CALL READ (d,q$)

Eingabe eines Strings aus Gerät d nach q$.

Beispiel:

```
20 CHAR ADDR$(70)
1010 CALL READ (1,ADDR$)
```

Bemerkung:

Die Systemprozedur READ hat eine ähnliche Funktion wie die Anweisung INPUT. Jedoch bestehen folgende Unterschiede:

- Das Eingabegerät d muß nicht vorher mit PRINT DEV(d) spezifiziert werden, sondern wird unmittelbar als Parameter angegeben.
- Vor Eingabe wird kein Fragezeichen (?) ausgegeben, um den Bediener zur Eingabe aufzufordern.
- Es werden alle Zeichencodes gelesen (also z.B. auch Komma, das bei INPUT als Trennzeichen dient).
- Die Anzahl der Zeichen ist gleich der in CHAR reservierten Länge des Strings. Vorzeitiger Abbruch der Eingabe ist nicht möglich; ebenso nicht die Eingabe überzähliger Zeichen.

Magnetband-System (MBE-621)

Zum Betrieb von Magnetband-Laufwerken mit Controllern vom Typ MBE-621 dienen die Systemprozeduren

RF (u,a,l)	Read File
WF (u,a,l)	Write File
RFS (u,a,l)	Read File String
WFS (u,a,l)	Write File String
WFM(u)	Write Filemark
BSP (u,n)	Backspace
FWP (u,n)	Forward
BSPF (u)	Backspace to Filemark
FWDF(u)	Forward to Filemark
REW (u)	Rewind

Sie werden über CALL aufgerufen. Das Laufwerk wird mit u bezeichnet (Ø...3). Das Speicherfeld wird mit a angegeben (Zahlenfeld bzw. Stringvariable), die Länge des zu lesenden bzw. zu schreibenden Blocks mit l (Zahlen bzw. Zeichen). Der Parameter n gibt die Anzahl der Blöcke an.

Außerdem ist für die Fehlerbehandlung die Systemvariable

TERR	Tape Error

vorgesehen.

- m CALL RF (u,a,l)

  Lesen von l Zahlen (je 4 Byte) vom Band in Zahlenfeld a.

- m CALL WF (u,a,l)

  Schreiben von l Zahlen (je 4 Byte) aus Zahlenfeld a auf das Band.

- m CALL RFS (u,a.l)

  Lesen von l Zeichen vom Band in Stringvariable a.

- m CALL WFS (u,a,l)

  Schreiben von l Zeichen aus Stringvariable a auf das Band.

- m CALL RFM (u)

  Schreiben einer Filemark.

• m CALL BSP (u,n)

Band um n Blöcke zurückspulen.

• m CALL FWD (u,a)

Band um n Blöcke vorlaufen lassen.

• m CALL BSPF (u)

Band bis zur nächsten Filemark zurückspulen.

• m CALL FWDF (u)

Band bis zur nächsten Filemark vorlaufen lassen.

Bemerkungen:

Bei BSP und FWD zählt eine Filemark als Block (keine Fehlermeldung). Bei n = 0 wird das Band zur nächsten Filemark gespult (Funktion wie bei BSPF bzw. FWDF).

Nach Ausführung von BSPF steht das Band vor, nach Ausführung von FWDF hinter der Filemark.

• TERR

Liefert einen Fehlercode, dessen Zahlenwert folgenden Fehlerarten entspricht:

0	Kein Fehler
1	Laufwerk nicht on-line
2	" nicht bereit
3	Speicherfeld-Adressen zu klein
4	Bandanfang (BOT)
5	Parity-Fehler
6	Blocklänge <16 byte
7	Schreibversuch trotz Schreibsperre
8	Keine Betriebsart erkannt oder kein Band aufgelegt
9	Blocklänge zu groß
10	Filemark gelesen
13	Bandende (EOT)
>13	EOT + weiterer Fehler

