Mikrocomputer-Gerätegeneration

M C 80.3x

- B A S I C - 8 0 . 3 0 -

Sprachbeschreibung

Z.-Nr. 50300-4810.00 BA

VEB Elektronik Gera
Ausgabe 10/85

Inhaltsverzeichnis

Ø. Allgemeine Einführung

BASIC-8Ø.3x ist eine problemorientierte Programmiersprache. Sie ist eine Modifikation der Programmiersprache BASIC, die Anfang der 6Ø-er Jahre in den USA, speziell für die Ausbildung in der Rechnerprogrammierung an Universitäten und Schulen, entwickelt wurde.
Im Sprachaufbau lehnt sich BASIC-8Ø.3x an die Programmiersprache BASIC (Beginners All Purpose Symbolic Instructions Code) an, die durch ihre leichte Erlernbarkeit und Handhabung eine große internationale Verbreitung für Microrechner gefunden hat.
BASIC-8Ø.3x wurde speziell für den MC 8Ø.3x entwickelt und hat gegenüber dem Standard-BASIC alle Voraussetzungen zur Nutzung von Prozeßsteuerungen.
Es existieren Vereinbarungen, die es ermöglichen, Programme im Maschinencode des Microprozessors U88Ø einzubeziehen. Damit können innerhalb eines BASIC-Programmes Prozessdaten angefordert und ausgegeben werden. Mit BASIC-8Ø.3x kann somit softwareseitig der hardwaremässige Leistungsumfang des Gerätes erweitert werden. In Verbindung mit einem geeigneten Echtzeitbetriebssystems sind damit die meisten Probleme beherrschbar.
Das BASIC-8Ø.3x kann wie folgt aufgerufen werden:
Nach dem Laden von BASIC-Editor (4ØØØ-5FFF) und Interpreter (CØØØ-DFFF) wird mit "BNEW XXXX" ein BASIC-Freibereich angelegt, wobei "XXXX" die hexadezimale Länge (max.6ØØØH) ist.Die Festlegung des Prozedurnamens (BASIC-8Ø Unterprogramm) erfolgt durch die Anweisung

"BDEF Name"

wonach sofort mit

"BEDIT Name"

mit dem Erstellen des BASIC-Unterprogramms begonnen werden kann. Ein Löschen von BASIC-Programmen erfolgt durch die Anweisung.

"BDEL name"

Der Programmstart erfolgt durch "RUN Name". Nach dem Kommando "BNEW" steht ein allgemeiner BASIC-Bereich (BASIC-Hauptprogramm) ohne Name zur Verfügung, der mit "BEDIT" ohne Name aufgerufen und mit "RUN" gestartet werden kann. Stört dies, so ist ein Löschen mit "BDEL" (ohne Name) möglich.

1. Grundaufbau des BASIC-8Ø.3x

Der BASIC-Editor dient der Erstellung und Testung von BASIC-Programmen. Zum Editor gehört ein Übersetzer und ein Rückübersetzer für BASIC. Der Programmierer gibt Anweisungen als "Quellzeile" in den Rechner ein, die Quellzeile wird übersetzt und in einem Zwischencode abgelegt. Zwecks Anzeige wird der Zwischencode rückübersetzt und auf dem Bildschirm erscheint wieder die Quellzeile. Diese ist ggf. etwas modifiziert (Leerzeichen, Zahlendarstellung). Als "Programm" liegt also jeweils die Anweisungsfolge im Zwischencode vor. Wenn das Programm mit Hilfe des Editors korrigiert wird, erfolgt gleichzeitig eine Korrektur des Zwischencodes.
Der BASIC-Interpreter dient der Abarbeitung eines mit dem Editor erstellten BASIC-Programmes. Der Interpreter enthält alle Maschinenbefehlsfolgen, die zur Abarbeitung der Programme nötig sind. Der Zwischencode enthält alle Informationen zur Auswahl der Maschinenbefehlsfolgen und zur Verwendung von Datenadressen.
Es erfolgt keine Übersetzung des Programmes bis auf das Maschinenniveau, wie es bei einem Compiler der Fall wäre. Da die Auswahl der Maschinenbefehlsfolgen im Interpreter zusätzlich Rechnerzeit in Anspruch nimmt, ist ein interpretativ abgearbeitetes Programm langsamer als ein compiliertes Programm. Der Zeitfaktor liegt im Bereich 1,2 bei Programmen mit umfangreichen numerischen Berechnungen. Der Zwischencode ist jedoch kürzer als der Code eines compilierten Programmes, da die Informationen in komprimierter Form vorliegen. Die interpretative Programmabarbeitung bietet einige Vorteile für die Programmerstellung:

- Die Programmerstellung kann interaktiv, d.h. im Dialog mit dem Rechner geschehen.
- Ein Programm kann sofort in den Rechner eingegeben, dort auf einfache Weise korrigiert und in Teilen getestet werden.
- Sofortige Meldung und Korrektur von Syntaxfehlern bei der Eingabe.
- Möglichkeit der sofortigen Einarbeitung der Testergebnisse in den Programmentwurf.
- Wesentlich verbesserte Testmöglichkeiten durch variables Einfügen von Testanweisungen, Schrittbetrieb und Programmlaufkontrolle.

2. Sprachaufbau des BASIC-8Ø.3x

2.1. Zeichenvorrat

Die Grundsymbole für die Sprache BASIC-8Ø.3x sind aus dem ASCII-Zeichenvorrat (siehe Anlage) gewählt.

Um ein Programm mit BASIC-8Ø.3x aufbauen zu können, ist folgender Zeichenvorrat zu verwenden. Dieser enthält:

- Buchstaben

A,B,C,.....Z

Anmerkung: Kleinbuchstaben und Umlaute sind nur bei der Textverarbeitung, (nicht bei der Programmierung selbst) anwendbar

- Ziffern (Numerics)

Ø,1,2,3,....9

- Sonderzeichen

. , ; % & " ¤ { } [ ] usw.

2.2. Elementarmorpheme

Die Grundsymbole bilden Morpheme, denen eine Bedeutung (Semantik) zugeordnet ist.

2.2.1. Operatoren

- arithmetische Operatoren

+ Additionszeichen
- Subtraktionszeichen
* Multiplikationszeichen
/ Divisionszeichen

Anmerkung: In BASIC-8Ø.3x ist das Potenzieren nicht direkt möglich.

- Vergleichsoperatoren

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

- logische Operatoren

NEG Negation

NEG	
Ø	1
1	Ø

XOR Exklusiv "ODER"

	XOR	
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

AND logisches "UND"

	AND	
Ø	Ø	Ø
Ø	1	Ø
1	Ø	Ø
1	1	1

OR logisches "ODER"

	OR	
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	1

Es können alle Sprachelemente des BASIC-8Ø.3x mit diesem Zeichenvorrat aufgebaut werden, wobei die syntaktischen Regeln streng eingehalten werden müssen. Diese sind in den Syntaxdiagrammen im Abschnitt 1Ø genau beschrieben.

2.2.2. Konstanten

Wir unterscheiden im BASIC-8Ø.3x Zahlenkonstanten (Zahlenwerte) und Zeichenkettenkonstanten.
Die Zahlenkonstanten können aus dem (negativen) Vorzeichen, einer Folge von Ziffern, dem Dezimalpunkt, dem Zeichen E (für Exponent), dem Vorzeichen des Exponentens und der Ziffernfolge für den Exponenten bestehen.
Es muß aber bei den Zahlenkonstanten beachtet werden, daß das positive Vorzeichen nicht mit eingegeben wird.

Beispiel:
```
1.341
Ø.Ø7
-6542.1
2ØØØ
-1ØØE-6
```

Bei der Zeichenkettenkonstante wird die eigentliche Zeichenfolge in Hochkomma eingeschlossen.

Beispiel:
```
PRINT 'NAMEN'
PRINT 'PVC'
PRINT 'TU 144'
```

2.2.3. Variable und Felder

Variable und Felder müssen immer mit einem Buchstaben beginnen und können beliebig lang sein. Mehr als ein Leerschritt ist nicht erlaubt.

Beispiel:
```
PREIS
NUMMER 1
  Z4Ø
  COS PHI
```

Es werden vier Variablentypen unterschieden:

1. REAL-Variable
2. INTEGER-Variable
3. BOOLEAN-Variable
4. STRING-Variable

2.2.3.1. REAL-Variable

- Name für Gleitkommazahlen
- Zahlenbereich: Z =<1Ø hoch 38
- Verarbeitungsbreite: 7 Dezimalstellen
- Darstellung: intern, in einer Länge von 4 Byte

Bei der Darstellung von Brüchen muß beachtet werden, daß Rundungsfehler entstehen können.
REAL-Variablen unterscheiden sich in einfache und indizierte. Indizierte Variable sind Feldelemente. Mit ihnen können Felder und Matrizen verarbeitet werden.

Beispiel:
```
BALL (12) Element Nr. 12 des Feldes BALL
MAT (B)   Element Nr. B  des Feldes MAT
```

2.2.3.2. INTEGER-Variable

- Name für Festkommazahlen
- Zahlenbereich: -32768=>Z=<32767
- Darstellung: intern in einer Länge von 2 Byte im Zweierkomplement

Die INTEGER-Variablen werden vor der Berechnung in REAL-Variablen umgewandelt, da alle Berechnungen innerhalb von BASIC-8Ø.3x im Gleitkommaformat ausgeführt werden. Wie REAL-Variablen können auch INTEGER-Variablen indiziert werden. Ist eine INTEGER-Variable Ergebnis einer LET-Anweisung und der INTEGER-Zahlenbereich wird überschritten, so nimmt die INTEGER-Variable den größten bzw. kleinsten darstellbaren Wert an. Sie wird begrenzt. Eine Fehlerausschrift erfolgt dabei nicht.

2.2.3.3. BOOLEAN-Variable

BOOLEAN-Variable sind intern Einzelsbits mit dem Wertevorrat Ø und 1. Auf ein Speicherbyte passen 8 BOOLEAN-Variablen, da im Speicher nur ein Bit belegt wird. Es wird zuerst das niedrigste Bit (LSB-last significant bit) belegt. Eine Bildung von BOOLEAN-Feldern und Verwendung von BOOLEAN-Feldelementen ist ebenfalls möglich.

2.2.3.4. STRING-Variable

STRING-Variable sind Namen für Zeichenketten. Ihnen werden im Speicher ASCII-Zeichen zugeordnet.

Beispiel:

Die Zeichenkette VARIABLE habe die Bezeichnung TEXT1
Die Zeichenkette VORZEICHEN habe die Bezeichnung TEXT2

TEXT1 hat 8 Zeichen ——>TEXT1 (8)
TEXT2 hat 1Ø Zeichen——> TEXT2 (1Ø)

- die Einzelzeichen werden von Ø gezählt
- die Anzahl der Zeichen muß vereinbart werden
- restliche Zeichen werden automatisch mit Leerzeichen aufgefüllt

3. Standardfunktionen

In BASIC-8Ø.3x sind folgende Standardfunktionen vereinbart:

algebraische Darstellung	BASIC-Darstellung	Bedeutung
$\sqrt{x}$	SQR(x)	Quadratwurzel
ln(x)	LOG(x)	nat.Logarithmus
e^x	EXP(x)	e(=2.781) hoch x
sin(x)	SIN(x)	Sinus x
cos(x)	COS(x)	Cosinus x
|x|	ABS(x)	Absulutwert von x
-/+	SGN	Vorzeichen
	INT	Ganzzahlanteil
π	PI	3.14159

Alle diese Funktionen außer PI benötigen ein Argument x, das in Klammern anzugeben ist. In Bogenmaß sind die Argumente zu Funktion Cosinus und Sinus anzugeben.

Beispiel:

```
1Ø      LET A =PI*1ØØ
        LET B =SIN(3Ø PI/18Ø)
        LET C =10*COS(6Ø*PI/18Ø)
        LET D =LOG(A)
        LET E =-C*EXP(-PI/B)
        LET F =SQR(144)
        LET G =INT(A)
        LET H =ABS(E)
        LET I =SGN(E)*ABS(A/1ØØ)
        LET J =SGN(-1.234568)
2Ø      PRINT 'ERGEBNISSE FUNKTIONSBERECHNUNGEN'
        PRINT
3Ø      PRINT 'A=';A
        PRINT 'B=';B
        PRINT 'C=';C
        PRINT 'D=';D
        PRINT 'E=';E
4Ø      DPL 2,3Ø'F=';F
        DPL 3,3Ø'G=';G
        DPL 4,3Ø'H=';H
        DPL 5,3Ø'I=';I
        DPL 6,3Ø'J=';J
1ØØ     END
        RUN


        ERGEBNISSE FUNKTIONSBERECHNUNGEN

        A=314.159                F=12.ØØØØ
        B=5ØØ.ØØØE-Ø3            G=314.ØØØ
        C=5.ØØØØØ                H=9.33724E-Ø3
        D=5.74999Ø               I=-3.14159
        E=-9.337424E-Ø3          J=-1.ØØØØØ

OK
>_
```

4. Im BASIC-8Ø.3x verwendete Ausdrücke

Nach strengen syntaktischen Regeln können nunmehr mit den bisher definierten Elementen der Programmiersprache BASIC 8Ø.3x syntaktische Konstruktionen aufgebaut werden, die Ausdrücke genannt werden.
Dabei kann man immer wieder feststellen, daß in solchen Formeln ausgedrückte Zusammenhänge in verschiedenen Formen niedergeschrieben werden. Dies setzt eine strenge Formalisierung voraus.

