Ingenieurschule für Elektrotechnik
"Hanno Günther"
Velten-Hohenschöpping

Schulinternes Lehrmaterial

BASIC - 80

für den Mikrocomputer MC 80 des VEB Elektronik Gera
(Teil 1)

Stand: Mai 1984

1. BASIC-80 - Grundlagen

BASIC-80 wurde auf der Basis des Prozessors U 880 realisiert. Die Programmlänge des Softwarepaketes beträgt 8 KByte. Davon sind 4 KByte Interpreter und 4 KByte Editor.
BASIC-80- Programme werden nicht in den Maschinencode übersetzt (keine Compilierung). Die Abarbeitung von BASIC-80-Programmen erfolgt nach dem Interpreterprinzip. Der BASIC - Editor erzeugt aus BASIC - Befehlen einen speicherverschieblichen Zwischencode. Der Interpreter löst die Abarbeitung einer Folge von Maschinenbefehlen (U 880) aus, die vom Zwischencode bestimmt werden und das vom Anwender erstellte BASIC - Programm realisieren. Jedes BASIC - Programm ist nur mit dem Interpreter und dem Zwischencode lauffähig. Der Interpreter ist ein 4 KByte Programmpaket, das sich ab der Adresse 5000 bis 5FFF (hexadezimal) befinden muß und die Arithmetik und weitere wichtige Unterprogramme bereits mit enthält. Der Zwischencode mit dem Interpreter ist damit nach erfolgter Programmerstellung im MC 80 auf jeder anderen Mikrorechnerkonfiguration mit dem Prozessor U 880 lauffähig. Das weitere 4 KByte - Programmpaket BASIC - Editor dient mit dem Interpreter zur Erstellung und zum Test der BASIC - Programme.
Der Editor ist nur im MC 80 lauffähig.
Der Zwischencode belegt weniger Speicherplatz als ein vergleichbares compiliertes Maschinencodeprogramm. Bei längeren Programmen (ab etwa 6 KByte) ergeben sich damit erhebliche Speicherplatzeinsparungen.
Das Abarbeitungsprinzip des Interpreters bedingt eine etwa 1,5 fache Laufzeit im Vergleich zu dem von einem Compiler erzeugten Programm. Dieser Zeitfaktor ist bei der Problembearbeitung zu kalkulieren.

Die wesentlichen Bestandteile von BASIC-80 sind:

- Programmverzweigungen
- Ausgabeanweisungen
- vier Variablentypen
- Feldverarbeitung
- Vereinbarung von Assemblerfunktionen
- Gleitkommaarithmetik, Standardfunktionen.

Bei Programmverzweigungen werden bedingte Sprünge, Schleifen und Laufanweisungen unterschieden. Ausgabeanweisungen ermöglichen ein beliebiges Beschreiben des Bildschirmes, auch mit einfachen graphischen Darstellungen.
Es gibt REAL-, INTEGER-, BOOLEAN- und STRING-Variable, die als lokale, globale oder formale Variable vereinbar sind. In BASIC-80 können Vektoren vereinbart und verarbeitet werden. Da eine beliebige Indizierung der einzelnen Elemente möglich ist, können Matrizenberechnungen ausgeführt werden.
Vereinbarungen von Assemblerfunktionen sind in BASIC-80 möglich und enthalten die Speicheradresse, ab der das gewünschte Assemblerprogramm steht.

2. EDITOR BASIC-80

Der BASIC-Editor dient dem Erzeugen des Zwischencodes, dem Test erstellter Programme und der Korrektur (Überschreiben, Einfügen, Streichen) von Anweisungen.

Auswahl

Die Auswahl von BASIC-80 erfolgt analog zu anderen Softwareroutinen des MC 80 über die Menütabelle. Nach Neueinschalten des Gerätes ist vorher INIT anzuwählen. Nach Aufruf meldet sich BASIC-80 mit der Ausschrift:

BASIC %C0ØØ %E0ØØ

Diesen beiden hexadezimal angegebenen Adressen sind Vorzugsadressen, die den von BASIC verwendeten RAM-Bereich kennzeichnen. Sie können bei Bedarf geändert werden. Das erfolgt durch Überschreiben. Nach jeder Eingabe ist 'ENTER' zu betätigen. Bei der Anfangsadresse ist der niederwertige Teil immer ØØ und kann nicht geändert werden.
Wird BASIC während des Programmierens oder Testens verlassen und erneut angewählt, wird der vorher gewählte RAM-Bereich wieder angeboten.

Nach erfolgter Anwahl befindet sich der Editor in der ersten Zeile in Eingabebereitschaft. Über dieser ersten Programmzeile befindet sich der in zwei Teile gegliederte Vereinbarungsteil.
Der auf dem Display sichtbare lokale Vereinbarungsteil beginnt mit

DEF HAUPTPROGRAMM

Der darauf folgende lokale Vereinbarungsteil enthält die REAL -Vereinbarungen der Buchstaben A...Z. Das bedeutet, daß jedem Buchstaben eine Gleitkommazahl (real) zugeordnet werden kann. Jeder Buchstabe A...Z ist also Name einer REAL - Variable.
Der globale Vereinbarungsteil steht nicht sichtbar über dem lokalen Vereinbarungsteil am Anfang des gesamten Programmsystems. Über die Taste '↑' kann man ihn erreichen. Diese Vereinbarungen dürfen nicht unzulässig geändert werden.
Mit der Taste '↓' kann man sich im Programm wieder vorwärts bewegen, bis man am Ende den Eingabezustand wieder erreicht.

Bedienung, Bedienkommandos

Im Anzeigegrundzustand werden 6 Anweisungen angezeigt. Die 5 oberen Anweisungen sind einzeilig dargestellt, die letzte Anweisung wird vollständig ausgeschrieben. Eine Anweisung kann maximal drei Displayzeilen (95 Zeichen) umfassen.
Die unterste Anweisung, nach derem letzten Zeichen der Kursor steht, kann mittels Überschreiben und anschließendem Betätigen der ENTER-Taste korrigiert werden. Zum Überschreiben können alle Zeichentasten und die Tasten '↑' '↓' sowie 'CL' und die Tastenfunktionen cF und cL verwendet werden. Sie wirken repitierend.
Mit den beiden Tasten '↓' und '↑' kann jede Stelle im Programm angewählt werden.