Beispiele:

```
100 CALL RF (0,A(0),100)
120 CALL WF (1,BER(3,0),25)
150 CALL RFS (X,T$,64)
170 CALL WFS (0,M2$,ZZ)
200 CALL RFM (0)
210 CALL BSP (2,8)
215 IF TERR<3 OR TERR>16 THEN 230
220 GO TO TERR OF 800,810,...
230 ...
```

Spezielle Bildschirm-Befehle (BTH 2000)

BASEX kann (als Option) um eine Reihe von Steuerbefehlen für alphanumerische Bildschirm-Terminals vom Typ BTH 2000 erweitert werden:

DISC (x,y)	Set Cursor
DICH	Cursor Home
DILD	Line Delete
DILI	Line Insert
DICS	Clear Screen
DICF	Clear Foreground
DISB	Set Background
DISF	Set Foreground
DIXM	X-mit
DISP	Set Print

• m CALL DISC (x,y)

Positionieren des Cursors auf eine vorgegebene Spalte x einer angegebenen Zeile y. x kann den Wert Ø...73, y den Wert Ø...26 annehmen. Der Wert Ø bedeutet die erste Spalte bzw. Zeile.

Beispiele:

```
10 CALL DISC (10,11)
```

Setzen des Cursors auf die 11. Spalte und 12. Zeile

```
20 LET X = A+B,Y=C*D
30 CALL DISC (X,Y)
```

Setzen des Cursors auf die Position, die durch x, y vorgegeben wird.

• m PUT DICH

Positionieren des Cursors in Ausgangsstellung (erste Spalte der ersten Zeile).

• m PUT DILD

Löschen der Zeile über dem Cursor. Alle Zeilen unter der gelöschten Zeile werden um eine Zeilenposition nach oben geschoben. Am unteren Rand des Bildschirms erscheint eine Leerzeile.

• m PUT DILI

Einfügen einer Zeile. Es werden alle Zeichen unterhalb des Cursors um eine Zeile nach unten verschoben. Die unterste Zeile auf dem Bildschirm geht verloren. Der Cursor springt an die erste Position der eingefügten Leerzeile.

- m PUT DICS

  Löschen des Bildschirm-Inhaltes, verbunden mit Positionieren des Cursors in Ausgangsstellung.

- m PUT DISB

  Alle diesem Befehl folgenden Zeichen werden als Hintergrundzeichen dargestellt.

- m PUT DISF

  Alle diesem Befehl folgenden Zeichen werden als Vordergrundzeichen dargestellt.

- m PUT DIXM

  Übertragung des Bildschirm-Inhaltes zum Rechner (nur im "Batch Mode"). Es werden alle Vordergrund-Zeichen vom letzten Transmit-Zeichen an übertragen. Sofort danach muß eine Eingabe programmiert sein.

  Beispiel:

```
250 PUT DIXM
260 CALL READ (3,A$)
```

- m PUT DISP

  Übertragung des Bildschirm-Inhaltes auf das dazugehörige Hardcopy-Gerät.

- m PUT DICF

  Löschen der Vordergrunddaten. Es werden alle hell dargestellten Zeichen im Vordergrund des Bildschirms durch Blanks (Leerzeichen) ersetzt, und der Cursor springt in die Ausgangsstellung.

Graphische Ausgabe

Zum Betrieb von graphischen Ausgabegeräten dienen die Systemprozeduren

HOME
PLOT (x,y,z)
SYMB (h,aØ)

Sie werden über CALL aufgerufen.

Als Ausgabegerät kann angeschlossen sein:

- ein Speicheroszillograph
- ein XY-Schreiber, oder
- ein Inkremental-Plotter

In den beiden ersten Fällen ist das Gerät über ein Interface angeschlossen, das für jede der beiden Koordinaten x (Breite) und y (Höhe) einen 10-bit-Digital-Analog-Wandler enthält (Auflösung 1024). Eine Einheit bei der Angabe von x, y und h entspricht hier also 1/1000 der vollen Schreibbreite bzw. -höhe.

Die Einheit für die Ausgabe beim Inkremental-Plotter entspricht der Schrittweite des Gerätes.