Es sind für Ausdrücke Vorschriften für das Niederschreiben festgelegt, die unbedingt einzuhalten sind, damit der Interpreter jeden Ausdruck in die ihm äquivalenten Elementaroperationen zerlegen kann. Bei Verstoß gegen diese Regeln erfolgt durch den Computer eine Fehleranzeige. Ähnlich wie bei Konstanten und Variablen unterscheidet man:

- arithmetische Ausdrücke
- STRING-Ausdrücke
- logische Ausdrücke (BOOLEAN- bzw. VERGLEICHS-Ausdruck)

Wie in der Mathematik sind für die Operatoren Vorrangsregeln festgelegt, da in diesen Ausdrücken die einzelnen Elemente durch Operatoren verknüpft werden.
Bei der Abarbeitung gilt dabei folgende Reihenfolge:

- (),
- *,/
- +,-
- =,>=,<=, <>, >, <
- NEG
- XOR
- AND
- OR

4.1. arithmetische Ausdrücke

Arithmetische Ausdrücke sind aufgebaute Konstruktionen, die aus

- Zahlenkonstanten
- Zahlenvariablen
- Standardfunktionen
- arithmetischen Operatoren
- (runde)Klammern

bestehen können. Die aus der Mathematik bekannten Regeln haben Gültigkeit, wobei aufgrund des vorgegebenen Zeichenvorrats die Darstellung gewählt werden muß.

Beispielweise muß der Bruch

$$\frac{a+b}{a-b}$$

so dargestellt werden:(A+B)/(A-B)

d.h., die klammernde Wirkung des Bruchstriches in der mathematischen Darstellung muß durch Klammern erzeugt werden.
Aufgeführte Regeln sind unbedingt einzuhalten:

1. Der einfachste arithmetische Ausdruck ist ein Elementarausdruck. Er besteht aus einer Variablen oder einer vorzeichenlosen Zahl.
2. Ist der Nenner eines Bruches kein Elementarausdruck, so muß er in Klammern gesetzt werden.
3. Es muß immer das Multiplikationszeichen geschrieben werden.
4. Nach den mathematischen Regeln müssen Klammern immer paarweise gesetzt werden. Erlaubt sind nur runde Klammern.
5. Es dürfen nicht mehrere Operationen bzw. Operator und negatives Vorzeichen nebeneinanderstehen (durch Klammern trennen!)
6. Überflüssige Klammern schaden nicht, wenn sie mathematisch richtig sind.

Beispiel:

Mathemat. Schreibweise	BASIC 3Ø
$x(y-1)$	X*(Y-1)
$\frac{b}{-c}$	B/(-C)
$b\sqrt{c-d}$	B*SQR(C-D)
$a^2+2ab+b^2$	A*A+2*A*B+B*B
$\sqrt[5]{x-y}$	EXP(1/5*LOG(X-Y)

4.2. STRING-Ausdrücke

Sie bestehen aus:

- STRING-Konstanten
- STRING-Variablen
- Trennzeichen (',' oder ';')

Es lassen sich Texte verarbeiten bzw. Zeichenketten miteinander verknüpfen. Dabei werden die STRING-Konstanten in Hochkomma eingeschlossen.
Es kann sowohl auf die gesamte Zeichenkette, einen Teil der Zeichenkette oder auf eingegebenes Zeichen zugegriffen werden.
Dies soll anhand von Beispielen demonstriert werden.

Hinweis: Siehe Abschnitt 2.3.4. !

Beispiele:

Es sind folgende Zeichenkettenvariable definiert:

TEXT1 mit 'WART'
TEXT2 mit 'BURG'

dann kann mit der Zeichenkettenkonstante

- 'GRAF'

folgende Verknüpfung erzeugt werden:

Bsp.a: TEXT1;TEXT2 ⟶ WARTBURG
TEXT2;TEXT1 ⟶ BURGWART

Bsp.b: TEXT2,'GRAF' ⟶ BURGGRAF

Verknüpfung der Zeichenkette in der aufgeführten Reihenfolge:

- Trennzeichen ';': Aneinanderreihen der Zeichenketten ohne Zwischenraum
- Trennzeichen ',': Es werden Zwischenräume zwischen die Zeichenketten eingefügt.

Es werden die Zeichenkettenvariable und die Zeichenkettenkonstante wie in Beispiel a verknüpft.

Bsp.c: TEXT2(2,2);TEXT1(Ø,1) ⟶ RGW

Dabei werden aus dem TEXT2 nur die Zeichen RG und aus dem TEXT1 das Zeichen W verknüpft. Dies wird durch die Ziffern mit der runden Klammer nach TEXT2 und TEXT1 festgelegt. Allgemein bedeuten dabei in den Klammern:

(2,2)
- Länge der Zeichenkette 2 Zeichen
- Nr. des ersten Zeichens der Teilzeichenkette (Zählung von Ø)

4.3. BOOLEAN-Ausdrücke

Boolsche Ausdrücke können aus

- Boolschen Variablen
- Vergleichausdrücken
- arithmetischen Ausdrücken und logischen Operatoren

bestehen.

1. Der boolsche Wert eines arithmetischen Ausdrucks ist

   - Ø, wenn der Wert eines arithmetischen Ausdrucks gleich Ø ist,
   - 1, wenn der Wert des arithmetischen Ausdrucks ungleich Ø ist.

   Vergleichsausdrücke liefern den Wert Ø oder 1 in Abhängigkeit des Vergleiches zweier numerischer Ausdrücke oder eines Stringausdrucks mit einer Textkonstanten oder STRING-Variablen.

2. Die boolschen Operanden können über die logischen Operationen

   AND logische - UND - Verknüpfung
   OR logische - ODER - Verknüpfung
   XOR logische ANTIVALENZ-ODER-Verknüpfung

   zu einem boolschen Ergebnis verknüpft werden. Dabei ist XOR gegenüber AND und OR priorisiert.
   Die boolsche Funktion

   NEG logische Negation

   wird von einem boolschen Wert geschrieben und negiert ihn. NEG ist am höchsten priorisiert. Eckige Klammern ermöglichen die vorzugsweise Abarbeitung der geklammerten Ausdrücke und damit die Änderung der Abarbeitungspriorität.

   Im Beispiel

   A OR NEG B XOR C

   wird zuerst B negiert, dann mit C über XOR verknüpft und zuletzt mit A über OR verknüpft.

   Im Beispiel

   A OR NEG [B XOR C]

   wird zuerst B und C über XOR verknüpft, dann wird negiert und zuletzt mit A über OR verknüpft.

5. Aufbau eines BASIC-8Ø.3X Programms

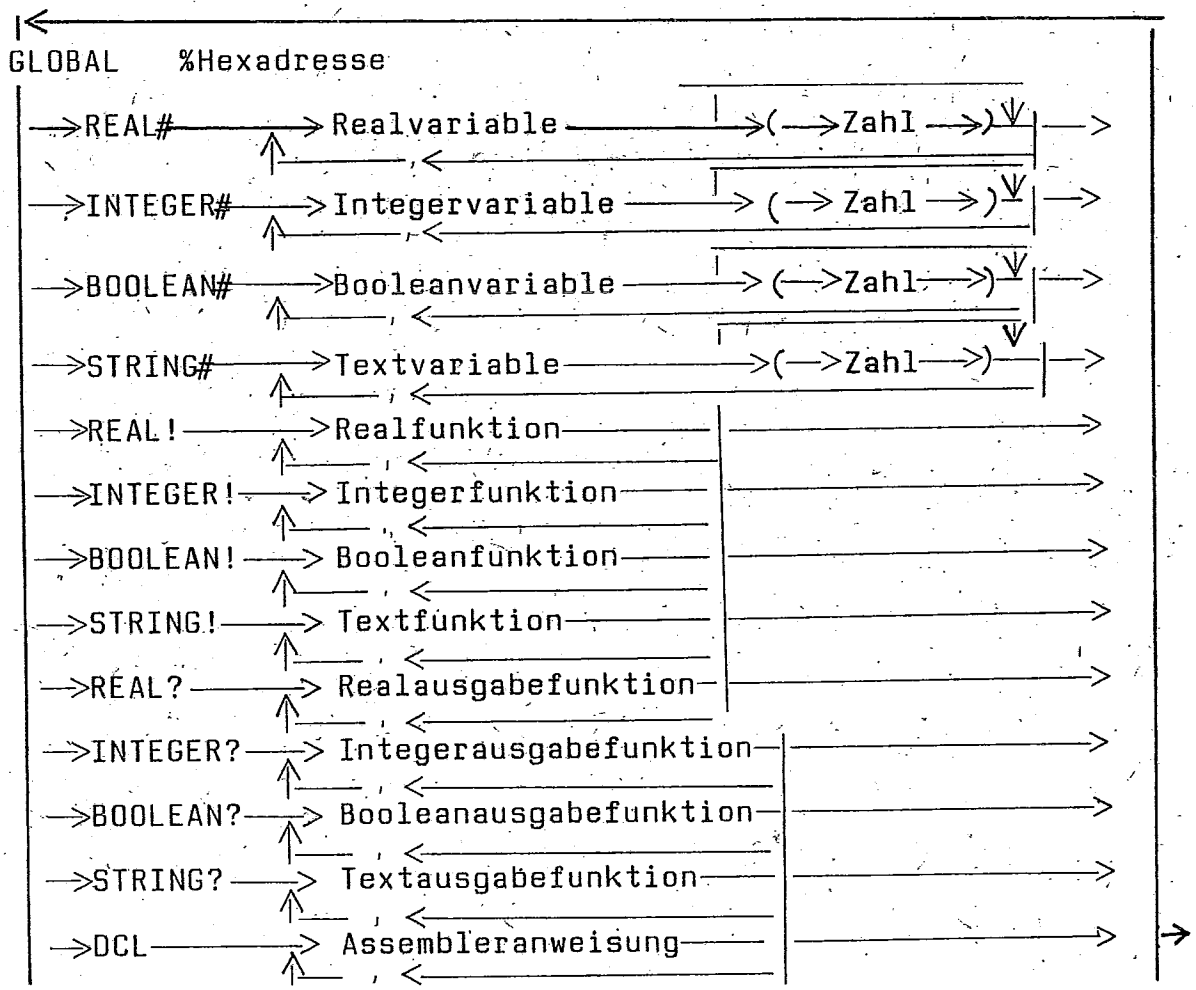
Das BASIC-8Ø.3X Programm besteht aus:

- Vereinbarungsteil
- Anweisungsteil
- Programmabschluß

5.1. Vereinbarungsteil

Die zur Abarbeitung des Programms benötigten Variablen, zusätzlichen Funktionen und Prozeduren werden im Vereinbarungsteil aufgeführt. Dadurch kann dem MC 8Ø die Möglichkeit gegeben werden, die benötigten Speicherplätze zu realisieren.
Es werden unterschieden:

- lokale Vereinbarungen
- globale Vereinbarungen



Hinweis: Siehe Abschnitte 2.2.3., 3,
5.1.2.1. und
5.1.2.2.

5.1.1. lokaler Vereinbarungsteil

Das Hauptprogramm und jede Prozedur beginnt mit der Anweisung

```
' DEF Name
```

Dabei ist der "Name" eine beliebige Zeichenfolge, die zuvor mit "BDEF" festgelegt wurde. Alle auf "DEF" folgenden Anweisungen gehören zum lokalen Vereinbarungsteil. Die dort getroffenen Vereinbarungen haben nur lokal im entsprechenden Programm Gültigkeit. Wird das BASIC-8Ø.3X aufgerufen, erscheint auf dem Bildschirm die Ausschrift:

```
DEF 'Name'
REAL A, B, C, ...Z
```

Es können dann weitere lokale Variable nach der Zeile REAL A,B,C, ...Z vereinbart werden.

Beispiel:
```
REAL A,B,C,...Z
REAL ZINS (51)
INTEGER ZAEHL1,ANZ
```

Anmerkung:
- Es dürfen keine Zeilennummern den Zeilen im Vereinbarungsteil vorangestellt werden!
- Die im Grundzustand geschaffenen lokalen Vereinbarungen können zu Beginn gestrichen werden.
- Es wird die Anzahl der Feldelemente vereinbart.
- Der Vereinbarungsteil darf nur erweitert werden. Streichen von Vereinbarungen oder Ändern von Feldgrößen führen dann zu Fehlern, wenn die danach folgenden Variablen im Programm schon benutzt worden sind.