Jeweils die untere Zeile der Displayausschrift steht zur Korrektur bereit. Wird das Programmende erreicht, so wird automatisch auf den Eingabemodus umgeschaltet und weitere Anweisungen können hinzugefügt werden.
Im Eingabemodus ist das Display ab der 6. Zeile frei, die davorstehenden Anweisungen werden auf dem Display darüber einzeilig dargestellt. Es kann eine Anweisung oder ein Kommando eingetragen werden, der Abschluß (Übernahme) erfolgt mit 'ENTER'. Kommandos werden sofort ausgeführt, Anweisungen mit Zeilennummer werden in das Programm einsortiert und Anweisungen ohne Zeilennummer werden in die laufende Anweisungsfolge eingetragen. Die erste Anweisung eines Programmes muß mit einer Zeilennummer versehen werden. Bei der Eingabe von Anweisungen in das Programm wird dieser Eingabemodus solange beibehalten, bis eine andere Kommandotaste betätigt wird ('OFF', '↓', '↑', vgl. Tabelle).
Sollen in ein Programm Anweisungen eingefügt werden, wird nach Anwahl der Vorgängerzeile 'ENTER' betätigt. Im angenommenen Eingabemodus kann nun eine beliebige Anzahl von Anweisungen ergänzt werden.
Werden die Anweisungen mit Zeilennummern versehen, erfolgt ein automatisches Einsortieren und der Eingabemodus kann ab diesem Bereich weiter verwendet werden.
Alle zeichenerzeugende Tasten und die Tasten '↵', '←' dienen der Texteingabe. Alle anderen Tasten '↓', '↑' 'OFF', 'ENTER', '⇥' sowie die Tasten der Control - Ebene sind Kommandotasten. Ihre Wirkung ist in der nachfolgenden Tabelle aufgeführt.
Die Tasten der Control - Ebene sind erreichbar durch gleichzeitiges Betätigen der Controltaste (schwarze Taste mit Gefühlspunkt) und der angegebenen Zeichentaste. In der Tabelle sind diese Tastenfunktionen mit einem vorangestellten c gekennzeichnet.

Taste	Aktivität
↓	Anwahl der nächsten Anweisung
↑	Anwahl der vorherstehenden Anweisung
OFF	Herstellen des Anzeigegrundzustandes
cA	Annahme des Eingabezustandes am Programmanfang
cS	Streichen der angewählten Anweisung
cL	Löschen des Zeichens auf der Kursorposition, der nachfolgende Text wird herangeschoben
cP	Erzeugen eines Leerzeichens auf der Kursorposition, der nachfolgende Text wird weggeschoben. Damit können im Text Einfügungen realisiert werden.
ENTER	Eingabezeile enthält Anweisung ohne Zeilennummer: Die Anweisung wird in die laufende Anweisungsfolge eingetragen, auch Korrektur der Anweisung ist möglich
ENTER	Eingabezeile enthält Anweisung mit Zeilennummer: Die Anweisung wird entsprechend der Zeilennummer in das Programm einsortiert, die Anzeige erfolgt ab der neuen Position. Doppelte Zeilennummervereinbarungen werden angenommen.
ENTER	Eingabezeile enthält nur die Zeilennummer: Anzeige ab dieser Zeilennummer
ENTER	Eingabezeile enthält ein Kommando laut nachfolgender Tabelle: Ausführung dieses Kommandos

Taste	Aktivität
ENTER	Zuvor wurde eine andere Kommandotaste ('↑', '↓', 'OFF') betätigt: Annahme des Eingabemodus; ermöglicht das Einfügen weiterer Anweisungen.
cN	Diese Funktion wird nur wirksam nach Programmunterbrechung mittels STOP - Anweisung oder OFF nach RUN! Ansonsten erfolgt die Fehlerausschrift >> DEF-EBENE. Schrittbetrieb oder Direktmodus
cC	Programmfortsetzung nach STOP oder OFF nach RUN

Kommando	Aktivität (Alle Kommandos nur im Eingabemodus eintragen)
NEW	Löschen des gesamten von BASIC beanspruchten RAM-Bereiches, Treffen der Initialvereinbarungen
RUN	Start des geschriebenen Programms am Programmanfang. Es können Fehlermeldungen auftreten
	Das laufende Programm kann mit 'OFF' unterbrochen werden. Dabei wird der Anzeigegrundzustand an der Unterbrechungsstelle angenommen.
BYE	Verlassen des BASIC - Systems
PUT name	Auslagern des geschriebenen Programms
PUT name Zeilennummer	Auslagern bis ausschließlich Zeilennummer
PUTD	wie PUT Löschen des Quellenbereiches
GET name	Einfügen eines vorher ausgelagerten Programmstückes
LIST	Ausdrucken des Gesamtprogrammes auf einem Drucker

Arbeit mit PUT und GET

Das Kommando PUT hat zwei Funktionen. Es dient einmal dem Auslagern des Gesamtprogramms, das dann als Assemblerfunktion verwendet werden kann.
Unter Angabe einer Zeilennummer kann mit PUT oder PUTD ein Programmteil ausgelagert werden, der mit GET an anderer Stelle wieder eingefügt werden kann. Damit ist eine Umsortierung der Programmanweisungen möglich. Zu beachten ist, daß dabei die Zeilennummerordnung verletzt werden kann.
Die Kommandos PUT, PUTD sowie GET beziehen sich auf einen Pufferbereich, der als Assemblerfunktion deklariert werden muß. Am Ende des globalen Vereinbarungsteiles muß deshalb hinzugefügt werden (Bsp.):

```
GLOBAL %E000
DCL  PUF
```

Damit wird PUF als Assembleranweisung vereinbart. Im Anwendungsfall wird damit jedoch ein Pufferbereich für PUT und GET geschaffen, der ab der Adresse EØØØ (hexadezimal) beginnt.
Im folgenden Beispiel werden die Kommandos PUTD und GET verwendet:

```
GLOBAL %EØØØ          } am Ende des globalen Vereinba-
     DCL PUF          } rungsteiles anfügen
                        (↑ und ENTER)
     DEF Hauptprogramm
     REAL A,B,C,D,E,F,G,H  ) sichtbarer lokaler Vereinba-
     REAL I,J,K,L,M,N,O,P,Q) rungsteil
     REAL R,S,T,U,V,W,X,Y,Z)
1Ø   FOR A=1 TO 8
     PRINT A,SQR(A)
     NEXT A
2Ø   LET B=1
     LET C=INP
     FOR D=B TO C
     LET B=B+1/C
     NEXT D
3Ø   END
```