- m CALL HOME

Die Ausgangsstellung (Koordinaten-Nullpunkt) wird eingenommen. Im Falle des Speicheroszillographen wird außerdem der Bildschirm gelöscht.

- m CALL PLOT (x,y,z)

Läßt den Strahl bzw. den Schreibstift vom jeweiligen Ausgangspunkt zum Zielpunkt mit den Koordinaten (x,y) wandern.

Ist z = Ø, so geschieht dies nichtschreibend.
Ist z = 1, so wird zwischen Ausgangs- und Zielpunkt linear interpolierend geschrieben.

- m CALL SYMB (h, aØ)

Schreibt den Textstring aØ (große Buchstaben, Ziffern, Sonderzeichen). Der Schriftzug wird waagerecht geschrieben, d.h. in Richtung der Koordinate x. Die linke untere Ecke des ersten Schriftzeichens ist die Ausgangsposition, d.h. die jeweilige Stellung des Strahls bzw. der Schreibfeder vor dem Aufruf von SYMB. Der Parameter h ist für die Schrifthöhe maßgebend; die Schriftzeichen haben die Höhe 7h. Der Rasterabstand der Zeichen beträgt ebenfalls 7h.

Beispiele:

```
100 CALL HOME
110 CALL PLOT (5,10,0)
120 CALL SYMB(1,"BASEX")
```

Kartenleser (MDS 6042)

Zum Betrieb des Lochkarten-Stapellesers MDS 6042 ist die Systemprozedur

CARD (d,a,f)

vorgesehen. Sie wird mit CALL aufgerufen und bewirkt das Lesen einer 80-spaltigen Lochkarte. Die Daten der Lochkarte werden in ein Zeichenfeld (Stringvariable) a gelesen, wo sie in Form von ASCII-Zeichen stehen und als String weiterverarbeitet werden können.

Mit d wird die Geräte-Nummer angegeben. Durch f wird ein Speicherplatz (dimensionierte Variable) definiert, in dem ein Fehler-Code steht:

Fehler-Code	0	kein Fehler
	1	Zeichen EOF (End of File) gelesen
	2	Kartenschacht leer bzw. zu voll
	3	Einziehfehler
	4	Lesefehler
	5	Parity-Fehler

Der Fehlercode kann nach Lesen einer Karte abgefragt werden.

• m CALL CARD (d,a,f)

Lesen einer Lochkarte am Gerät d nach Stringvariable a. Ein eventueller Fehler steht in f.

Beispiel:

```
100 CHAR KS(80)
...
200 CALL CARD (16,KS,F(0))
210 GOSUB 800
...
800 IF F(0)=0 THEN 820
810 GOTO F(0) OF 830,840,850,860,870
820 RETURN
830 PRINT "ERROR EOF"
835 RETURN
...
```

Sonstige Systemvariablen und -prozeduren

Im folgenden sind in BASEX benutzbare Systemvariablen und -prozeduren beschrieben, die allgemeinen Charakter haben, den Sprachumfang von BASEX ergänzen und bei Bedarf implementiert werden.

Systemprozeduren LDST und STST

Die Systemprozeduren

LDST (a, b$, l)	Load String
STST (a, b$, l)	Store String

dienen zum Datenaustausch zwischen einem Zahlenfeld a und einer String-Variablen b$ der Länge l. Sie werden mit CALL aufgerufen.

Das Zahlenfeld a ist mit DIM, der String b$ mit CHAR zu reservieren. Da jeweils 4 String-Zeichen einer Zahl entsprechen, sollte die Länge l ein ganzzahliges Vielfaches von 4 sein (l = 4, 8, 12, ...).

- m CALL LDST (a, b$, l)

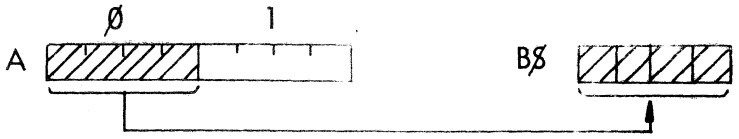
Der String b$ wird mit dem Inhalt des Zahlenfeldes a geladen; es werden l Bytes übertragen.