5.1.2. globaler Vereinbarungsteil

Im globalen Vereinbarungsteil werden globale Variable, Assemblerfunktionen, Assemblerausgabefunktionen und Assembleranweisungen declariert. Der globale Vereinbarungsteil kann mit der Anweisung VEDIT auf dem Bildschirm sichtbar gemacht werden.
Ebenso wie lokale Variable werden auch die globalen Variablen vom Programm verarbeitet. Die Speicheradresse wird vom Anwender festgelegt. Damit kann der Datenzugriff auch über ein Assemblerprogramm erfolgen. Wenn die im globalen Vereinbarungsteil verzeichneten Variablen im Programm bereits benutzt werden, darf die Adresse der Vereinbarung (GLOBAL %...) nicht mehr verändert werden. Eine Änderung in den Vereinbarungsteilen ist generell mit einem Spezialprogrammpaket möglich, das ein BASIC-Programm in die Quelltextform bringt, eine Korrektur des Quelltextes erlaubt und den Quelltext im Block wieder in BASIC übersetzt.
Ein Block im globalen Vereinbarungsteil beginnt mit

Bsp. GLOBAL %8ØØØ

8ØØØ ist eine hexadezimale Speicheradresse, auf die sich alle weiteren Eintragungen beziehen.

Die Typenbezeichnungen der globalen Variablen erfolgen analog denen der lokalen Variablen mit dem nachgestellten #.

```
Beispiel: REAL# A1,VE(45)
          BOOLEAN# B1,B2,B3(6)
          STRING  TEXP (7Ø)
```

Die in diesem Block befindlichen Variablen werden ab der Speicheradresse 8ØØØ angelegt. Somit haben die Variablen folgende hexadezimale Adr.:

```
A1    8ØØØ          eine REAL-Variable a 4 Byte
VE    8Ø04          45 REAL-Feldelemente a 4 Byte
                    = 18Ø Byte = B4H
B1    8ØB8 BitØ     BOOLEAN-Variable a 1 Bit
B2    8ØB8 Bit1
B3    8ØB8 Bit2...7
TEXP  8ØB9...88FE   STRING-Variable mit 7Ø Zeichen
```

Ein nächster Vereinbarungsblock könnte z.B. beginnen

```
GLOBAL %8CØØ
INTEGER F1,F2
```

Damit werden die INTEGER-Variablen F1 und F2 auf den Adressen 8CØØ und 8CØ2 gespeichert.

5.1.2.1. Assemblerfunktionen

Eine Assemblerfunktion ist ein Unterprogramm in Assemblersprache (U88Ø), welche einen Wert als Ergebnis liefert. Sie werden ähnlich wie Standardfunktionen verwendet. Assemblerfunktionen können ohne Argument oder mit mehreren Argumenten auftreten. "XE" sei z.B. eine Assemblerfunktion ohne Argumente, die einen Analog-Digital-Wandler abfragt und einen Wert als Ausgang liefert. In einer Anweisung wird diese Assemblerfunktion folgendermaßen verwendet:

```
LET XW=W-XE
```

Zuvor muß die Assemblerfunktion wie folgt vereinbart werden:

```
GLOBAL %8ØØØ
INTEGER! XE
```

Ab Adresse 8ØØØ beginnt dieses Assemblerprogramm, das als Unterprogramm formuliert ist und ein Ergebnis in INTEGER-Format im Registerpaar "HL" liefert. Aus programmtechnischen Gründen ist es oftmals günstig, mit einer Sprungverteilertabelle zu arbeiten. Damit können auch mehrere Assemblerfunktionen in einem Block deklariert werden.

Beispiel:

```
GLOBAL %8EØØ
INTEGER! XMIN,XMAX,Soll
BOOLEAN! STEU1,STEU2
INTEGER! VEN
```

Ab der Adresse 8EØØ stehen dann die Sprünge zu den eigentlichen Programmen!

```
JMP MIN
JMP MAX
JMP SOLL
JMP STEU1
JMP STEU2
JMP VEN
```

5.1.2.2. Assemblerausgabefunktion

Eine Assemblerausgabefunktion ist ein Unterprogramm in Assemblersprache (U88Ø), die einen Wert als Eingang benötigt. "VEN" sei z. B. eine Assemberausgabefunktion, die ein Stellglied ansteuert. Die Assemblerfunktion wird in BASIC folgendermaßen verwendet:

```
LET VEN=I+K*(W-XE)
```

Das Ergebnis des arithmetischen Ausdrucks wird der Ausgabefunktion in den CPU-Register übergeben. Vereinbart werden Assemblerausgabefunktionen mit nachgestelltem "?".

```
GLOBAL %8EØF
INTEGER? VEN
```

Assemblerausgabefunktionen können sich nur auf der linken Seite einer LET-Anweisung befinden.

5.1.2.3. Assembleranweisungen

Eine Assembleranweisung ist ein Unterprogramm in Assemblersprache ohne zusätzliche Schnittstellen. In BASIC-8Ø.3X wird eine Assembleranweisung wie folgt verwendet:

Beispiel:

```
REGLER
```

vereinbart wird diese Assembleranweisung

```
GLOBAL %8E12
DCL REGLER
```

Wenn die Anweisung REGLER erreicht wird, wird im Assemblerprogramm auf der Adresse 8E12 gestartet.
Die Einbindung von Assemblerprogrammen in BASIC-8Ø.3X ermöglichen die Ausnutzung des gesamten Befehlssatzes des Prozessors U88Ø und damit eine spezifische hardwarenahe Programmanweisung.

5.2. Anweisungsteil

Im Standard BASIC beginnt jede Anweisungszeile mit einer Zeilennummer, die Anweisungen sind in steigender Reihenfolge der Zeilennummer sortiert. Bei BASIC-8Ø.3X muß nicht jede Zeile numeriert werden. Zeilennummer sind notwendig:

- bei der ersten Anweisung eines Programms
- nach einer Kommentaranweisung REM
- nach IF...THEN.. als Abschluß der Aktivität
- als Sprungziel (GOTO)

Eine numerierte Zeile mit den nachfolgenden Zeilen ohne Nummer entspricht in den anderen BASIC-Versionen einer komplexen Programmzeile mit mehreren Anweisungen.

5.2.1. Übersicht über alle Anweisungen, Kommandos und Funktionen im BASIC-8Ø.3X

Alle mit "*" gekennzeichneten Anweisungen, Funktionen und Kommandos sind als fester Bestandteil des BASIC-8Ø.3X im Editor vereinbart. Alle Anweisungen, Funktionen ohne "*" sind im globalen Vereinbarungsteil vermerkt und für den Anwender gewissermaßen nicht fest. Insbesondere gilt für sie:

- Bezeichnung kann vom Anwender im Bedarfsfall geändert werden, ohne daß dies Auswirkungen auf die Z-Codekompatibilität hat
-der globale Vereinbarungsteil kann erweitert oder gekürzt werden

Befehlssatz BASIC-8Ø.3X

	Befehle	Anweisungsgruppen
	ABS	Operation
*	AND	Operation
*	BEGIN LOOP	Laufanweisung
*	BOOLEAN	Vereinbarung
*	BOOLEAN!	Vereinbarung
*	BOOLEAN#	Vereinbarung
*	BOOLEAN?	Vereinbarung
	CHR	Operation
	CLEAR	Ausgabeanweisung
	COS	Operation
*	DCL	Vereinbarung

*	DEF	Vereinbarung
	DPL	Ausgabeanweisung
	END	Programmabschluß
	EXP	Operation
*	FN	Funktionen und Prozeduren
*	FN=	Funktionen und Prozeduren
*	FNEND	Funktionen und Prozeduren
*	FOR,TO,STEP,NEXT	Laufanweisung
*	FORMAT	Ausgabeanweisung
*	GLOBAL %	Vereinbarung
*	GOSUB, RETURN	Unterprogrammtechnik
*	GOTO	Sprunganweisung
*	IF, THEN	Programmverzweigung
*	IF,DO,DOEND	Programmverzweigung
*	IF,DO,ELSEDO	Programmverzweigung
*	IF,AND,DO,DOEND	Programmverzweigung
	IINPUT	Eingabeanweisung
	INPRDY	Eingabeanweisung
	INKEY	Eingabeanweisung
	INPUT	Eingabeanweisung
	INSTRING	Operation
	INT	Operation
*	INTEGER	Vereinbarung
*	INTEGER!	Vereinbarung
*	INTEGER#	Vereinbarung
*	INTEGER?	Vereinbarung
	LEN	Operation
*	LET	Wertzuweisung
	LOG	Operation
*	NEG	Operation
*	OR	Operation
	PRINT	Ausgabeanweisung
*	REAL	Vereinbarung
*	REAL!	Vereinbarung
*	REAL#	Vereinbarung
*	REAL?	Vereinbarung
*	REM	Kommentaranweisung
*	RUN	Anweisung
	SGN	Operation
	SIN	Operation
	SQR	Operation
*	STRING	Vereinbarung
*	STRING!	Vereinbarung
*	STRING#	Vereinbarung
*	STRING?	Vereinbarung
	STOP	Anweisung
*	SUBEND	Unterprogrammtechnik
	TAB	Operation
	TRACE	Testhilfe
	TRON	Testhilfe
	TROFF	Testhilfe
	WAIT	Anweisung
	WHILE,DO,LOOP	Laufanweisung
	XOR	Operation

Hinweis: Die Anweisungsgruppe "Vereinbarung" wird in 5.1. erläutert.

5.2.2. Erläuterung der Anweisungsgruppen

5.2.2.1. Operationen

ABS

Befehlsstruktur: ABS (Argument)

Diese numerische Funktion liefert den Absolutwert des Arguments. Negative Argumente werden mit -1 multipliziert, positive bleiben unverändert.

Beispiel: LET A=ABS(-1ØØ)⟶A=1ØØ

AND

Befehlsstruktur: A AND (Bedingung)

Dies ist ein boolscher Operator, zwischen zwei boolsche Operanden gestellt (siehe 4.3.) die UND-Verknüpfung realisiert. Als boolsche Operanden gelten dabei auch Terme mit dem Operator OR, laut Prioritätsregeln.

Beispiel:
```
LET A1=Ø
LET A2=1
LET A3=A1 AND A2 ⟶A3=Ø
          (A1-A3 BOOLEAN - Variablen)
```

CHR

Befehlsstruktur: CHR¤(n)

Es ist eine Stringfunktion, die ein Zeichen mit dem nachfolgend in () angegebenen Code (dezimal) erzeugt.

Beispiel: PRINT CHR¤(65)⟶'A' auf dem Bildschirm

COS

Befehlsstruktur: COS (Argument)

Die Funktion liefert den Cosinus des Winkels, der als Argument eingegeben wird. Dabei ist zu beachten, daß dieser Winkel in Bogenmaß (Winkel in Grad*PI/18Ø) angegeben wird.

Beispiel: LET A=COS(PI)⟶A=-1

EXP

Befehlsstruktur: EXP (Argument)

Die Funktion EXP berechnet den Wert der Funktion e hoch Argument. Sie dient als Umkehrfunktion des natürlichen Logarithmus.

Beispiel: LET A=EXP(1) ⟶ A=2,71828=e

INSTRING

Befehlsstruktur: INSTRING (text1=text2)

Es handelt sich um eine numerische Funktion, die als Ergebnis die Position des ersten Textausdruckes "text1" im zweiten Textausdruck "text2" liefert. Ist "text1" nicht in "text2" enthalten, so wird als Ergebnis der Wert Ø geliefert.
Das Zeichen "=" dient als Trennzeichen zwischen beiden Textausdrücken. Das in anderen BASIC-Varianten dafür benutzte Zeichen "," kann hier nicht verwendet werden, da es vom Interpreter als zum Textausdruck selbst zugehörig erkannt wird.

Beispiel: PRINT INSTR ('A'='XYZABC')

Der Wert 4 wird ausgegeben, da das Zeichen "A" an vierter Position der Kette "XYZABC" steht.

INT

Befehlsstruktur: INT (Argument)

Die Funktion INT erzeugt den größten ganzzahligen Wert, der kleiner oder gleich dem Argument ist.

Beispiel: LET A=INT(4,99)⟶A=4

LOG

Befehlsstruktur: LOG (Argument)

LOG berechnet den natürlichen Logarithmus des eingegebenen Arguments.

Beispiel: LET A=LOG(1ØØ)/(LOG(1Ø))⟶A=2

LEN

Befehlsstruktur: LEN (text)

Es handelt sich um eine numerische Funktion, die als Wert die Länge des Textausdruckes liefert, der als Argument angegeben wurde. Damit sind die Längen von Variablen Zeichenketten ermittelbar.

Beispiel:

```
10      LET A=LEN('XYZ')
        LET B=LEN('ABC')
        PRINT A,B
        END
```

Es wird für A und B eine "3" ausgedruckt. Beide Zeichenketten haben die Länge von drei Zeichen.

NEG

Befehlsstruktur: LET A=NEG B

Diese Funktion ist eine logische Funktion, die von der nachfolgenden boolschen Variable das Komplement bildet.