Die Zeile 1Ø (↑) wird angewählt. Danach wird das Kommando

PUTD PUF 2Ø

eingegeben und mit 'ENTER' abgeschlossen. Es werden die Anweisungen ab der angewählten Programmzeile bis ausschließlich der Zeile mit der Nummer 2Ø in den Pufferbereich übernommen und aus dem Programm gestrichen.
Danach wird die Zeile NEXT D angewählt, 'ENTER' betätigt und das Kommando

GET PUF

ausgeführt. Damit wird der Pufferinhalt nach der Anweisung NEXT D in das Programm eingefügt und es ergibt sich folgende Anweisungsreihenfolge:

```
2Ø    LET B=1
      LET C=INP
      FOR D=B TO C
      LET B=B+1/C
      NEXT D
1Ø    FOR A=1 TO 8
      PRINT A,SQR(A)
      NEXT A
3Ø    END
```

Man beachte die falsche Zeilennummerreihenfolge. Diese kann gegebenenfalls korrigiert werden.
Um in den Pufferbereich einschreiben zu können, muß er leer sein. Ansonsten erfolgt die Fehlermeldung "≫ SPEICHER VOLL". Das Kommando GET löscht den Pufferbereich, so daß er wieder für das Kommando PUT frei ist.
Mit dem Kommando GET läßt sich ein BASIC-Programm von einem Speicherbereich außerhalb des BASIC-Systems holen. Danach läßt es sich bearbeiten (korrigieren, testen).
Mit PUT ist es dann wieder auf den Originalbereich auszulagern und kann dann als eigenständiges Assemblerprogramm (Task) wieder gestartet werden.

3. Anweisungen

Im Standard - BASIC beginnt jede Anweisungszeile mit einer Zeilennummer, die Anweisungen sind in steigender Reihenfolge der Zeilennummern sortiert. Bei BASIC-80 muß nicht jede Zeile numeriert werden. Zeilennummern sind notwendig:

- Bei der ersten Anweisung des Programms (aber nicht im Vereinbarungsteil),
- nach einer Kommentaranweisung (REM),
- nach IF ... THEN ... als Abschluß der Aktivität,
- als Sprungziel (GOTO).

Zu Beginn ist am MC 80 die Funktion INIT auszuführen und BASIC-80 anzuwählen (Menütabelle).
Danach lassen sich wie im ersten Beispiel die Programme eingeben und starten.

Das Kommando NEW mit Abschluß durch 'ENTER' bewirkt das Löschen aller Eintragungen und Wiederherstellung des Initialzustandes.

3.1. Wertzuweisung 'LET'

BASIC-80 verfügt als Initialvereinbarung über Variable mit den Bezeichnungen A,B,C,...,Z. Mit der Anweisung LET können diesen Variablen Zahlenwerte zugewiesen werden, die direkt angegeben werden oder Ergebnisse eines arithmetischen Ausdruckes sind.

```
1Ø  LET A=5
    LET B=A * A+3
    LET C=B/A * SIN(B)
    LET X=4
```

Arithmetische Ausdrücke werden wie in der Mathematik gewohnt notiert, Punktrechnung geht vor Strichrechnung, Klammern werden mit Vorzugsrecht bearbeitet, mehrfache Klammerung ist möglich. Das Multiplikationszeichen wird mit ' * ' und das Divisionszeichen mit '/' dargestellt. Exponieren ist nicht direkt möglich. Mehrzeilige Bruchdarstellung ist als einzeilige Form mit Klammern aufzulösen. Im Beispiel entspricht der Anweisung

```
LET C=B/A * SIN(B)
```

der arithmetische Ausdruck

$$c = \frac{b}{a} \cdot \sin b$$

Der arithmetische Ausdruck

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

muß notiert werden :

```
LET D=(A+B)/(A-B)
```

Die Argumente der Funktionen sind stets in Klammern () zu setzen.
Die Standardfunktionen sind im Abschnitt 5 beschrieben.
Funktionen können auch selbst definiert werden.

Nach Eingabe der Beispielsanweisungen kann das Programm mit dem Kommando RUN und Betätigen von 'ENTER' gestartet werden. Nach fehlerfreier Abarbeitung erfolgt auf dem Display rechts unten die Meldung READY. Nach Betätigen einer beliebigen Taste steht das Programm wieder im Editormodus am Programmanfang. Über die Kursortaste '↓' kann man sich an das Programmende bewegen. Im Beispiel wurden bei der Programmabarbeitung den Variablen die Werte zugewiesen. Die Zuweisung erfolgt im Programmablauf nach außen hin nicht sichtbar. Die Variablenwerte kann man erst durch eine Ausgabeanweisung erfahren (z. B. PRINT). Wir schreiben deshalb am Programmende hinzu:

```
PRINT A
PRINT B
PRINT C
PRINT X
```

Nach erneutem Start (RUN und ENTER) stehen auf dem Display untereinander die vier zugewiesenen Zahlenwerte.
Bei jedem Programmstart mittels RUN bleiben entgegen manchen anderen BASIC - Versionen vorher zugewiesene Variablenwerte erhalten. Wird eine Variable in einem arithmetischen Ausdruck verwendet ohne ihr vorher einen Wert zuzuordnen, so enthält sie den Wert der größten darstellbaren negativen Zahl $-17Ø.1Ø^{36}$
Eine Fehlermeldung erfolgt nicht. Wenn im Vereinbarungsteil korrigiert wird, können sich Veränderungen der Zuordnung der Werte zu den Variablen ergeben, was beim Neustart zu berücksichtigen ist.

Der Ausdruck nach LET ist nicht als mathematische Gleichung, sondern als Zuweisungsoperation zu verstehen. In anderen höheren Programmiersprachen wird dafür das Zeichen ':=' verwendet. Im Unterschied dazu wird bei Bedingungsabfragen das Zeichen '=' auch als Vergleichsoperator verwendet.
Nach Zeilennummern kann das Schlüsselwort LET auch entfallen. Nach dem Schlüsselwort DO kann ebenfalls direkt ohne LET eine Wertzuweisungsoperation notiert werden.