- m CALL STST (a, b$, l)

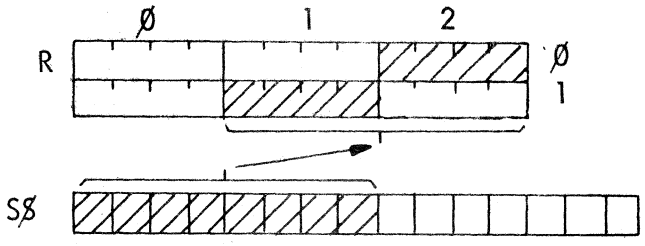
Der Inhalt des Strings b$ wird im Zahlenfeld a gespeichert; es werden l Bytes übertragen.

Beispiele:

```
100 DIM A(1)
110 CHAR B$(4)
200 CALL LDST(A(0),B$,4)
```




```
300 DIM R(1,2)
310 CHAR S$(15)
320 CALL STST(R(1,1),S$,8)
```



Bemerkung zu den Parametern:

a = ein- oder zweifach indizierte Zahlen-Variable
b$ = String-Variable
l = Anzahl der übertragenen Bytes (4, 8, 12, ...)

a kann ein beliebiges Element des Zahlenfeldes sein; es wird von dort aus übertragen. Die Länge von b$ kann größer als l sein; die Übertragung beginnt stets mit dem ersten Zeichen von b$.

Systemgenerierung Lochstreifen-Systeme

Jeder Benutzer kann eine spezielle Konfiguration des BASEX-Systems vornehmen, die seinen Bedürfnissen entspricht.

Für Lochstreifen-orientierte Anlagen erhält er einen Programmstreifen, in dem das gesamte BASEX-System enthalten ist. Im Dialog wählt er die Funktionen, Systemvariablen und -prozeduren aus, die er für seine Aufgabe braucht; die übrigen werden überlesen.

Zum Schluß kann der Benutzer weitere verfügbare oder auch selbst erstellte Routinen einlesen.

Auf diese Weise entsteht ein speicheroptimales BASEX-System.

Die Systemgenerierung erfordert folgende Manipulationen:

Taste RS betätigen, Schalter BS einlegen. Lader-Lochstreifen (im RUBOUT-Bereich) in Teletype-Leser oder in schnellen Leser einlegen (im letzteren Fall zusätzlich Schalter 4 einlegen). Taste ST betätigen. Jetzt wird der Lader in die Plätze 'Ø6 bis 'FF des RAMs eingelesen.

Nach Halt des Lochstreifenlesers sind Schalter BS und 4 (falls erforderlich) in Normalstellung zu bringen. Taste ST betätigen. Nach dem Einlesen meldet sich der Rechner mit '*** BASEX***'. Auf die nun folgenden Fragen ist für Ja mit 'Y', für Nein mit 'N' zu antworten.

Es folgt der Dialog zwischen Benutzer und System (Beispiel mit Erklärungen siehe nächste Seite). Zum Schluß fragt das System nach der Eingabe weiterer Routinen.

Wird diese Frage negativ beantwortet, meldet sich der Rechner mit '*READY'. Bei positiver Antwort müssen jetzt die einzelnen Systemprozeduren über den Leser eingelesen werden. Das geschieht folgendermaßen: Die Lochstreifen müssen im RUBOUT-Bereich auf den Lochstreifenleser gelegt werden, und die Start-Taste ist so oft zu betätigen, bis der Lochstreifen ganz eingelesen ist. Nach dem Einlesen der letzten Prozedur noch einmal die Taste ST betätigen, worauf sich der Rechner mit '* READY' meldet.