Beispiel:
```
LET A1=Ø
LET A2=NEG A1 ⟶ A2=1
```

OR

Befehlsstruktur: LET A=B OR C

Durch die Funktion OR werden zwei boolsche Variablen durch ein logisches ODER verknüpft.

Beispiel:
```
LET A1=Ø
LET A2=1
LET A3=A1 OR A2 ⟶ A1=1
```

SGN

Befehlsstruktur: SGN (Argument)

SGN liefert eine Information über das Vorzeichen des Arguments. Folgendes gilt:

```
(Argument) > Ø....SGN(Arg.)=1
(    "   ) < Ø....SGN(Arg.)=-1
(    "   ) = Ø....SGN(Arg.)=1
```

Beispiel: LET B=SGN(-15) ⟶ B=-1
LET B=SGN(Ø) ⟶ B=1

SIN

Befehlsstruktur: SIN (Argument)

Die Funktion liefert den Sinus des Winkels, der als Argument eingegeben wird. Es ist zu beachten, daß dieser Winkel in Bogenmaß (Winkel in Grad*PI/18Ø) angegeben wird.

Beispiel: LET A=SIN(PI) ⟶ A=Ø

SQR

Befehlsstruktur: SQR (Argument)

Die Funktion SQR berechnet die Quadratwurzel des eingegebenen Arguments. Negative Argumente werden auch negativ.

Beispiel: LET A=SQR(2) ⟶ A=1,41421

TAB

Befehlsstruktur: PRINT TAB (Argument)

Von der aktuellen Bildschirmposition werden die von den angegebenen Argumenten angegebene Anzahl von Leerzeichen ausgeführt.

Beispiel: PRINT '*';TAB(1Ø);'*'
Druckbild * *

XOR

Befehlsstruktur: LET A=B XOR C

Durch die Funktion XOR werden zwei boolsche Variablen durch ein Exklusiv-ODER verknüpft (Antivalenz).

Beispiel: LET A1=1
LET A2=Ø
LET A3=A1 XOR A2 ⟶ A3=1

5.2.2.2. Wertzuweisung

LET

Befehlsstruktur: LET Variable=arithm. Ausdruck

Durch die Standardvereinbarung werden die REAL-Variablen A,B, ...Z reserviert. Mit der Anweisung LET können diese Variablen Werte in Form eines Ausdrucks, einer anderen Variablen oder einer Konstanten desselben Typs zugewiesen werden.

Beispiel:
```
LET A=Ø
LET R=X
LET S=Ø:J=Ø
```

Der linke Teil der LET-Anweisung wird vom rechten Teil durch das Ergibtzeichen (=) getrennt. Der rechts stehende Ausdruck wird berechnet und das Ergebnis der links stehenden Variablen zugewiesen. Mit Hilfe des Trennzeichens ":" ist es möglich, mehrere Zuweisungen in einer Zeile zu schreiben. Dafür entfällt das Schlüsselwort LET. Hinter dem Schlüsselwort DO kann auch ohne LET eine Wertzuweisung notiert werden. In dieser Anweisung ist das Zeichen "=" ein Zuweisungsoperator.
Im Unterschied dazu wird bei Bedingungsabfragen das Zeichen "=" als Vergleichsoperator gedeutet. Sind im Vereinbarungsteil Felder deklariert, dann können auf beiden Seiten der LET-Anweisung auch indizierte Variablen stehen. Die Zuweisung erfolgt im Programmablauf und ist nach außen hin nicht sichtbar. Die Variablenwerte kann man erst nach einer Ausgabeanweisung (PRINT DPL) auf dem Bildschirm zur Anzeige bringen.

5.2.2.3. Ausgabeanweisung

CLEAR

Befehlsstruktur: CLEAR

Dieser Befehl löscht alle alphanumerischen und Sonderzeichen vom Bildschirm.

DPL

Befehlsstruktur:
```
DPL a,b 'Zeichenkette'
    ||
    |└── Nr. der Zeile
    └─── Nr. der Spalte
```

Mit DPL kann eine Bildschirmausschrift hinsichtlich ihrer Stellung auf dem Bildschirn in Zeile und Spalte positioniert werden. a kann den Wert zwischen Ø und 25, b zwischen Ø und 72 annehmen. Ansonsten entspricht die Anweisung DPL der PRINT-Anweisung.

PRINT

Befehlsstruktur:
```
PRINT
PRINT 'Zeichenkette'
PRINT Parameter
```

Die PRINT-Anweisung wird zur Ausgabe von Zahlenwerten, Werten von Variablen bzw. mathematischen Formen oder beliebigen Begleittexten verwendet. Nach PRINT mit den entsprechenden Trennzeichen werden die Elemente der Druckliste aufgelistet.
Dabei gilt folgendes:

- leere Druckliste bewirkt neue Zeile
- Trennzeichen Semikolon verhindert eine Zeilenschaltung
- mehrere Kommata möglich
- auszudrückende Zeichenketten werden in Hochkommata eingeschlossen

In der untersten Zeile wird die erste PRINT-Zeile geschrieben. Die nachfolgende Zeile schiebt den gesamten Displayinhalt eine Zeile weiter nach oben. Deshalb wird durch PRINT allein eine Leerzeile erzeugt.

Beispiel:

```
5       PRINT
1Ø      LET A=12
        LET B=5
        LET C=A*B
2Ø      PRINT 'A=';A,,,,,,,,;'B=';B
        PRINT 'C='C
        PRINT 'C=';A*B
3Ø      DPL 5,4;C*B
        DPL 6,4;C*A
        DPL 5,2Ø'ERGEBNIS = ';C
```

Es wird folgende Displayausschrift erzeugt:

```
A=12,ØØØØ          B=5,ØØØØØ
C=6Ø,ØØØØ
C=6Ø,ØØØØ

3ØØ.ØØØ            ERGEBNIS = 6Ø.ØØØØ
72Ø.ØØØ
```

Wahl des Ausgabeformates

Mit der Formatangabe, die Bestandteil des Textausdruckes ist, kann die Anzahl der Stellenzahl vor und nach dem Komma festgelegt werden. Dabei ist analog der internen Stellenzahl die Ausgabe von maximal 7 Ziffern für die Mantisse möglich..
Die allgemeine Schreibweise lautet:

```
FORMAT !a.b!
        ├──── mit !: tabellengerecht
        └──── ohne!: textgerecht
```

FORMAT !a.b!

- Anzahl der Nachkommastellen (nk): Ø <=nk<=7
- Anzahl der Vorkommastellen (vk): 1<=vk<=7ist vk=Ø, dann Ingenieurformat
- mit !: ohne Exponent, mit Überlaufanzeige
- ohne !: der Exponent wird eingeplant aber nur bei Notwendigkeit ausgegeben

- Ingenieurformat (vk=Ø)

Zahlen kleiner 1 und größer 999.9 werden mit einem Exponenten in Dreierschritten dargestellt. Die Nachkommastellenzahl entspricht der Gesamtziffernzahl für die Mantisse, sie muß grösser oder gleich 3 sein. So könnten Ergebnisse folgender Form ausgegeben werden:

1.5
123
12.45E+Ø3
1.267E-12

- kein Ingenieurformat (vk=1...7)

Ist die anzugebende Zahl zu groß, dann gibt es 2 Möglichkeiten: Wird der Exponent eingeplant (ohne !), dann erfolgt die Ausgabe der angegebenen Stellen vk als Mantisse mit einem anschließenden Exponenten als Korrekturfaktor.
Wird kein Exponent eingeplant (mit !), dann erfolgt die Ausgabe der angegebenen Stellen vk als Mantisse, wobei auf die letzte Stelle das Überlaufzeichen " " kommt. Ist die auszugebende Zahl kleiner als die mit vk darstellbare Zahl, dann werden Leerzeichen ausgegeben.

- Tabellengerecht

Hier erfolgt die spaltenweise Ausgabe stets so, daß unabhängig von der Größe der Zahlen die gleiche Anzahl Druckpositionen belegt werden. Mit diesem Format ist eine übersichtliche Tabellendarstellung möglich. Dabei gibt es folgende Festlegungen zu beachten:

. Das Vorzeichen "+" wird als Leerzeichen ausgegeben.
. Nachfolgende Nullen werden ausgegeben.
. Wird kein Ingenieurformat gefordert, steht der Dezimalpunkt an der gleichen Stelle.
. Nach der Mantisse werden 4 Stellen für den Exponent eingeplant und ein zusätzliches Leerzeichen ausgegeben, wenn das vordere "!" nicht steht.

- Textgerecht

Bei diesem Format werden nur die unbedingt notwendigen Stellen ausgegeben. Es entfallen deshalb:

. Vorzeichenstelle "+"
. Stellen für führende und nachfolgende Nullen nach dem Komma
. Stellen für den Exponenten, wenn dieser eingeplant, aber nicht notwendig ist

- Überlaufanzeige - ohne Exponent

Überschreitet die Zahl den möglichen Ausgabebereich, dann wird auf die letzte Druckposition das Überlaufzeichen " " ausgegeben. Sie ist beim Ingenieurformat und teilweise bei textgerechter Darstellung wenig sinnvoll.

- Standardformat

Das Standardformat entspricht der Angabe Format Ø.6!
Die Formatangabe bleibt solange erhalten, bis sie durch eine andere geändert wird.
Ist die Nachkommastellenzahl der auszugebenden Zahl größer als die in der Formatangabe gewünschte, wird auf die vorgegeb ene Stellenzahl gerundet.

Beispiel:

```
1Ø      LET A=987.6542
        LET B=PI
        LET C=12.3456
2Ø      PRINT A;B;C
        PRINT FORMAT 3.2 A;B;C
        PRINT FORMAT 3.2 A,,,B,,,C
        PRINT FORMAT 3.2! A;B;C
        PRINT FORMAT !3.2! A;B;C
        PRINT FORMAT 5.Ø A;B;C
        PRINT FORMAT Ø.3 A;B;C
        PRINT FORMAT Ø.6 A;B;C
1ØØ     END
```

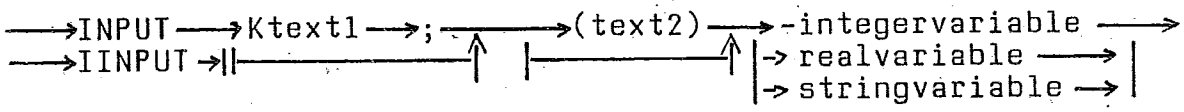
Displayausschrift

```
 987.654            3.14159          12.3456
 987.65    3.14     12.35
987.65                        3.14             12.35
 987.65                 3.14          12.35
 987.65    3.14        12.35
988        3      12
988        3.14   12.3
987.654                       3.14159          12.3456
```

5.2.2.4. Eingabeanweisung

INPUT und IINPUT

Syntax:



Mit INPUT können einer Variablen eines laufenden Programms über die Tastatur Texte (bei STRING-Variablen) oder Zahlenwerte zugeordnet werden.
IINPUT dient zur Eingabe im Interruptmodus.

Besonderheit:

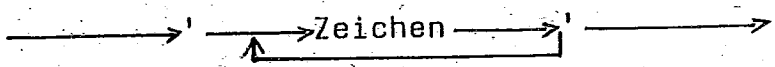
Das Programm wird sofort fortgesetzt, ohne auf die Bedienung zu warten. Die Bedienung der Tastatur/Display geschieht interruptgesteuert. Die Variable gilt als nicht belegt, auf sie darf nicht unmittelbar zugegriffen werden.
Die Variable wird im Interrupt belegt, danach wird eine Fertigmeldung in der bool. Variable INPRDY niedergelegt (INPUT-Ready), INPRDY=Ø bedeutet:

Eingabe läuft, auf Variable nicht zugreifen!

INPRDY=1: Eingabe beendet

Vorteil: Das BASIC-Programm kann neben der Eingabeanforderung noch weitere Aufgaben bearbeiten, z. B. Anzeigefunktionen wahrnehmen, on-line-Steuerung. Die Fertigstellung der Eingabe kann programmtechnisch an beliebiger Stelle abgefragt werden.

ktext1: Ein Aufforderungstext wird ausgegeben. Der Aufforderungstext verschiebt die Eingabeposition, Eingabefeld nach rechts.
ktext1 ist eine Textkonstante und in ' ' einzuschließen.



text2: Ein Vorzugstext wird auf das Display und in den Eingabetextpuffer eingeschrieben. Dieser Text gilt als eingegeben. Gleichzeitig wird mit Angabe des Vorzugstextes der Schreibbereich auf dem Display und die Länge des Textpuffers auf die Zeichenzahl des Vorzugstextes begrenzt. Wird kein Vorzugstext angegeben, so gilt der Textpuffer als leer. Das Display wird initial nicht beschrieben. Der Eingabebereich ist auf die Zeile begrenzt.