3.2. Die Funktion 'INP'

Mit der Funktion INP können Texte oder Zahlenwerte über die Tastatur eingegeben werden. Um der Variablen A einen Wert über die Tastatur zuordnen zu können, kann man notieren:

```
10  LET A=INP
```

Die Programmabarbeitung verharrt an dieser Stelle, erwartet auf der aktuellen Kursorposition eine Eingabe. Die Übernahme und Programmfortsetzung erfolgt durch Betätigen von 'ENTER'. Im Beispiel steht nach RUN der Kursor in der unteren Zeile am Anfang. Zwecks besserer Bedienführung ist es günstiger, vorher einen Begleittext auszuschreiben :

```
10  PRINT 'A';
    LET A=INP
```

In diesem Beispiel wird auf der letzten Zeile zuerst "A=" ausgeschrieben. Das Semikolon am Ende der PRINT-Anweisung verhindert die Zeilenschaltung. Die Eingabe wird nach der Ausschrift "A=" erwartet. Die nächste PRINT-Anweisung wird dann ab der neuen, durch die Eingabe veränderten Kursorposition ausgeführt. Eine leere PRINT-Anweisung

```
PRINT
```

bewirkt eine einfache Zeilenschaltung.
Werden Zahlen erwartet, so wird bei keiner oder falscher Eingabe eine Ø übernommen. Die Zahlen müssen mit dem Vorzeichen oder einer Ziffer beginnen. INP kann auch zur Eingabe von Texten benutzt werden.

3.3. Ausgabeanweisungen

Ausgabeanweisungen sind ausschließlich auf den MC 80 - Bildschirm bezogen. Wenn andere Bildschirmgeräte, Drucker oder andere Datenspeicher angesteuert werden sollen, müssen entsprechende Treiberroutinen als Assemblerprogramm ergänzt und im Vereinbarungsteil vermerkt werden.

3.3.1. Die Anweisung 'PRINT'

Die Anweisung PRINT wurde in den vorangegangenen Beispielen bereits mehrfach benutzt, um den Wert von Variablen auf dem Display erscheinen zu lassen. Mit dieser Anweisung kann jeweils genau eine Displayzeile beschrieben werden. Das ist im Normalfall die unterste Displayzeile. Danach wird der gesamte Displayinhalt um eine Zeile nach oben verschoben.
Die Anweisung PRINT kann zum Ausdrucken von Zahlenwerten und Begleittexten verwendet werden. Im Beispiel wird auf dem Display die Ausschrift

B =15 verlangt.

Anweisungen:

```
10 LET B=10+5
   PRINT 'B=';B
```

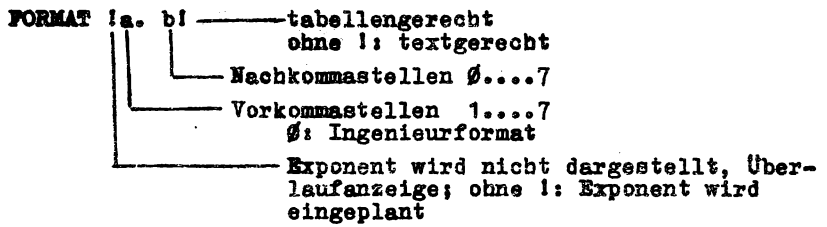
Werden in einer PRINT-Anweisung eine Variable oder ein arithmetischer Ausdruck angegeben, so erscheint auf dem Display der zugehörige Zahlenwert. Auszudruckende Begleittexte werden in '...' geklammert. Das Semikolon ist ein einfaches Trennzeichen. Wird statt dessen als Trennzeichen ein Komma verwendet, so wird ein Leerzeichen zwecks Tabulierung vorgerückt, mehrere Komma sind möglich. Damit können günstig Tabellen gestaltet werden.
Ein Semikolon am Ende der PRINT-Anweisung verhindert die Zeilenschaltung, es kann auf der gleichen Zeile weiter ausgegeben werden.
Mit der PRINT-Anweisung ist es möglich, den Inhalt von Textvariablen wiederzugeben.

3.3.2. Wahl des Ausgabeformates

Bisher wurde das Ausgabeformat von Zahlen nicht beeinflußt. BASIC-80 enthält eine Formatsteuerung, die es gestattet die Stellenzahl vor und nach dem Komma festzulegen.
Wird das Format nicht festgelegt, werden insgesamt 6 Stellen ausgeschrieben. Die Anweisung PRINT 7 bewirkt dann folgende Displayausschrift:

7.ØØØØØ

Zahlen größer/gleich 1ØØØ werden mit Exponent dargestellt, der in Dreierschritten gezählt wird.
Zahlen von 1 bis kleiner als 1ØØØ werden immer ohne Exponent dargestellt.
Das Zahlenformat kann geändert werden, indem die PRINT-Anweisung mit einer Formatsteuerung ergänzt wird.
Die Formatanweisung hat folgende Struktur:



- Ingenieurformat:
  Es wird eine Mantisse im Bereich 1...999.9 und ein Exponent in Dreierschritten dargestellt, z.B.

  1.5
  123
  12.45E+Ø3
  1.267E-12

  Die Nachkommastellenzahl entspricht hier der Gesamtstellenzahl, sie muß ≥ 3 sein, der Dezimalpunkt wird entsprechend der Vorkommastellenzahl (1...3) gerückt.

- Kein Ingenieurformat:
  Die Vorkommastellenzahl beträgt 1 bis7. Ist die Zahl zu groß (größere Vorkommastellenzahl nötig), so wird ein Exponent berechnet und ggf. ausgegeben oder es erfolgt eine Überlaufanzeige. Ist die Zahl kleiner als mit der gewünschten Stellenzahl darstellbar, so wird 'Ø' ausgeschrieben (keine Berechnung von negativen Exponenten).

- Tabellengerecht:
  Die Zahlen werden so ausgeschrieben, daß unabhängig vom Wert die gleichen Druckpositionen belegt werden:
  - Bei Vorzeichen '+' wird ein Leerzeichen ausgegeben
  - führende Nullen werden mit Leerzeichen dargestellt
  - nachfolgende Nullen werden ausgeschrieben
  - außer bei Wahl des Ingenieurformates steht der Dezimalpunkt an der gleichen Stelle
  - tritt kein Exponent auf, kann er aber eingeplant werden (vorderes! nicht gesetzt), indem 4 Leerzeichen dargestellt werden. Nach dem Exponenten wird ein nachfolgendes Leerzeichen angegeben.

- Textgerecht:
  Alle führenden und folgenden Informationen ohne Informationsgehalt werden weggelassen: Vorzeichen '+', führende Nullen, nachfolgende Nullen, Exponent.