Beispiel für Systemgenerierungs-Dialog:

```
MEMORY SIZE 48K ? Y/N    (KERNSPEICHERGROESSE 48K)
MEMORY SIZE 32K ? Y/N    (KERNSPEICHERGROESSE 32K)
FAST PUNCHER ?Y/N        (SCHNELLER LOCHER)
FAST READER ? Y/N        (SCHNELLER LESER)
DYNAMIC INTERRUPT INPUTS ? Y/N   (INTERRUPT-EINGAENGE)
   BEI Y: ABFRAGE AUF LEVEL UND EINGAENGE
   BEI N: NAECHSTE FRAGE:
LOG           ? Y/N      (NAT. LOGARITHMUS)
EXP           ? Y/N      (E-FUNKTION)
SQR           ? Y/N      (QUADRATWURZEL)
TAN COS SIN ? Y/N        (TANGENS COSINUS SINUS)
ATN           ? Y/N      (ARCUSTANGENS)
RND           ? Y/N      (ZUFALLSZAHLENGENERATOR)
MSEC ? Y/N  (MILLISEKUNDEN)
SEC  ? Y/N  (SEKUNDEN)
MIN  ? Y/N  (MINUTEN)
HOUR ? Y/N  (STUNDEN)
LEV  ? Y/N  (LEVEL)
INB  ? Y/N  (BITWEISER DIGITALER EINGANG)
INW  ? Y/N  (WORTWEISER DIGITALER EINGANG)
IND  ? Y/N  (WORTWEISER DIGITALER BCD-EINGANG)
OUTB ? Y/N  (BITWEISER DIGITALER AUSGANG)
OUTW ? Y/N  (WORTWEISER DIGITALER AUSGANG)
OUTD ? Y/N  (WORTWEISER DIGITALER BCD-AUSGANG)
INA  ? Y/N  (EINKANAL-ANALOGEINGANG)
OUTA ? Y/N  (EINKANAL-ANALOGAUSGANG)
INC  ? Y/N  (ZAEHLER-INHALT ABFRAGEN)
OUTC ? Y/N  (ZAEHLER-INHALT SETZEN)
OUTT ? Y/N  (ZEITZAEHLER-INHALT SETZEN)
ACTC ? Y/N  (ZAEHLER AKTIVIEREN)
HLTC ? Y/N  (ZAEHLER ANHALTEN)
ACTT ? Y/N  (ZEITZAEHLER AKTIVIEREN)
LDST ? Y/N  (STRING LADEN AUS REAL)
STST ? Y/N  (REAL LADEN AUS STRING)
READ ? Y/N  (EINGABE STRING)
DISP ? Y/N  (BILDSCHIRM-INHALT AUSDRUCKEN)
DIXM ? Y/N  (BILDSCHIRM-INHALT SENDEN)
DISF ? Y/N  (UMSCHALTUNG AUF VORDERGRUND-DARSTELLUNG)
DISB ? Y/N  (UMSCHALTUNG AUF HINTERGRUND-DARSTELLUNG)
DICF ? Y/N  (VORDERGRUND-DATEN LOESCHEN)
DICS ? Y/N  (BILDSCHIRM LOESCHEN)
DILI ? Y/N  (ZEILE EINFUEGEN)
DILD ? Y/N  (ZEILE LOESCHEN)
DICH ? Y/N  (CURSOR IN GRUNDSTELLUNG)
DISC ? Y/N  (CURSOR POSITIONIEREN)
STNU ? Y/N  (STATEMENT-NUMMER)
ERR  ? Y/N  (FEHLER-VARIABLE)
STIM ? Y/N  (BASEX-UHR SETZEN)
SUBROUTINE PROCEDURE ? Y/N
```

Einbau von Routinen in BASEX

Der Benutzer kann für das BASEX-System Maschinencode-Programme (Routinen) erstellen, die er bei der Systemgenerierung hinzufügt (s. 8.1.2.1).

Den Routinen stehen die Register 'Ø2 bis '1F zur Verfügung. Als Variablenspeicher sollten nur Register benutzt werden, damit die Routine reenterable wird.

Es gibt verschiedene Arten von Routinen:

- für einfache Systemvariablen
- für indizierte Systemvariablen
- für Systemprozeduren (CALL ...)