Vorteil bzw. Notwendigkeit:

a) Bei Menuebedienungen können Vorzugsdaten angegeben werden, die der Bediener im Standardfall nur zu quittieren braucht (ENTER), er kann aber auch andere Daten angeben.
Günstig für die Programmgestaltung ist die Integrierung dieses Vorzugstextes in die INPUT-Anweisung. Im BASIC-8Ø.3X

existiert für einen vergleichbaren Fall eine Anweisung "EDIT textvariable", die die Korrektur der Textvariablen ermöglicht.
Die INPUT-Anweisung des BASIC-80 vereint beide Möglichkeiten INPUT und EDIT, so daß auch eine kompatible Programmgestaltung zum BASIC-Programm möglich ist.

b) Die Eingrenzung des Schreibbereiches ist ebenfalls für die Menuebedienung nötig, um zu verhindern, daß bei falscher Bedienung andere Bereiche des Display falsch beschrieben werden.
Zur Eingabe sind folgende Tasten wirksam:

- alle zeichenerzeugenden Tasten
- ← → |← ←| Kursorbewegungen links, rechts und zum Zeilenanfang
- DEL INS Streichen und Einfügen von Einzelzeichen
- ENTER Übernahmetaste

Nach Betätigung der ENTER-Taste wird der eingegebene und ggf. korrigierte Text entweder in die angegebene Stringvariable übernommen oder in einen numerischen Wert gewandelt und in die numerische Variable übernommen.

Beispiele zu INPUT:

1)
```
INPUT A
```

Eingabe für eine einfache numerische Variable ist bei allen BASIC-Varianten als Standard vorgesehen. Der Kursor blinkt auf der zuletzt von PRINT bestimmten Position, auf der Kursorposition steht ein "?", das bei der Eingabe des ersten Zeichens übertippt wird. Die ganze Zeile ab der Kursoranfangsposition kann beschrieben werden. Ausgewertet werden die Zeichen, die eine Zahl darstellen einschließlich führender Leerzeichen.

2)
```
INPUT 'A=';A
```

Zusätzlich zu INPUT A wird ein Anforderungstext auf dem Display ausgeschrieben. Die Anweisung ist identisch mit

```
PRINT 'A=' ;
INPUT A
```

verschiebt also auch die Anfangsposition des Kursors für die Eingabe nach rechts.

3)
```
INPUT 'A=';('.....')A
```

Als Vorzugstext werden die 5 Punkte ausgeschrieben, die damit die Eingabeposition markieren. Es sind nur 5 Zeichen eingebbar.

4)
```
INPUT 'A=';('  2.Ø')A
```

Als Vorzugstext wird der an konstante Text "2.Ø" ausgeschrieben, das heißt für den Anwender: Ein Vorzugswert für A wird angeboten, der ggf. nur bestätigt werden braucht. Der Kursor blinkt unter der Ziffer 2, die davorliegenden Leerzeichen gehören mit zum Schreibbereich.

5) INPUT 'A=';(FORMAT !2.1!;A)A

Als Vorzugstext wird der alte Wert, der in A steht, angeboten. Die Formatwahl der Textkonvertierung gestattet auch hier eine gewünschte Eingrenzung des Schreibbereiches. Enthält A den Wert 2, so erscheint die gleiche Displayausschrift wie bei viertens. Die Zahl wird mit führenden Leerzeichen statt dem Vorzeichen "+" oder Vornullen dargestellt, der Kursor blinkt immer unter dem ersten Zeichen nach den führenden Leerzeichen. Die davorliegenden Positionen sind beschreibbar, um z. B. größere Zahlen eingeben zu können.

Beispiel:

```
IINPUT 'A=';('  2.Ø')A
  .
  .
WHILE NEG INPRDY DO
  .
  .
LOOP
LET B=A+...
```

Das Beispiel zeigt die Anwendung der IINPUT-Anweisung. Die Eingabe wird mit IINPUT angestoßen, d. h. der Anforderungs- und Vorzugstext erscheint, die Tastatur ist bedienbar; das BASIC-Programm wird jedoch fortgesetzt. Im Beispiel wird eine Schleife abgearbeitet, solange "NEG INPRDY".
Erst danach wird der Wert der Variablen A weiter verwendet.

INKEY¤

Befehlsstruktur: LET A=INKEY¤

Es handelt sich um eine Stringfunktion, die bei Abarbeitung das Zeichen, das der aktuell betätigten Taste auf der Standardtastatur zugeordnet ist, erzeugt. Damit können beispielsweise Programmverzweigungen aufgrund von Tastendrücken programmiert werden, die zur Realisierung einer Bedienerkommunikation, beispielsweise zum Ausfüllen von Menuebildern, notwendig sind.
Es werden auch Steuertasten (z. B. Kursorbewegung) erfaßt.

Beispiel:

```
STRING T¤
LET T¤=INKEY¤
```

Der Textvariablen T¤, die nur ein Zeichen umfaßt, wird die aktuelle Taste zugeordnet.
Im Programm wird gewartet, bis eine schreibzeichenerzeugende Taste (keine Steuertaste) betätigt wird: "Solange das Tastenzeichen kleiner als das Leerzeichen ist, wartet das Programm einer Schleife!"

Ansonsten wird das Zeichen ausgeschrieben. Das folgende Beispiel gibt diesen Sachverhalt vollständig wieder:

```
        STRING T¤
1Ø      BEGIN LOOP
        LET T¤=INKEY¤
        IF T¤<' ' DO
        LOOP
        PRINT T¤
        GOTO 1Ø
```

Die INKEY¤-Funktion liefert die aktuell betätigte Taste. Ist keine Taste betätigt, wird ein Steuerzeichen NUL geliefert. Dies wird auch mit der Textfunktion CHR¤(Ø) erzeugt. (ASCII-Code Ø).

Im folgenden Beispiel werden bei Betätigen der Kursortasten die darauf folgenden Programmabschnitte abgearbeitet.

```
        STRING T¤
Ø       BEGIN LOOP          ;große Schleife für Kursorauswer-
                             tung
        BEGIN LOOP          ;Schleife: Warten auf Tastatur
        LET T¤=INKEY¤       ;Zuordnen Tasten
        IF CHR¤(Ø)=T¤ DO
        LOOP                ;Warten bis eine Taste betätigt
                             ist
2Ø      IF CHR¤(17)  T¤ DO  ;Schleifenaustritt bei Taste ENTER
25      IF CHR¤(8)=T¤ DO    ;Auswerten Taste:
                             Kursor nach links?
          .
          .
```

Anweisungen werden ausgeführt, wenn
"← " betätigt wird, z. B.

```
        PRINT 'KURSOR NACH LINKS'
          .
          .
        ELSEDO
              IF CHR¤(12)=T¤ DO ;Kursor nach rechts?
          .
          .
```

Anweisungen bei "→", z. B.

```
        PRINT 'KURSOR NACH RECHTS'
          .
          .
        ELSEDO
27      IF CHR¤(1Ø)=T¤ DO
          .
          .
```

Anweisungen bei "↓", z. B.

```
        PRINT 'KURSOR NACH UNTEN'
        ELSEDO
28      IF CHR¤(11)=T¤ DO
          .
          .
```

Anweisungen bei "↑", z. B.

```
        PRINT 'KURSOR NACH OBEN'
```

```
29      DOEND       ;Schließen aller IF-Klammern
        DOEND
        DOEND
        DOEND
3Ø      LOOP        ;Schließen der Schleife für Kursor
          .
          .
31      PRINT 'ENTER'
        GOTO 1Ø
```

5.2.2.5. Kommando RUN

RUN

Befehlsstruktur: RUN

Mit dieser Funktion wird ein BASIC-Programm gestartet.

5.2.2.6. Anweisung STOP

STOP

Befehlsstruktur: STOP

Nach Abarbeitung der Anweisung "STOP" folgt eine Annahme des Betriebssystems-Kommandoabfragezustandes. Der Abarbeitungszustand des Programms bleibt jedoch erhalten, so daß dieses fortgesetzt werden kann.

5.2.2.7. Kommentaranweisung REM

REM

Befehlsstruktur: REM Zeichenkette

Mit "REM" können Kommentare im Klartext in das Programm eingefügt werden. Bei der Abarbeitung des Programmes werden diese Programmzeilen jedoch übersprungen.

5.2.2.8. Die Anweisung WAIT

WAIT

Befehlsstruktur: WAIT

Mit der Anweisung "WAIT" kann der Programmlauf unterbrochen werden. Das Festsetzen ist durch Drücken einer beliebigen Taste (außer Funktions- und Sondertasten) möglich. Diese Anweisung kann benutzt werden, um eine schnelle Folge von Ausgaben auf

dem Bildschirm sichtbar machen.

5.2.2.9. Laufanweisungen

BEGIN LOOP

```
Befehlsstruktur:      BEGIN LOOP
                      Anweisung(en) A
                IF Bedingung DO
                Anweisung(en) B
                LOOP
```

Es werden bei dieser Programmschleife mindestens die Anweisungen A realisiert. Ist die Bedingung erfüllt, wird die nach LOOP folgende Anweisung abgearbeitet. Wird die Bedingung nicht erfüllt, werden die Anweisungen B abgearbeitet und es erfolgt ein Rücksprung zur Zeile BEGIN LOOP, bis die Bedingung erfüllt ist. Dabei muß aber gewährleistet sein, daß sich die Bedingung während des Schleifendurchlaufs ändert, weil sonst die Schleife nicht verlassen wird. Im Grundfall können die Anweisungen A und B auch ganz wegfallen. Es wird am Ende der Schleife mit den Anweisungen A die Aushilfsbedingung abgefragt, aber nur wenn die Anweisungen B wegfallen.

Beispiel:

```
        BEGIN LOOP
1Ø      INPUT A
2Ø      IF A<>125 DO
        LOOP
        END
```

WHILE,DO,LOOP

```
Befehlsstruktur:        WHILE  Bedingung DO
                Anweisung(en)
                LOOP
```

Bei einer "WHILE,LOOP"-Schleife kann in Abhängigkeit von einer Bedingung beliebig oft durchlaufen werden. Es muß gesichert sein, daß sich die Bedingung während des Schleifendurchlaufes ändert, da sonst der Schleifendurchlauf nicht abgebrochen wird. Wird diese Bedingung erfüllt, kann das Programm mit der nach "LOOP" folgenden Anweisung weiter bearbeitet werden. Alle Schleifen können geschachtelt werden.

Beispiele:

```
        PRINT FORMAT 2.Ø
        LET B=3Ø
        DPL 14,Ø 'AUSSENSCHLEIFE'
        DPL 16,Ø 'INNENSCHLEIFE'
1Ø      WHILE B>Ø DO
        DPL 14,2Ø 'ZAEHLER1=';B
```

```
        LET B=B-1
        LET A=1Ø
        WHILE A>Ø DO
        DPL 16,2Ø 'ZAEHLER2=';A
        LET A=A-1
        LOOP
        LOOP
        END
```

FOR,NEXT

Befehlsstruktur:

```
FOR Variable=Parameter A TO Para-
meter E
STEP Parameter S
        NEXT Variable
```

Bei dieser Befehlsstruktur wird die sogenannte FOR-NEXT-Schleife realisiert. Die Variable (Laufvariante) besitzt einen durch den Parameter A (Anfangswert) und durch Parameter E (Endwert) definierten Wertebereich. Als Zähler dient die Laufvariable, die zu Beginn auf den Anfangswert gesetzt wird. Die Anweisungen nach FOR bis zur NEXT Anweisung (Schleifenrumpf) werden ausgeführt. Danach addiert NEXT den Parameter S (Schrittweite) zur Laufvariablen und prüft, ob sie größer als der Endwert ist. Ist das nicht der Fall, wird das Programm von Anfang des Schleifrumpfes fortgesetzt. Ist der Endwert erreicht, so wird das Programm mit der Anweisung nach NEXT fortgesetzt, wobei die Schleife damit verlassen wird.
A, E und S können positiv als auch negativ sein. STEP kann ohne Fehlerausschrift weggelassen werden. Dann wird die Schrittweite =1 als Standard festgelegt.

Beispiel:

```
        PRINT FORMAT 3.2
1Ø      INPUT A
        FOR I=Ø TO 52Ø STEP A
        PRINT 'ZAEHLER=';I
        NEXT I
2Ø      PRINT 'STEP='A
        PRINT 'SCHLEIFE'I
        WAIT
        GOTO 1Ø
        END
```

5.2.2.1Ø. Sprunganweisung

GOTO

Befehlsstruktur: GOTO Zeilennummer

Mit GOTO wird ein unbedingter Sprung (Sprung ohne Bedingung) erreicht. Es erfolgt in der Programmzeile GOTO ein Aussprung aus dem Programm und ein Einsprung in die durch GOTO angewie-

sene Programmzeile. Es wird empfohlen, Verzweigungen mit GOTO nur dort zu realisieren, wo es wirklich sinnvoll ist. Denn wird GOTO in einem komplexen Programm oft verwendet, leidet darunter sehr die Übersichtlichkeit des Programmes.