- Überlaufanzeige:
  Im Druckbereich wird kein Exponent vorgesehen. Überschreitet die Zahl den möglichen Anzeigebereich, so wird auf der letzten Druckposition ein '%' als Fehlerprotokollierung geschrieben. Ist das vordere '!' nicht gesetzt, so werden bei tabellengerechter Darstellung 5 Leerzeichen bei fehlendem Exponent ausgegeben. Die Überlaufanzeige (Angabe des vorderen !) ist semantisch sinnlos bei Ingenieurformat und teilweise bei textgerechter Darstellung (vgl. Beispiele).

- Rundung:
  Es wird auf die vorgegebene Stellenzahl auf- bzw. abgerundet

- Maximale Stellenzahl:
  Es werden niemals mehr als 7 Stellen ausgegeben.

- Standardformat:
  Wird keine Formatanweisung angegeben, so ist das FORMAT Ø.6! festgelegt.

Eine einmal getroffene Formatvereinbarung wird solange weiterverwendet, bis eine andere Vereinbarung des Formates erfolgt.

Beispiele:

Zahl:	1.5	0.078	1024	-15
FORMAT Ø.4	1.5	78E-Ø3	1.Ø24E+Ø3	-15
FORMAT Ø.4!	␣1.5ØØ␣␣␣␣	␣78.ØØE-Ø3␣	␣1.Ø24E+Ø3␣	-15.ØØ␣␣␣␣
FORMAT 2.2	1.5	Ø.Ø8	1Ø.24E+Ø2	-15
FORMAT 2.2!	␣␣1.5Ø␣␣␣␣	␣␣Ø.Ø8␣␣␣␣	␣1Ø.24E+Ø2␣	-15.ØØ␣␣␣␣
FORMAT !2.2!	␣␣1.5Ø	␣␣Ø.Ø8	␣1Ø.2 %	-15.ØØ

3.3.3. Die Funktion 'TAB'

TAB ist eine Funktion mit einem Argument. Entsprechend dem Ganzzahlanteil des Arguments werden Leerzeichen ausgeschrieben. TAB kann Bestandteil der Ausgabeanweisung PRINT und DPL sein.

Im Beispiel:

```
1Ø  LET A=5
    PRINT '+';TAB(A);'+'
```

werden zwischen den beiden Zeichen '+' 5 Leerzeichen ausgegeben.

Im folgenden Beispiel wird eine Sinushalbwelle mit '*' ausgeschrieben:

```
1Ø  FOR A=Ø TO PI STEP PI/8
    PRINT TAB(SIN(A)*2Ø);'*'
    NEXT A
```

3.3.4. Die Anweisung 'DPL'

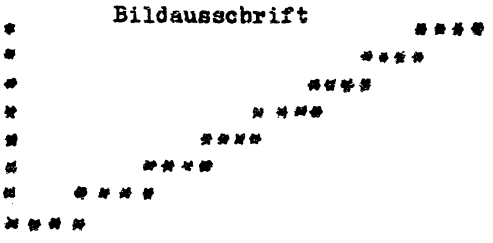
Mit der Anweisung 'DPL' kann das Display an beliebiger Stelle beschrieben werden. Die Argumente der DPL - Anweisung und die Bildschirmausschrift selbst entspricht formmäßig der PRINT-Anweisung. Es erfolgt kein Bildrollen. Nach DPL werden die Zeile und die Spalte angegeben, ab der gedruckt werden soll. Beispiel:

```
1Ø  DPL 4,5 'A=';A
          |  └──Spalte
          └─────Zeile
```

Die Zählung von Spalte und Zeile beginnt mit Ø. Anstelle der Zahlen können auch arithmetische Ausdrücke angegeben werden. Beispiel:

```
1Ø  FOR A=Ø TO 7
    DPK A,Ø '*'
    NEXT A
    FOR A=Ø TO 31
    DPL 7.75-A/4,A '*'
    NEXT A
```

Bildausschrift



3.3.5. Die Anweisung 'CLEAR'

Die Anweisung CLEAR bewirkt ein Löschen des Displays. Nach RUN wird das Display automatisch gelöscht.

3.3.6. Graphische Darstellung

Im BASIC-80 besteht die Möglichkeit, eine analoge Kurve darzustellen. Dafür entfällt die zweite Displayzeile. Die graphische Darstellung wird mit der Anweisung

GRAPH

eingeschaltet und mit der Anweisung

GRAPH AUS

abgeschaltet. Die Zuweisung von Werten erfolgt über die Ausgabefunktion GRA. Mit jeder Wertzuweisung zu GRA wird auf der Kurve rechts dieser Wert zugefügt und die Kurve nach links verschoben.
Die Abszisse hat 253 Werte, die Ordinate umfaßt den Wertebereich von

$$-128 \leq \text{GRA} \leq 127.$$

Werden größere Werte angeboten, erfolgt keine Begrenzung. Im Beispiel wird eine expotentiell abklingende Sinusschwingung dargestellt:

```
1Ø    GRAPH
      FOR A=Ø TO 25.3 STEP Ø.1
      LET GRA=1ØØ*SIN(A)*EXP(-A/2Ø)
      NEXT A
```

Bei Verlassen der Programmbearbeitung bleibt die Graphik eingeschaltet, wenn die Anweisung GRAPH AUS nicht abgearbeitet wurde.

3.4. Programmverzweigungen

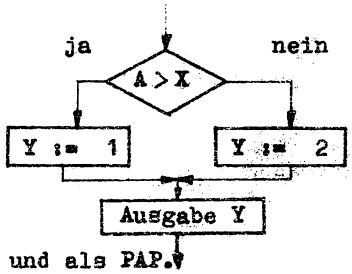
3.4.1. Vollständige strukturierte Verzweigung

Programmzweige müssen oft in Abhängigkeit bestimmter Bedingungen durchlaufen werden. In einem Programmablaufplan (PAP) wird eine vollständige strukturierte Programmverzweigung folgendermaßen dargestellt:

Jeder Zweig kann selbst wieder beliebig in sich strukturiert sein, also weitere Verzweigungen enthalten, die wieder zusammengeführt werden.

Eine Verzweigung könnte z. B. folgendermaßen aussehen:

```
1Ø  LET A=5
    LET X=4
    IF A>X DO
    LET Y=1
    ELSEDO
    LET Y=2
    DOEND
    PRINT Y
```



als BASIC-Programm und als PAP.