1. Einfache Systemvariable:

   Am Anfang der Routine stehen 8 Kennbytes:

   a) In den beiden ersten Bytes wird die Länge der Routine (ohne die 8 Kennbytes) als 2-Byte-Hexa-Zahl angegeben.

   b) Im dritten Byte steht als Kennung Hexazahl '59.

   c) Das vierte Byte enthält die Definition der Transferrichtung ~~(als Hexazahl)~~:

für INPUT-Typ:	Ø	(Hexa 'ØØ)
für OUTPUT-Typ:	-1	(Hexa 'FF)

   d) In den restlichen vier Bytes wird der Name der Routine angegeben; wenn weniger als vier Zeichen benötigt werden, müssen die restlichen Bytes mit Leerinhalt versehen werden.

   Es folgt die erste Instruktion der Routine; an diese Stelle springt das Programm nach dem Aufruf durch das BASEX-Programm.

   Soll ein Datenaustausch mit dem BASEX-Programm stattfinden, so ist die Gleitkomma-Zahl über die Register 'Ø2...'Ø5 (Akku @ + 3 weitere Plätze, in für Zahlen üblicher Lage) auszutauschen. Bei Variablen vom OUTPUT-Typ steht sie dort nach dem Aufruf; bei Variablen vom INPUT-Typ ist sie vor dem Rücksprung dorthin zu bringen.

   Der Rücksprung aus der Routine zurück in das BASEX-Programm erfolgt durch den Befehl JPX,,,'7C.

Beispiel:

Routine für einfache Systemvariable vom INPUT-Typ (LEV):

```
0000                    0    ,      0;
0001          /
              KENNBYTES;
0002 0000          2* H    ,'0009;
                                                  09 00
0003 0002             H    ,'59;
                                                  59
0004 0003             D    ,      0;              00
0005 0004          3* T    ,"LEV";
                                                  4C 45 56
0006 0007             D    ,      0;              00
0007          /
              ;
0008 0008          4(*LDC  ,@    ,      0/        1B 04 80 00
              EINSPRUNG;
0009 000C             GL   ,@    ;                24
0010 000D             ANC  ,@    ,'0F    ;        B0 0F
0011 000F             JPX  ,     ,       ,'7C /   F2 7C
              RUECKSPRUNG;
0012          /
              ;
0013                  Z    ;
```

2. Indizierte Systemvariable

Am Anfang stehen 8 Kennbytes:

a) Siehe 1.a).

b) Im dritten Byte steht die Kennung als Hexazahl '5A.

c) Siehe 1.c).

d) Siehe 1.d).

Der übrige Aufbau ist identisch mit 1); jedoch erfolgt vom BASEX-Programm die Übergabe eines Index' als 2-Byte-Ganzzahl in den Registern '09, '0A.

Beispiel:

Routine für indizierte Systemvariable vom INPUT-Typ (INB):

```
0000                    0    ,      0;
0001          /
          KENNBYTES;
0002 0000           2* H    ,'0028;                    28 00
0003 0002              H    ,'5A;                      5A
0004 0003              D    ,      0;                  00
0005 0004           3* T    ,"INB";                    49 4E 42
0006 0007              V    ;
0007          /
         ;
0008          /
         ;
0009 0008           4(*LDC ,@    ,      0/             1B 04 80 00
          EINSPRUNG;
0010 000C              LDCD,    6,'03F0 ;              1E 81 06 F0 03
0011 0011           2=*ANR ,    6,      9;             1E B5 06 09
0012 0015              BNOC,    9,      8,A    ;       79 09 00 04
0013 0019              ORC ,    6,      1;             C1 06 01
0014 001C A    :       LDA ,@    ,'3001 ,    6;       8E 01 30 06
0015 0020              ANC ,    9,      7;             B1 09 07
0016 0023              EOC ,    9,     -1;             D1 09 FF
0017 0026 B    :       IZ  ,    9,       ,C    ;       43 09 04
0018 0029              SRO ,@    ;                     28
0019 002A              JPL ,     ,B      ;             F8 FB
0020 002C C    :       ANC ,@    ,      1;             B0 01
0021 002E              JPX ,     ,       ,'7C /        F2 7C
          RUECKSPRUNG;
0022          /
         ;
0023                  Z    ;
```