5.2.2.11. Programmverzweigungen

IF, THEN

Befehlsstruktur: IF Bedingung THEN
Anweisung
Zeilennummer Anweisung

Durch IF-Anweisungen werden die Abfrage von Bedingungen, unter denen eine oder mehrere Anweisungen abgearbeitet werden oder sich Programme verzweigen, ermöglicht. Eine Bedingung ist bei IF-Anweisungen immer eine zweiwertige Entscheidung (Ja oder Nein). Sie wird als Vergleichsausdruck

Parameter Vergleichsoperator Parameter

dargestellt. Folgende Vergleichsoperatoren sind zugelassen:

>größer	<kleiner	= gleich
>=größer gleich	<=kleiner gleich	<> ungleich

Es können aber auch boolsche Ausdrücke als Bedingungen verwendet werden.
Mit IF,THEN kann eine Bedingung abgefragt und eine Anweisung in Abhängigkeit von dieser Bedingung ausgeführt werden. Es muß dabei beachtet werden, daß die nachfolgende Zeile eine Zeilennummer besitzt. Nachfolgende Zeilen ohne Zeilennummer werden als zur IF-Zeile zugehörig anerkannt und nur im Ja-Fall abgearbeitet.

IF,DO,DOEND

Befehlsstruktur: IF Bedingung DO
Anweisung
DOEND

Die Abarbeitung einer Vielzahl von Anweisungen, die an eine Bedingung geknüpft sind, können mit dieser Befehlsstruktur ausgeführt werden. Die Anweisungsanzahl ist nicht eingeschränkt. Wichtig ist der Abschluß der bedingten Anweisung mit DOEND. Ist dieser Teil der Befehlsstruktur nicht vorhanden, erfolgt zwar eine Fehlerausschrift, die Fehlerstelle wird aber nicht erkannt. Zu realisieren sind mit IF,DO bedingte Abarbeitungen und Programmverzweigungen.

Beispiel:

```
1Ø      INPUT X
2Ø      IF X=33 DO
```

```
        PRINT '     JA'
        DOEND
        GOTO 1Ø
1ØØ     END
```

Beispiel:

```
1Ø      CLEAR
2Ø      INPUT X
        PRINT
        INPUT Y
        PRINT
        INPUT Z
        PRINT
3Ø      IF X=5 AND Y=22 AND Z=35Ø DO
        PRINT
        PRINT 'ZAHLENCODE ERKANNT'
        GOTO 1ØØ
        DOEND
4Ø      GOTO 1Ø
1ØØ     END
```

IF,DO,ELSEDO

Befehlsstruktur: IF Bedingung DO
Anweisung(en) A
ELSEDO
Anweisung(en) B
DOEND

IF,DO,ELSEDO ist eine Erweiterung der "IF,DO"-Anweisung. Es werden nur bei erfüllter Bedingung die Anweisung(en) A abgearbeitet. Danach springt das Programm zu der nach DOEND folgenden Anweisung. Ist die Bedingung nicht erfüllt, werden die Anweisung(en) B abgearbeitet.

Beispiel:

```
1Ø      INPUT A
2Ø      IF A=15 DO
        PRINT '  JA'
        ELSEDO
        PRINT '  NEIN'
        DOEND
3Ø      GOTO 1Ø
1ØØ     END
```

5.2.2.12. Unterprogrammtechnik

GOSUB,RETURN

Befehlsstruktur: GOSUB Zeilennummer
Anweisung(en) A
Zeilennummer Anweisung

```
        Anweisung(en) B
        RETURN (SUBEND)
```

Mit GOSUB können BASIC-Unterprogramme vom Hauptprogramm aus aufgerufen werden. Wenn eine bestimmte Folge von Anweisungen mehrfach im Hauptprogramm benötigt wird, ist es immer sinnvoll, Unterprogramme einzusetzen. Diese Anweisungen werden einmal als Unterprogramm geschrieben und dann durch "GOSUB" an beliebiger Stelle des Hauptprogrammes abgearbeitet. Die Adresse des Unterprogramms stellt die Zeilennummer der ersten Zeile des Unterprogramms dar.
Es erfolgt eine Fehlerausschrift, wenn diese Zeilennummer in der ersten und dritten Zeile der Befehlsstruktur nicht vorhanden ist oder sie nicht übereinstimmen. Die Anweisung(en) A müssen nicht vorhanden sein. Sind die Anweisung(en) B nicht vorhanden, wird nur eine extrem kurze Zeitschleife realisiert. Diese Anweisung(en) B stellen das eigentliche Unterprogramm dar. Es muß mit "RETURN" abgeschlossen werden, sonst führt es zu Fehlern in der Programmabarbeitung, bei denen keine Fehlerausschrift erfolgt. RETURN bewirkt einen Sprung zu der Anweisung im Programm, die unmittelbar der aufrufenden "GOSUB" Anweisung folgt.

Beispiel:

```
1Ø      FOR A=Ø TO 1ØØ
        PRINT A
        GOSUB 2Ø
        NEXT A
        END
2Ø      LET B=5ØØ
        WHILE B>Ø DO
        LET B=B-1
        LOOP
        RETURN
```

5.3. Programmabschluß

Der Programmschluß wird durch das Symbol

END

gekennzeichnet. Obwohl der Programmabschluß durch "END" nicht in jedem Fall notwendig ist, sollte er stets verwendet werden, da es sonst vor allem bei umfangreichen Programmen zu fehlerhaften Abarbeitungen kommen kann. Man kann auch in einem Programm die Anweisung "END" mehrfach verwenden.

6. Felder, Vektoren und Matrizen

BASIC-8Ø.3x ermöglicht es, Felder zu vereinbaren und zu verarbeiten. Da die Standardvereinbarung keine Felder enthält, müssen sie zusätzlich ergänzt werden. Die Zahl in den runden Klammern gibt dabei die Anzahl der Feldelemente an. Im Unterschied zu anderen BASIC-Versionen sind hier nur eindimensionale Felder möglich. Eine Kontrolle bei Überlauf über die obere Indexgrenze erfolgt nicht. In diesem Fall werden die nachfolgend vereinbarten Plätze angesprochen.

Um Matrizen verarbeiten zu können, benötigt man zweidimensionale Felder. Durch geeignete Indizierung kann man **auch eindimensionale Felder verarbeiten.**
Erfolgt die Abspeicherung von Matrizen zeilenweise, dann kann beispielsweise die Matrix A mit 3 Zeilen und 4 Spalten bei der Feldvereinbarung REAL AA(12) wie folgt abgelegt werden:

Matrixelemente

a11 a12 a13 a14 a21 a22 a23 a24 a31 a32 a33 a34

Speicherplätze

AA(Ø) AA(1) AA(2) AA(3) AA(4).....AA(11)

Die einzelnen Matrixelemente lassen sich nach der Beziehung

```
                    Zeilenindex                    Spaltenindex
Feldname ( - 1          * Spaltenzahl + - 1          )
```

aufrufen.

Beispiel:

Das Element a14 (Zeilenindex = 1, Spaltenindex = 4)
wurde auf AA(Ø*4+3) = AA(3) abgespeichert

Das Element a34 (Zeilenindex = 3, Spaltenindex = 4)
wurde auf AA(2*3+4) = AA(11) abgespeichert

Werden die Matrixelemente spaltenweise abgelegt, erfolgt der Aufruf nach der Beziehung

```
                   Spaltenindex                   Zeilenindex
Feldname ( - 1          * Zeilenzahl + - 1          )
```

Da sich Spaltenindex und Zeilenindex ständig ändern, können zur Verarbeitung größerer Matrizen zwei geschachtelte Laufanweisungen eingesetzt werden.

7. Textverarbeitung

Mit BASIC-8Ø.3x ist eine Textverarbeitung möglich. Einer einfachen Textvariablen kann nur ein Einzelzeichen zugeordnet werden. Für Zeichenketten sind Textfelder erforderlich. Im Bsp. wird der Textvariablen TX die Zeichenkette 'HYPOTHENUSE' zugeordnet, die in der folgenden PRINT-Anweisung wieder aufgerufen wird.

```
1Ø        STRING TX(12)
          LET TX='HYPOTHENUSE='
          INPUT A
          PRINT
          INPUT B
          PRINT
          PRINT TX;SQR(A*A+B*B)
          END
```

Mit der LET-Anweisung wird einer Textvariablen eine Zeichenkette zugeordnet. Dabei muß auf der linken Seite des Gleichheitszeichens die Textvariable stehen.
Es können auch mehrere Zeichenketten verknüpft werden. Nach dem Gleicheitszeichen ist es dann ein Textausdruck, der die gleicheForm wie eine PRINT-Anweisung hat.

Beispiel:

```
1Ø        STRING W1(4);W2(8);W3(12)
          LET W1='LAUT'
          LET W2='SPRECHER'
          LET W3=W1;W2
          PRINT W3
          PRINT W1;W2
          END
```

Es wird zweimal untereinander das Wort "LAUTSPRECHER" ausgedruckt, einmal aus der Verknüpfung W1 und W2 zu W3, zum anderen aus der Verknüpfung in der PRINT-Anweisung.
Es können aber auch Zahlen oder arithmetische Ausdrücke in Zeichenketten gewandelt werden.

Beispiel:

```
1Ø        STRING TX(12)
          LET A=3.5
          LET TX='A='; FORMAT Ø.3;A/2
          PRINT TX
```

Der Text A=1.75 wird ausgedruckt und der Rest der Textvariablen wird mit Leerzeichen aufgefüllt.

7.1. Auswahl von Teilketten

In den letzten Beispielen wurden die Textvariablen als geschlossene Zeichenketten (Felder) behandelt. Von den STRING-Feldern können auch Einzelzeichen oder Teilzeichenketten ausgewählt werden.
Mögliche Schreibweise sind:

```
Stringvariable (index)
Stringvariable (index, länge)
```

Im 1. Fall wird entsprechend dem angegebenen Index ein Zeichen, im 2. Fall ab dem Index eine Teilzeichenkette der fixierten Länge ausgewählt.
Die Zählung des Index beginnt, wie bei Vektoren, mit Null.
Wenn STRING SA(8) vereinbart wurde, dann ist:

- SA(3,5) die Teilzeichenkette von SA(3) bis SA(7), also 5 Zeichen
- SA die gesamte Zeichenkette
- SA(1) das 2. Zeichen der Zeichenkette, da das 1. Zeichen auf SA(Ø) steht

Beispiel:

```
          STRING SA(8)
1Ø        LET SA='ABCDEFGH'
          PRINT SA(3,5)
          PRINT SA
          PRINT SA(1)
          END
```

Displayausschrift:

```
DEFGH
ABCDEFGH
B
```

Teilketten können auch **verknüpft** und anderen Variablen zugewiesen werden.

Beispiel:

```
          STRING STR(13),STS(5)
1Ø        LET STR='TRANSFORMATOR'
          LET STS=STR(Ø,3);STR(5,2)
          PRINT STR ' = ';STS
          END
```

Displayausschrift:

```
TRANSFORMATOR   = TRAFO
```

7.2. Vergleich von Zeichenketten

Zeichenketten können durch Vergleichoperatoren verglichen werden. 2 Zeichenketten sind nur dann gleich, wenn sie die gleiche Länge haben und in jedem Zeichen übereinstimmen. Eine Zeichenkette ist kleiner als eine andere, wenn der Zahlenwert des ersten Zeichens, das von dem entsprechenden Zeichen in der zweiten Zeichenkette abweicht, kleiner als der Zahlenwert des Zeichens in der zweiten Kette ist oder wenn sie ein echter Anfang der zweiten Zeichenkette ist. Der Zahlenwert ist aus dem ASCII-Code zu entnehmen.

Beispiel:
```
ANTON BNTON
ANTON ANTO
ANTON=ANTON
```

Auf der linken Seite des logischen Ausdrucks kann ein beliebiger Textausdruck, auf der rechten Seite darf nur eine Textvariable oder eine konstante Zeichenkette stehen.
Logische Operatoren (AND..) sind nicht möglich. Mit Hilfe der IF- oder WHILE-Anweisung kann die Abfrage formuliert werden.

Beispiel:

Es druckt 8 eingegebene Werte in alphabetischer Reihenfolge aus:

```
          STRING NAME(8Ø),TX(1Ø),TY(1Ø)
1Ø        PRINT FORMAT 1.Ø
          FOR A=Ø TO 7
          PRINT 'NAME':A+1;'=';
          INPUT NAME (1Ø*A,1Ø)
          PRINT
          NEXT A
2Ø        LET TY=' '
          FOR B=Ø TO 7
          PRINT
          LET TX='ZZZZZZZZZZ'
          FOR A=Ø TO 7
          IF TY< NAME(1Ø*A,1Ø) AND NAME(1Ø*A,1Ø)< TX DO
          LET TX=NAME(1Ø*A,1Ø)
          DOEND
          NEXT A
          PRINT TX;
          LET TY=TX
          NEXT B
          END
```

8. Prozeduren und Funktionen

Prozeduren und Funktionen gehören, wie der Unterprogrammaufruf GOSUB, zur Unterprogrammtechnik. Gegenüber Unterprogrammen haben die Prozeduren und Funktionen Vorteile bei der Parameterübergabe. Ist beispielsweise die Matrizenmultiplikation als Unterprogramm formuliert, dann sind vor jedem Unterprogrammaufruf die Matrixelemente auf die im Unterprogramm verwendeten Speicherplätze umzuladen. Die Werte der Ergebnismatrix müssen nach dem Rücksprung aus dem Unterprogramm wieder auf die Ergebnismatrix des Hauptprogrammes ausgespeichert werden. Wenn ein gleicher Algorithmus mit verschiedenen Variablen und Parametern als Unterprogramm ausgeführt werden soll, ist also jedesmal ein Umladen der Werte notwendig.