Die Bedingung befindet sich zwischen IF und DO.
Der Ja - Zweig wird von DO und ELSEDO eingeschlossen, der Nein - Zweig befindet sich zwischen ELSEDO und DOEND. Die PRINT - Anweisung dient der Kontrolle der Programmausführung und stellt die Programmfortsetzung nach der Verzweigung dar.
Als Vergleichsoperatoren sind zugelassen:

>	größer
<	kleiner
≥	größergleich
≤	kleinergleich
=	gleich
<>	ungleich

Als Bedingung können aber auch beliebige boolsche Ausdrücke verwendet werden. Dazu sei auf die Syntaxdiagramme und spätere Beispiele verwiesen.
Der NEIN-Zweig dieser vollständigen Programmverzweigung kann entfallen, indem der JA- Zweig sofort mit DOEND abgeschlossen wird. ELSEDO tritt dann nicht auf.
Beispiel: Betragsbildung

```
IF A<Ø DO
LET A=Ø-A
DOEND
```

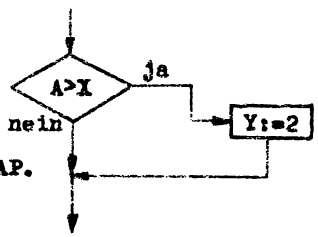
Soll der Ja - Zweig entfallen, können die Anweisungen zwischen DO und ELSEDO weggelassen werden. Meist läßt sich die Bedingung dann aber anders formulieren.
Beispiel: Erhalten des negativen Betrages

```
IF A<Ø DO
ELSEDO
LET A=Ø-A
DOEND
```

3.4.2. Bedingte Abarbeitung einer Anweisung

Eine weitere Formulierungsmöglichkeit der Programmverzweigung ist die IF ... THEN Anweisung. Diese Formulierung ist aus Kompatibilitätsgründen mit anderen BASIC-Versionen in BASIC-80 aufgenommen worden. Sie ermöglicht aber nur die Realisierung des Ja-Zweiges ohne weitere Strukturierungsmöglichkeit. Der Abschluß des Ja-Zweiges wird durch Verwendung einer Zeilennummer bei der nächsten Anweisung im Programm gekennzeichnet.
Dazu folgendes Beispiel:

```
    IF A>X  THEN
    LET Y=2
 2Ø PRINT Y
```



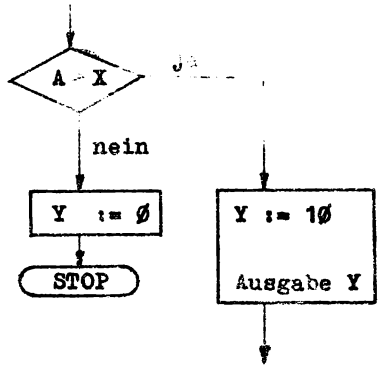
als BASIC-Programm und als PAP.
(Vergleiche auch 3.4.3.)

3.4.3. Bedingter Sprung

Der bedingte Sprung wird mit der IF ... THEN Anweisung realisiert. In Abhängigkeit von einer Bedingung kann der Programmlauf ab einem beliebigen Punkt fortgesetzt werden.

Beispiel:

```
   IF A > X  THEN
   GOTO 1ØØ
4Ø LET Y=Ø
   PRINT Y
   END
   .
   .
   .
1ØØ LET Y=1Ø
    PRINT Y
    .
    .
```



als BASIC - Programm und als PAP.

Ist die Bedingung (A > X) erfüllt, so wird das Programm ab der Zeile 1ØØ fortgesetzt.
Die Möglichkeit birgt die Gefahr der nichtstrukturierten Programmierung. Sie sollte deshalb möglichst vermieden werden.
Eine nicht strukturierte Programmierung ist vom ersten Anschein oft optimaler, zeit- und speicherplatzsparender. Bei Programmveränderungen und -erweiterungen besteht die Gefahr, das eine Vielzahl von GOTO-Befehlen die Übersicht erschwert. Man spricht von sogenannter "Spagettiprogrammierung". In jedem Falle sollte versucht werden, strukturiert zu programmieren.
Nur bei Aussprüngen aus dem Gesamtalgorithmus, z. B. bei Fehlermeldungen und Beenden der Programmbearbeitung kann ein solcher bedingter Sprung nützlich sein.

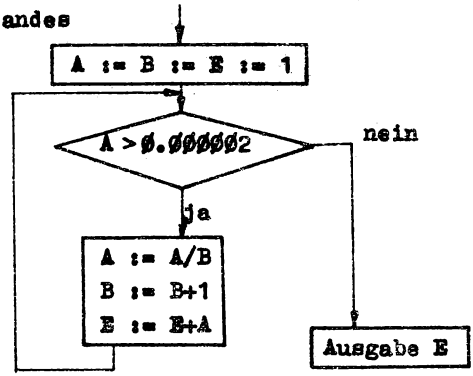
3.4.4. Bedingte wiederholte Ausführung eines Programmblockes

Mit der WHILE - Anweisung kann eine Schleife beliebig oft in Abhängigkeit von einer Bedingung durchlaufen werden. Dabei ist zu garantieren, daß sich die Bedingung während des Schleifendurchlaufes ändert, andernfalls verharrt das Programm an dieser Stelle in einer Eigenschleife. Der Abschluß

der zur Schleife gehörenden Aktivitäten wird mit LOOP gekennzeichnet.
Als Beispiel sei ein Programm zur Berechnung einer Zahlenfolge mit Abbruchkriterium angegeben.

H i n w e i s: NEW und ENTER bewirken das Herstellen des Initialzustandes

```
1Ø  LET A=1
    LET B=1
    LET E=1
    WHILE A> 2E-Ø6  DO
    LET A=A/B
    LET B=B+1
    LET E=E+A
    LOOP
    PRINT E
```



als BASIC-Programm und als PAP.