3. Systemprozeduren (CALL...)

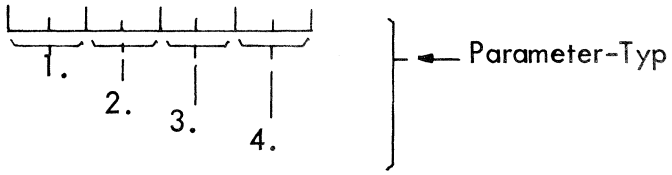
Im Fehlerfall kann das Register '7B vom Benutzer gesetzt werden, und im BASEX-programm mit der Systemvariablen ERR abgefragt werden.

Am Anfang stehen 8 Kennbytes:

a) Siehe 1.a).

b) Im dritten Byte steht als Kennung die Hexazahl '5C.

c) Das vierte Byte enthält die Definition der 4 Parameter-Typen (2 bit je Parameter):



Die Art der Parameter ist als Bitmuster festzulegen und dann als 1-Byte-Hexazahl anzugeben.

Es gilt für:	kein Parameter (NOP)	ØØ
	Zahl (Real)	Ø1
	String	1Ø
	Zahlenfeld (Array)	11

Beispiel: | 1 1 | 1 Ø | Ø 1 | Ø Ø | = 'E4

Array String Real NOP

Registeradressen:

für 1. Parameter		'6B | '6A | '69 | '68
für 2.	"	'6F | '6E | '6D | '6C
für 3.	"	'73 | '72 | '71 | '7Ø
für 4.	"	'77 | '76 | '75 | '74

Kernspeicherzuweisung für Real

N+3	N+2	N+1	N

Exponent — Mantisse

für String

N+3	N+2	N+1	N

Basisadresse des Strings — Länge des Strings

für Array

N+3	N+2	N+1	N

Anfangsadresse des Arrays

Die Parameter werden bei Aufruf der Systemprozedur durch das BASEX-Programm in die Register übergeben.

d) Siehe 1.d).

Es folgt die erste Instruktion der Routine; an diese Stelle springt das Programm nach dem Aufruf durch das BASEX-Programm.

Der Rücksprung aus der Routine zurück in das BASEX-Programm erfolgt durch den Befehl JPX,,,'5Ø.

Beispiel:

Routine für Systemprozedur (LDST).

```
0000                    O    ,       0;
0001          /
              KENNBYTES;
0002 0000            2* H    ,'001D;
                                                  1D 00
0003 0002               H    ,'5C;
                                                  5C
0004 0003               H    ,'E4;
                                                  4E
0005 0004            4* T    ,"LDST";
                                                  4C 44 53 54
0006          /
              ;
0007 0008 A   :         LDX  ,@    ,        ,'68 / 82 68
              EINSPRUNG;
0008 000A               STX  ,@    ,        ,'6E ; E2 6E
0009 000C               CSL  ;     4,B      ;      F9 04 03
0010 000F               JPL  ,       ,A     ,      F8 F8
0011 0011 B   :         ADCD,'6E   ,      1;      1E 91 6E 01 00
0012 0016               ADCD,'68   ,      1;      1E 91 68 01 00
0013 001B               SBC  ,'70  ,      1;      A1 70 01
0014 001E               BZ   ,'70  ,       ,C     ; 41 70 03
0015 0021               JPX  ,     ,       ,    4; F2 04
0016 0023 C   :         JPX  ,     ,       ,'50 / F2 50
              RUECKSPRUNG;
0017          /
              ;
0018                    Z    ;
```

BASEX-Fehlerliste

Nachstehend sind alle Fehlermeldungen aufgeführt, die vom BASEX-Interpreter während des Kommandobetriebs (K) oder während des Ausführungsbetriebs (R) erkannt werden, mit Angabe des Fehlercodes:

1	K,R	Eingabezeile zu lang (mehr als 72 Zeichen)
2	K	Operator-Stack-Überlauf (zu tiefe Verschachtelung von Ausdrücken)
3	K	Erwartete Ganzzahl nicht im erlaubten Bereich (erlaubt: 1...9999 bei Statement-Nummern und CHAR-Index; 0...9999 bei DIM-Index)
4	K,R	Speicherüberlauf (zu viele Statements, zu große Bereiche, Runtime-Stack-Überlauf (bei verschachtelten Programmen)
5	K,R	Über- oder Unterschreitung des Gleitkomma-Zahlenbereichs
6	K,R	Paritätsfehler bei Eingabe
7	R	Standardfunktion bei gegebenem Argument nicht berechenbar
8	R	Standardfunktion wurde bei Systemgenerierung gelöscht
9	K	Mehr als ein Stringoperator auf der rechten Seite der String-Zuweisung
10	K	Erwarteter Operand nicht gefunden oder vom falschen Typ
11	K,R	Erwarteter Operator nicht gefunden oder vom falschen Typ
12	K	Öffnende und schließende Klammern passen in Anzahl und/oder Typ nicht zusammen
13	K,R	Falsche Syntax beim Aufbau eines Operanden (Variable, Funktion, Zahl, String, Formatangabe)
14	K	Illegale Folge von Operatoren und Operanden in Ausdrücken mit Strings
15	K	Argumente bei indizierten Variablen oder Funktionen nicht korrekt oder zu viel
16	R	Bereichsüberschreitung bzw. -unterschreitung bei indizierten Variablen oder bei Teilstrings
17	K,R	Benutzerfunktion, Unterprogramm oder Segment nicht definiert (Name steht nicht in Objekttafel)
18	K	Nicht existierendes Statement oder Kommando
19	R	Sprungziel nicht definiert (nicht vorhandene Anweisungsnummer)
20	K	Nach einem Kommando falsche Angaben oder Fehler bei Datei-Behandlung (z.B. Datei-Name nicht existent oder Platten-Fehler)
21	R	NEXT ohne FOR, RETURN ohne GOSUB, ENDS ohne LINK, ENDS in Root oder falsche Verschachtelung von Schleifen, Unterprogrammen und Segmenten
22	R	DATA-Liste erschöpft oder nicht kompatible Zuordnung in READ oder INPUT
23	R	Im FOR-Statement Schrittweite = 0
24	K	Zuordnung fehlt (oder an falscher Stelle) bei LET, DEF, FOR, EQUI oder EQUO
25	K	In diesem Statement nicht erlaubter String-Ausdruck
26	K	Unerlaubtes Statement nach ON INT oder AFTER
27	K	Parameter in CALL stimmen in Anzahl und/oder Typ nicht mit Definition überein
28	R	Wert des Ausdrucks nach ON INT, AFTER, START, ENAB oder DISAB nicht im erlaubten Bereich
29	R	verbotener Startauftrag
30	K	Neustart nach INIT durch RUN nicht erlaubt
31	K,R	Interpretersegment auf Plattenspeicher fehlt
32	R	Überlauf der START-/AFTER-Tabellen
33	K,R	Fehler Kernspeicher
34	K,R	BUS-Fehler

35	K,R	Netz-Fehler
36	K,R	Kein Unterprogramm vorhanden (z.B. kein Programm für DEV(6) existent)
37	K	Statementnummer im Segment kleiner oder gleich der maximalen " der Root " " " " "
38	K	LINK im Segment
39	R	Segment im LINK-Statement wurde bei INIT nicht geladen
40	K	FILE Länge zu kurz

*) Argument in SQR negativ oder Exponent in EXP zu groß (>88) oder zu klein (<-73) oder Argument in LOG nicht positiv oder beim "Integer-Hoch" Exponent >65535 oder beim "Real-Hoch" negative Basis oder Hochzahl >8388607.