Prozeduren und Funktionen erlauben eine saubere Trennung der unterschiedlichen Teile des Programmes und geben wertvolle Unterstützung für einen modularen Aufbau umfangreicher Programme. Sie stehen nach dem Hauptprogramm.

8.1. Prozeduren

Sollen N Zeilenvorschube erfolgen, dann kann man im Hauptprogramm die Anweisung

```
FOR I=1 TO N
PRINT
NEXT I
```

schreiben.
Das gleiche Problem läßt sich aber auch als Prozedur formulieren:

```
DEF VORSCHUB (N)
INTEGER I
FOR I=1 TO N
PRINT
NEXT I
RETURN
SUBEND
```

Im Prozedurkopf steht nach DEF der Prozedurname, gefolgt von einer in runden Klammern eingeschlossenen Liste formaler Variablen oder formalen Parameter.
Dem Prozedurkopf darf keine Zeilennummer vorangestellt werden! Zur Kennzeichnung formaler Variablen werden verwendet;

```
# für formale REAL-Variable
% für formale INTEGER-Variable
& für formale BOOLEAN-Variable
$ für formale STRING-Variable
```

Diese Sonderzeichen stehen vor dem Namen und gehören selbst nicht mit zur Variablenbezeichnung. Sollen Felder übergeben werden, ist nur der Feldname (ohne Anzahl der Feldelemente) anzugeben.
Nach den **formalen** Variablen mit vorangestellten Sonderzeichen können weitere REAL-Variable stehen, die als formal Parameter bezeichnet werden. Sie existieren nur in der Prozedur. Ihnen wird beim Aufruf ein aktueller Wert zugewiesen. Im Gegensatz dazu werden den formalen Variablen aktuelle Variable beim Aufruf zugeordnet, die gelesen und beschrieben werden können. Über die formalen Variablen kann die Prozedur dem aufrufenden Programm Ergebnisse übermitteln.

Die formale Parameterliste des Prozedurkopfes muß mit der aktuellen Liste des Prozeduraufrufes in Typ und Anzahl der Variablen und Parameter genau übereinstimmen.

Nach dem Prozedurkopf folgt der Prozedurkörper. In ihm müssen lokale Variable definiert werden, die nur in dieser Prozedur gültig sind. Im anschließenden Anweisungsteil mit vorangestellter Zeilennummer können die lokalen Variablen und die Variablen der formalen Parameterliste verarbeitet werden. Ein direkter Zugriff zu den lokalen Variablen des aufrufenden Programms ist nicht möglich.
Der Rücksprung in das aufrufende Programm wird durch

"RETURN"

bewirkt. Diese Anweisung kann mehrmals in einer Prozedur stehen. SUBEND schließt die Prozedur ab. Entfällt die formale Parameterliste, so muß der lokale Vereinbarungsteil mit REAL beginnen.

8.2. Funktionen

Funktionen bilden innerhalb der Prozeduren einen Sonderfall. Die Funktion übermittel direkt einen Ergebniswert vom Typ REAL. Der Aufbau von Funktionen entspricht dem von Prozeduren. Vor "RETURN" muß noch zusätzlich

FN = arithmetischer ausdruck

stehen. An Stelle von "SUBEND" wird "FNEND" geschrieben. Funktionsnamen bestehen aus dem Buchstaben "FN" gefolgt von einem normalen Namen. Eine Funktion wird innerhalb eines Ausdruckes durch Angabe des Namens und gegebenenfalls einer Parameterliste analog der Standardfunktionen aufgerufen. Der von der Funktion zurückgegebene Wert nimmt die Stelle des Funktionsaufrufs ein.
Eine Funktion muß mindestens aus zwei Zeilen bestehen. Ein rekursiver Aufruf von Funktionen ist möglich.

Beispiel:

Die Berechnung des Gesamtwiderstandes zweier parallelgeschalteter ohmscher Widerstände ist mit Hilfe einer Funktion zu programmieren.

```
            REAL R1,R2;RP
1Ø          REM IM HP WERDEN DIE WIDERSTAENDE EIN- UND DAS
            ERGEBNIS AUSGEGEBEN
2Ø          PRINT 'R1=';
            INPUT=R1
            PRINT 'R2=';
            INPUT=R2
            PRINT
            LET RP=FNPAR(R1,R2)
            PRINT 'PARALLELWID=';RP
            GOTO 2Ø
            END

            DEF FNPAR(A,B)
3Ø          LET FN=A*B/(A+B)
            RETURN
            FNEND
```

A und B fungieren als formale Variablen. Durch den Funktionsaufruf werden diesen die aktuellenVariablen R1 und R2 zugeordnet. Durch die LET-Anweisung in Zeile 3Ø wird der Ergebniswert dem Funktionsnamen zugewiesen. Damit ist er für das Hauptprogramm verfügbar.

9. Testhilfe

Es werden zwei Testmöglichkeiten unterschieden, die auch kombiniert verwendet werden können:

TRACE und Schrittbetrieb

TRACE

"Trace" heißt "Spur", d.h. im Tracebetrieb kann die Spur, in der das Programm läuft, alle durchlaufenen Anweisungen aufzeichnen. Im Tracebetrieb wird das Programm also nicht angehalten, es läuft ggf. im Echtzeitbetrieb, allerdings wird die Rechenzeit etwas höher. Der Rechenzeitzuwachs ist abhängig vom Umfang der Traceprozedur und von der Anwenderprogrammstruktur und kann sich um den Faktor 1,1 bis 1Ø erhöhen.

Mit

TRON

="Trace on" wird der Tracebetrieb eingeschaltet, mittels

TROFF

="Trace off" wird er abgeschaltet. Die Anweisungen "TRON" und "TROFF" werden bei Bedarf in das Anwenderprogramm eingearbeitet. Damit kann also auch nur ein Teil des Programms kontrolliert werden. Initial nach RUN ist Trace abgeschaltet (TROFF). Bei eingeschaltetem Trace wird nach jeder Anweisung die TRACE-Prozedur durchlaufen. Diese ist als normale BASIC-Prozedur realisiert und eingekettet. In dieser TRACE-Prozedur kann der Anwender selbst festlegen, was bei TRACE erfolgen soll. Dabei hat er Zugriff auf den Prozedurnamen, der mit Trace abgearbeitet wird und auf die Zeilennummer der nachfolgenden Anweisung. Dazu dienen die Funktionen "ERRP¤" und "ERRL". Ein Zugriff auf die lokalen Variablen der zu testenden Prozedur ist nicht möglich.
Im einfachsten Fall wird die nachfolgende Zeilennummer jeweils aufeinanderfolgend auf den Bildschirm geschrieben. Die TRACE-Prozedur lautet dann:

```
DEF TRACE
REAL A
PRINT ':' ;ERRL;
END
```

bzw.

```
DEF TRACE
REAL A
IF ERRL<>Ø DO
PRINT ':';ERRL
END
```

Dieser Fall entspricht dem Leistungsvermögen anderer verbreiteter BASIC-Interpreter. Die TRACE-Prozedur läßt sich wesentlich komplexer gestalten. Beispielsweise soll eine globale Variable überwacht werden. Nur wenn sich deren Wert ändert, soll der Wert und die Programmstelle (nachfolgende Zeilennummer) auf dem Display dokumentiert werden. Die TRACE-Prozedur muß dann mit folgendem Inhalt versehen werden:

a) Ergänzung im globalen Vereinbarungsteil:

```
GLOBAL %                ; schaffen von zwei zusätzlichen
REAL# VA,VZ             ; globalen Variablen
```

- die zu überprüfende Variable ist ebenfalls global und heißt X1.

b) TRACE-Prozedur:

```
            DEF TRACE
            REAL A
1Ø          REM Zeilennummer merken für eine nichtnummerierte
                Zeile
2Ø          IF ERRL<>Ø THEN VZ=ERRL
3Ø          REM Variable testen
4Ø          IF VA<>X1 DO  VA=X1
5Ø          PRINT 'X1=';VA;':';VZ
6Ø          DOEND
            END
```

Erläuterung

2Ø: Die Funktion "ERRL" liefert den Wert Ø, wenn eine nichtnummerierte Zeile ansteht. Die Zeile 2Ø realisiert die Speicherung der jeweils voranstehenden Zeilennummer in der Variablen VZ.

4Ø: Nur wenn eine Änderung der Variablen X1 vorliegt, wird der nächste Block durchlaufen.

5Ø: In diesem Fall erfolgt die Ausschrift (bsp:)

```
....X1=1Ø5.7:12Ø
```

Je umfangreicher die TRACE-Prozedur ist, desto mehr Rechenzeit verlangt der TRACE-Betrieb. Jedoch läßt sich der Trace-Modus gezielt einschalten (TRON/TROFF), damit ist auch eine hohe Rechenzeit vertretbar.
Ist keine Prozedur "TRACE" vorhanden, so wird kein "TRACE" ausgeführt.

Die Funktion zur Fehlerstellenanzeige

Die Funktion zur Fehlerstellenanzeige "ERRL,ERRN,ERRP¤ " werden ebenfalls zur Anzeige der TRACE-Programmstelle benutzt.

ERRL ist eine numerische Funktion und übergibt die Zeilennummer der nachfolgenden Zeile, wenn diese nummeriert ist, oder Ø, wenn die nachfolgende Zeile keine Nummer besitzt.

ERRN ist eine numerische Funktion und liefert die Fehlernummer des aufgetretenen Fehlers.

ERRP¤ liefert als Stringfunktion den Namen der Prozedur, indem der Fehler auftrat, d.h. den Namen der zu "ERRL" zugehörigen Prozedur.

STOP und Schrittbetrieb

Nach Abarbeitung der Anweisung "STOP" folgt eine Annahme des Betriebssystems-Kommandoabfragezustandes. Der Abarbeitungszustand des Programms bleibt jedoch erhalten, so daß dieses fortgesetzt werden kann.
Die Unterbrechungsstelle wird angezeigt, zum Beispiel:

```
STOP AT 11ØA=B  IN PRG1 STACKBYTES:2ØØ
```

11Ø A=B ist die anstehende Anweisung, die nachfolgend abgearbeitet wird

PRG1 ist der Prozedurname

2ØØ ist die Zahl der aktuell belegten Bytes im lokalen Stack.

Im Kommandomodus wird vorzugsweise das Kommando "NEXL" angeboten, so daß sofort ein Schrittbetrieb erfolgen kann.
Nach Betätigen von OFF kann ein anderes Kommando angewählt werden. Es können Variablenbelegungen angezeigt werden, Variablen können verändert werden usw. Es können alle BASIC-Anweisungen als Kommando abgearbeitet werden, einschließlich des Aufrufes kompletter Prozeduren. Anweisungen wie "GOTO", die Programmsprünge ausführen, werden nicht abgearbeitet.
Das Programm kann mittels dem Kommando "CONT" an der Unterbrechungsstelle fortgesetzt werden. Das Kommando "NEXL" ermöglicht den Schrittbetrieb.

Programmkorrekturen während des Zustandes STOP

Generell sind keine Korrekturen im lokalen Vereinbarungsteil zulässig. Der globale Vereinbarungsteil kann in zulässiger Weise geändert werden. Programmänderungen sind nur hinter der Unterbrecherstelle zulässig.
Da der Programmzustand beibehalten wird, darf das Programm nicht unzulässig verändert werden.
RUN stellt auf jeden Fall den Grundzustand ein.
Der Betriebssystem-Kommandomodus ist bzgl. Stackbelegung usw. dem BASIC-Zeileninterpretationszustand gleichgestellt.