Das Ergebnis des Programms ist die Zahl e (Basis des natürlichen Logarithmus) nach der Zahlenfolge

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{1!} + \ldots + \frac{1}{n!}\right)$$

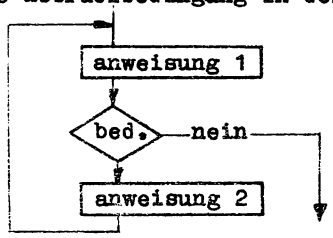
Die Bedingung ist hier das Abbruchkriterium A > Ø.ØØØØØ2. Die Bedingung steht zwischen WHILE und DO, wie bei IF ... DO können auch beliebige boolsche Ausdrücke verwendet werden. Die Anweisungsfolge, die bei erfüllter Bedingung abzuarbeiten ist, steht zwischen DO und LOOP. Bei nicht erfüllter Bedingung wird nach LOOP fortgesetzt.
Ein weiteres Beispiel stellt eine Zeitschleife dar:

```
LET A=5ØØ
WHILE A>Ø DO
LET A=A-1
LOOP
PRINT A
```

Mit diesem Beispiel wird eine Zeitschleife von ca. 4 Sekunden realisiert.

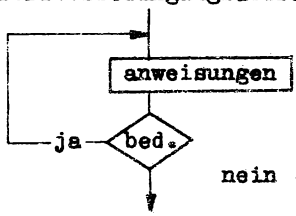
Mit der Weiterentwicklung des Geräts ist eine Programmschleife realisierbar, die die Abbruchbedingung in der Schleife abfragt:

```
BEGIN  LOOP
anweisung 1
IF anweisung DO
anweisung 2
LOOP
```



Damit läßt sich auch die Austrittsbedingungsabfrage am Schleifenende realisieren:

```
BEGIN  LOOP
anweisungen
IF bedingung DO
LOOP
```



3.5. Unbedingter Sprung

Wenn das Programm unbedingt oder in einem bedingten Zweig an anderer Stelle fortgesetzt werden soll, kann die Anweisung (Bsp.)

```
GOTO 1ØØ
```

benutzt werden. Das Programm wird an der mit der Zeilennummer 1ØØ markierten Zeile fortgesetzt. Diese Anweisung wurde bereits im Abschnitt 3.4.3. verwendet. Es gelten die dort ausgeführten Bemerkungen bezüglich strukturierter Programmierung.

3.6. Laufanweisung

Wenn eine Anweisungsfolge mehrmals mit einem laufend verändertem Parameter ausgeführt werden soll (z. B. Ausdrucken von Tabellen, Berechnung von Vektoren mit laufendem Index), kann man die Laufanweisung benutzen.
Es soll z. B. eine Tabelle von Funktionswerten (Wurzel) aller geraden natürlichen Zahlen von 2 bis 1ØØ ausgedruckt werden.

Dazu gehört folgendes Programm:

```
10      FOR A=2 TO 100 STEP 2
        PRINT A,SQR(A)
        NEXT A
```

Die PRINT-Anweisung wird zuerst mit dem Wert 2 für A, dann mit A=4, A=6 usw. bis A=100 ausgeführt. In diesem Beispiel ist A die Laufvariable. Ihr Anfangswert ist 2 und der letzte verwendete Wert ist 100. Die Schrittweite (STEP) ist 2. Wenn als Schrittweite 1 gewählt wird, kann der Ausdruck 'STEP' entfallen.
Anfangs- und Endwert und Schrittweite können Variable oder Ausdrücke sein, vgl. die Syntaxdiagramme.
Als Beispiel soll die Zahlenfolge, die zur Bildung der Zahl e führt (vgl. Bsp. in 3.4.4.), ausgedruckt werden.

```
10      LET A=1
        LET B=1
        FOR E=1 TO 2.718 STEP A
        PRINT E
        LET A=A/B
        LET B=B+1
        NEXT E
```

Die Laufanweisung wird abgebrochen, wenn die Laufvariable größer als der Endwert ist.
Die Schrittweite kann auch negativ sein. Dann wird die Laufanweisung abgebrochen, wenn die Laufvariable kleiner als der Endwert ist. Wenn die Abbruchbedingung von vorherein erfüllt ist (Anfangswert ist größer als Endwert bei positiver Schrittweite), so wird die Laufanweisung ohne Durchlauf sofort verlassen.

3.7. Unterprogrammaufruf

Oft verwendete indentische Programmteile können als Unterprogramme geschrieben werden. Der Aufruf eines Unterprogrammes erfolgt mit GOSUB zeilennummer.

Ein Unterprogramm muß mit RETURN abschließen.
Im Beispiel werden im Unterprogramm vom Bediener 3 Zahlenwerte übernommen, die den Variablen A,B, und C zugeordnet werden. Im Hauptprogramm werden mit diesen Variablen an unterschiedlicher Stelle zwei unterschiedliche Algorithmen ausgeführt.

```
1Ø  GOSUB 3Ø
    LET X=A*B/C
    PRINT X
    .
    .
    .
    GOSUB 3Ø
    LET Y=A/B+C
    PRINT Y
    .
    .
    .
    END
3Ø  PRINT 'A=';
    LET A=INP
    PRINT 'B=';
    LET B=INP
    PRINT 'C=';
    LET C=INP
    RETURN
```

3.8. Kommentaranweisung 'REM'

Nach der Anweisung REM kann in das Programm eine beliebige Textfolge aufgenommen werden, die der Kommentierung dient und nicht abgearbeitet wird. Das unterstützt eine übersichtliche Programmerstellung. Die nächste abzuarbeitende Anweisung muß mit einer Zeilennummer beginnen.

3.9. Programmhalt und Programmunterbrechung

Mit der Anweisung

WAIT

kann der Programmlauf unterbrochen werden, bis eine beliebige Taste betätigt wird. Danach wird das Programm normal fortgesetzt.

Die Anweisung kann benutzt werden, um eine schnelle Folge von PRINT-Anweisungen sichtbar zu machen.

In dem Beispiel des Abschnittes 3.6 zum Ausdrucken der Wurzeln der Zahlen von 2 bis 1ØØ kann besser notiert werden:

```
1Ø   FOR A=2 TO 1ØØ STEP 2
     PRINT A,SQR(A)
     WAIT
     NEXT A
```

Erst so ist die eigentliche Tabelle sichtbar.

Die Anweisung WAIT kann auch benutzt werden, um in einem neu erstelltem Programm zwecks Test an bestimmten Stellen Haltepunkte zu setzen, um den aktuellen Bearbeitungszustand zu erfahren.

Die Anweisung

STOP

dient der Programmunterbrechung, um im Schrittest die Weiterarbeit zu ermöglichen, Variablenwerte zu kontrollieren, zu setzen usw..