10. Anlagen (A)

A 1 ASCII-Zeichenvorrat

Hex	ASCII	Bedeutung	
ØØ	NUL	NULL	Null
Ø1	SOH	START OF HEADING	Kopfzeilenbeginn
Ø2	STX	STAR OF TEXT	Textanfangszeichen
Ø3	ETX	END OF TEXT	Textendezeichen
Ø4	EOT	END OF TRANSMISSION	Ende der Übertragung
Ø5	ENQ	ENQUIRY	Aufforderung zur Datenübertragung
Ø6	ACK	ACKNOWLEDGE	Positive Rückmeldung (ENTER)
Ø7	BEL	BELL	Klingelzeichen
Ø8	BS	BACKSPACE	Rückwärtsschritt
Ø9	HT	HORIZONTAL TABULATION	Horizontaltabulator
ØA	LF	LINE FEED	Zeilenvorschub
ØB	VT	VERTICAL TABULATION	Vertikaltabulator
ØC	FF	FORM FEED	Seitenvorschub
ØD	CR	CARRIAGE RETURN	Wagenrücklauf
ØE	SO	SHIFT OUT	Dauerumschaltungszeichen
ØF	SI	SHIFT IN	Rückschaltungszeichen
10	DLE	DATA LINK ESCAPE	Datenübertragungsumschaltg
11	DC1	DEVICE CONTROL 1	Gerätesteuerzeichen 1
12	DC2	DEVICE CONTROL 2	Gerätesteuerzeichen 2
13	DC3	DEVICE CONTROL 3	Gerätesteuerzeichen 3
14	DC4	DEVICE CONTROL 4	Gerätesteuerzeichen 4
15	NAK	NEGAT. ACKNOWLEDGE	Negative Rückmeldung
16	SYN	SYNCHRONOUS IDLE	Synchronisierung
17	ETB	END OF TRANSMISSION BLOCK	Ende des Datenübertragungsblocks
18	CAN	CANCEL	Ungültig
19	EM	END OF MEDIUM	Ende der Aufzeichnung
1A	SUB	SUBSTITUTE	Substitution
1B	ESC	ESCAPE	Umschaltung
1C	FS	FILE SEPARATOR	Hauptgruppentrennzeichen
1D	GS	GROUP SEPARATOR	Gruppenkennzeichen
1E	RS	RECORD SEPARATOR	Untergruppentrennzeichen
1F	US	UNIT SEPARATOR	Teilgruppentrennzeichen
2Ø	SP	SPACE	Leerzeichen
21	!	EXCLAMATION POINT	Ausrufezeichen
22	"	QUOTATION MARK	Anführungszeichen
23	#	NUMBER SIGN	Nummerzeichen
24	¤ ≙$	DOLLAR SIGN	Dollarzeichen
25	%	PERCENT SIGN	Prozentzeichen
26	&	AMPERSAND	Kommerzielles UND-Zeichen
27	'	APOSTROPHE	Hochkamma
28	(	OPENING PARENTHESIS	runde Klammer (offen)
29	)	CLOSING PARENTHESIS	runde Klammer (geschlossen)
2A	*	ASTERISK	Stern
2B	+	PLUS	Pluszeichen
2C	,	COMMA	Komma
2D	-	HYPHEN (MINUS)	Bindestrich (Minuszeichen)
2E	.	PERIOD (DECIMAL)	Punkt
2F	/	SLANT	Schrägstrich

Hex	ASCII	Bedeutung	
3Ø	Ø		
31	1		
32	2		
33	3		
34	4		
35	5		
36	6		
37	7		
38	8		
39	9		
3A	:	COLON	Doppelpunkt
3B	;	SEMI-COLON	Semikolon
3C	<	LESS THAN	Kleiner als
3D	=	EQUALS	Gleichheitszeichen
3E	>	GREATER THAN	Größen als
3F	?	QUESTION MARK	Fragezeichen
4Ø	@	COMMERCIAL AT	Kommerzielles a-Zeichen
41	A		
42	B		
43	C		
44	D		
45	E		
46	F		
47	G		
48	H		
49	I		
4A	J		
4B	K		
4C	L		
4D	M		
4E	N		
4F	O		
5Ø	P		
51	Q		
52	R		
53	S		
54	T		
55	U		
56	V		
57	W		
58	X		
59	Y		
5A	Z		
5B	[	OPENING BRACKET	eckige Klammer(offen)
5C	\	REVERSE SLANT	Schrägstrich (links)
5D	]	CLOSING BRACKET	eckige Klammer(geschlossen)
5E	^	CIRCUMFLEX	Zirkumflex
5F		UNDERSCORE	Unterstrich
6Ø	`	GRAVE ACCENT	Akzent,gravis

Hex	ASCII	Bedeutung	
61	a		
62	b		
63	c		
64	d		
65	e		
66	f		
67	g		
68	h		
69	i		
6A	j		
6B	k		
6C	l		
6D	m		
6E	n		
6F	o		
7Ø	p		
71	q		
72	r		
73	s		
74	t		
75	u		
76	v		
77	w		
78	x		
79	y		
7A	z		
7B	{	OPENING BRACE	geschweifte Klammer(offen)
7C	|	VERTICAL LINE	Vertikalstrich
7D	}	CLOSE BRACE	geschweifte Klammer(geschlossen)
7E	~	TILDE	Überstreichung
7F	DEL	DELETE (RUBOUT)	Löschen DEL

A 2 BASIC-Softwareschnittstellen

INPUT : DE = Zeiger auf die nächste Position des Zwischencodes BASIC, der abgearbeitet werden soll

IY = arithmetischer Stackpointer, arbeitet decrementierend

IX = Zeiger auf lokale Variable (ein Stack)

OUTPUT: DE = Zeiger auf nächste abzuarbeitende Position im Zwischencode, ggf. bei Parameterübernahme verändert

IY = unverändert

IX = unverändert

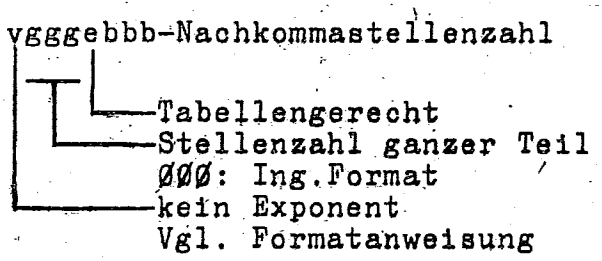
CY = Ø keine Fehlermeldung

CY = 1 Syntaxfehler führt zur Unterbrechung des BASIC-Programmes

Tabelle gibt über verwendbare Unterprogramme im BASIC-Interpreter Auskunft.

Name	Adr.	INPUT	OUTPUT	Bedeutung
TRM	CA6Ø	DE IY IX	DE BCHL	Abarbeitung eines arithm. Ausdruckes , Zeiger auf Zwischencode, wird inkrementiert, Gleitkommazahl als Ergebnis
TTX	CA63	DE HL (IY) C	DE HL (HL)	Abarbeitung eines String-Ausdruckes, Zeiger auf Zwischencode, Adresse des Textempfangspuffers Zahl der erzeugten Zeichen, Länge des Textempfangspuffers, aus Zwischencode ermittelter Text
VAD	CA66	DE IY IX C	DE HL B C	Ermitteln der Adresse einer Variablen, Zeiger auf Zwischencode, Variablenadresse Variablentyp Bits b4b3: ØØ REAL Ø1 INTEGER 1Ø BOOLEAN 11 STRING STRING-Länge oder BOOLEAN-Marke

Name	Adr.	INPUT	OUTPUT	Bedeutung
FKI	C63C	BCHL		Festkommainvertierung Gleitkommazahl
			HL	Festkommazahl, begrenzt
ZSS	CA5D	DE	DE	Zeichen in Textpuffer einschreiben
		(IY)	(IY)	Textpufferzeiger
			(DE)	noch verfügbare Textpufferlänge (IY) > Ø! Text
ZGL	CC88	DE (DE)	DE	ganze Zahl auf Textpuffer lesen Textpufferzeiger enthält Ziffernfolge
			HL	Zahl, positiv
GZL	CA72	DE	DE	Gleitkommazahl lesen Textpufferzeiger
		(DE)		enthält allg. Zahl mit VZ, z.B. +12.34E-5
			BCHL	Zahl im Gleitkommaformat
GZS	CA78	DE (IY)	DE	Gleitkommazahl ausschreiben Textpufferzeiger Formatbyte



	BCHL	Zahl in Gleitkommaformat
	(DE)	Zahl in Zeichenfolge

A 3 Zusammenfassung der Bedienfunktionen BASIC im MC 80.30/31

- Gerät einschalten
- BASIC-Editor von Kassette laden, auf Adresse 4000H
- BASIC-Interpreter von Kassette laden, auf Adresse C000H
- CATA ENTER BEDIT; BNEW; BDEF; BDEL müssen vorhanden sein
  ebenso Kennbyte 00 auf 6000H
  Kennbyte 1B auf C000H
- voraussichtliche Länge festlegen, BNEW länge ENTER
- BEDIT ENTER = Eintritt in Editormodus des BASIC-Hauptprogrammes
- mit BDEF name, ENTER können BASIC-Unterprogr. festgelegt werden
- löschen erfolgt durch BDEL name, ENTER
- diese sind durch BEDIT name, ENTER oditierbar
- Editormodus meldet sich mit DEF (name)
- Anweisungen streichen
- cOFF zurück zum Betriebssystem

A 3.1 Bedienkommandos des BASIC-Editors

Eingabemodus: Einfügen einer zusätzlichen Zeile

Die Eingabezeile, d.h. die drei unteren Displayzeilen sind leer, der Kursor steht links, Eingabebereitschaft
Eintragen der Anweisung mittels der gleichen Tasten wie zur Korrektur der Anzeigezeile, Abschluss mit ENTER
Der Eingabemodus wird für weitere Einfügungen beibehalten.
Die folgenden Kommandos (Kommandotasten im Unterschied zu Texttasten) gelten in beiden Modi und stellen nach Kommandoausführung den Anzeigemodus her.

Kommandos im BASIC-Editor

cW = Sprung zum Ende

cA = Sprung zum Anfang

cG = gehen zu Zeilennummer

cE = Anweisung einfügen

cS = Anweisung streichen

cO = eine Seite hoch

cU = eine Seite runter

c↑ = eine Zeile hoch

c↓ = eine Zeile runter

cOFF = verlassen des Editors

A 3.2 Fehlerausschriften

nach ENTER: Syntaxfehler bei Übersetzung erkannt, Kursor steht unter dem nicht identifizierbaren Wert, Zeile wird nicht angenommen.

nach RUN: >> SYNTAX

Syntaxfehler bei Abarbeitung wurde erkannt
Kennzeichnung in der Anweisung mit >>
- OFF betätigen (Anzeigegrundzustand)
- Anweisungen korrigieren
- RUN: Programm erneut starten

>> Zuweisung

Programmklammern sind syntaktisch falsch
Anzeige der fehlerhaften Anweisung
DO...ELSEDO....DOEND
DO...DOEND
WHILE...DO....LOOP
TO...NEXT

nach ENTER: >> Speicher voll

Diese Ausschrift erfolgt, wenn eine Anweisung eingegeben werden soll und der Programmspeicherbereich würde dabei überfüllt werden.

A 4 Beispiel "Nachladen eines BASIC-Programmes"

4.1. Voraussetzung für das Nachladen eines BASIC-Anwender-programmes sind folgende Aktivitäten:

- Gerät einschalten
- Speicher initialisieren
- Laden des BASIC-Interpreters -Editors, sowie des GRAB-30.

Ablauf:

. Einlegen Systemkassette

. Befehlseingabe "READ BASIC",
"ENTER"

. Kontrolle der RAM-Kettung über Befehlsaufruf "CATA" (Kettung muß bis Adr. CØØØ geschlossen und Freibereich auf Adr. 6ØØØ bis 8FFF definiert sein).

- Einlegen der Anwenderkassette mit BASIC-Programm des Namens "ABC" (Beispielname) und der Programmlänge 1ØØØ (Beispiellänge).
- Laden des Programmes mit "READ ABC",
"ENTER"
- Kontrolle der RAM-Kettung über Befehlsaufruf "CATA" (RAM-Kettung muß bis Adr. 7ØØØ geschlossen sein).
- Schließen des "RAM-Loches" zwischen Adr. 7ØØØ (Ende Anwenderprogramm) und Adr. 9ØØØ (Beginn Graphic-Interpreter) durch Befehlseingabe "RKET Ø1,7ØØØ,9ØØØ"
"ENTER"
- Kontrolle der RAM-Kettung über Befehlsaufruf "CATA" (Kettung muß bis mindestens CØØØ geschlossen sein).
- Start des BASIC-Programmes mit
"RUN ABC"
"ENTER"

4.2. Soll das gemäß Punkt 4.1 geladene BASIC-Programm gelöscht werden, so ist der Befehl

"BDEL ABC",
"ENTER"

zu realisieren.

Der für das BASIC-Programm definierte Arbeitsbereich bleibt dabei erhalten.

Weitere nachgeladene BASIC-Programme werden an diesen Arbeitsbereich (Kennzeichnung in "CATA" durch "E") angegliedert.

Ist es erforderlich, daß ein weiteres BASIC-Programm in dem durch "BDEL ABC",
"ENTER"

gelöschten Speicherbereich eingelagert werden soll, ist wie folgt zu verfahren:

Ablauf:

- "FILL 6ØØØ,7ØØØ,FF",
  "ENTER"
  (Befehl löscht den mit der Einlagerung des BASIC-Programmes geschaffenen Arbeitsbereich).

- "RKET Ø1,6ØØØ,9ØØØ"
  "ENTER"
  (Befehl kettet RAM bis Adr. 9ØØØ, d.h. bis Beginn "GRAB 30").

- "READ ABC",
  "ENTER"
  (Nachladen des Programmes ABC auf bisher genutzten Arbeitsbereich).

- "RUN ABC",
  "ENTER"
  (Start des geladenen Programmes).