3.10. Programmende

Das Ende eines Programmes sollte mit der Anweisung

END

gekennzeichnet werden. Werden im Programm Unterprogramme oder Prozeduren verwendet, so kommt es sonst zu fehlerhafter Abarbeitung, da Unterprogramme oder Prozeduren in unzulässiger Weise erreicht werden. Im einfachen Fall (die bisher verwendeten Beispiele) ist auch ein Abschluß ohne END möglich.

Die Anweisung END kann auch innerhalb eines Programmes mehrmals verwendet werden.
END bewirkt auf dem Display rechts unten die Ausschrift READY. Nach Betätigen einer beliebigen Taste wird der Editormodus wieder angenommen und das Programm kann korrigiert oder neu gestartet werden.

4. Boolsche Ausdrücke und boolsche Wertzuweisung

Boolsche Werte werden repräsentiert von boolschen Variablen, von Vergleichsausdrücken oder von einzelnen arithmetischen Ausdrücken.
Der boolsche Wert einzelner arithmetischer Ausdrücke ist 1, wenn der arithmetische Ausdruck ungleich Null ist. Ist er Null, so ist auch der boolsche Wert Null.

Boolsche Werte können über die logischen Operationen

AND	logische und - Verknüpfung
OR	logische oder - Verknüpfung
XOR	logische exclusiv-oder (antivalenz) - Verknüpfung

zu einem boolschen Ergebniswert verknüpft werden. Dabei ist

XOR gegenüber AND gegenüber OR priorisiert.

Die boolsche Funktion

NEG	logische Negation

wird von einen 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 usw.
Die Vergleichsausdrücke sind gegenüber den boolschen Operatoren priorisiert.
Insgesamt ergibt sich folgende Abarbeitungspriorität:

```
()[]
 * /
+ -
= >= <=  >< <>
NEG
XOR
AND
OR
```

Boolsche Ausdrücke können in Bedingungsabfragen oder in boolschen Wertzuweisungen benutzt werden.
In boolschen Wertzuweisungen muß nach dem Schlüsselwort LET eine boolsche Variable verlangt werden:

```
    BOOLEAN B1
10  LET B1=A > B AND S=Ø
```

Das die boolsche Variable repräsentierende Bit wird dann gesetzt, wenn beide Bedingungen A > B und S=Ø erfüllt sind.
Sonst wird es rückgesetzt.
Alle anderen Bits des entsprechenden Bytes werden nicht verändert.
Boolsche Feldelemente können nicht verarbeitet werden.

5. Standardfunktionen

Folgende Standardfunktionen sind in BASIC-80 verwendbar:

SIN	Sinus
COS	Cosinus
LOG	Natürlicher Logarithmus
EXP	e - Funktion
SQR	Quadratwurzel
INT	Ganzzahlanteil
ABS	Betrag
SGN	Vorzeichen
PI	3.14159

Alle Standardfunktionen außer PI benötigen ein Argument, das nach dem Funktionsnamen in () angegeben wird.
Beispiele:

LET A=B*SIN(T)	a = b . sint
LET A=ABS (D)	a = /d/
LET D=(EXP(X)-EXP(-X))/2	$d = \frac{e^{x} - e^{-x}}{2}$
LET F=2*PI*R*(H+R)	f = 2π . r (h+r)
LET V=4/3*PI*R*R*R	$v = \frac{4}{3}\pi . r^{3}$
LET U1=UØ*EXP(-T/TAU)	$u_1 = u_{\emptyset}\ e^{-\frac{t}{\tau}}$
LET A=SGN(B) * ABS(C)	a = sign b . /c/
LET A=INT(B)	a = intg b
LET DB=2Ø*LOG(U1/U2)/LOG(1Ø)	db=20 lg $\frac{U_1}{U_2} = 20\ \frac{\ln \frac{U_1}{U_2}}{\ln 1\emptyset}$

sign ≙ Vorzeichen
intg ≙ Ganzzahlanteil

Auch in diesen Beispielen ist es sinnvoll, die errechneten Werte auf dem Display zur Anzeige zu bringen.
Folgendes Programm realisiert den Ausdruck der Argumente und Funktionswerte einer Funktion in wählbaren Schritten bis zu einem gewünschten Endwert. Die Funktion sei

$$\text{?} = 4 \cdot \pi \cdot r^{2}$$

(Kugeloberfläche).

Mit jedem Tastendruck (bei längerem Betätigen repitierend) soll ein Wertepaar gedruckt werden. Am Anfang werden der gewünschte Endwert und die Schrittweite verlangt.

```
10  PRINT'       ENDWERT=';
    LET E=INP
    PRINT
    PRINT 'SCHRITTWEITE=';
    LET S=INP
    PRINT
    FOR R=Ø TO E STEP S
    PRINT 'RADIUS=';R,'KUGELVOL.=';4*PI*R*R
    WAIT
    NEXT R
    END
```

Verändern Sie in diesem Programm die Funktion, z. B. Berechnung einer Kreisfläche u. ä.

7. Textverarbeitung

Mit BASIC-80 ist Textverarbeitung möglich. Es gibt Textvariable, die zusätzlich vereinbart werden müssen. Einer einfachen Textvariablen kann nur ein Einzelzeichen zugeordnet werden. Für Zeichenketten sind Textfelder erforderlich.
Im Beispiel wird der Textvariablen TX die Zeichenkette 'HYPOTHENUSE' zugeordnet. In der folgenden PRINT-Anweisung wird diese Textvariable wieder aufgerufen.

```
    STRING TX(12)
10  LET TX='HYPOTHENUSE='
    PRINT TX;SQR(A*A +B*B)
```

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. Der Ausdruck nach dem Gleichheitszeichen ist dann ein Textausdruck. Er hat die gleiche Form wie bei einer PRINT - Anweisung.

Beispiel:

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

Es wird zweimal untereinander das Wort LAUTSPRECHER ausgedruckt, einmal aus der Verknüpfung von W1 und W2 zu W3 und einmal aus der Verknüpfung direkt in der PRINT - Anweisung. Zeilen oder arithmetische Ausdrücke können auch in Zeichenketten gewandelt werden:

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

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

Auswahl von Teiltextketten

In den Beispielen wurden die Textvariablen als geschlossene Zeichenketten (Felder) behandelt. Es ist auch möglich, Einzelzeichen oder Teilzeichenketten zu verarbeiten. Mit einfacher Indizierung wird aus einer Textvariablen ein Einzelzeichen herausgelöst:

```
      STRING TX(1Ø)
1Ø    LET TX='ABCDEFG'
      PRINT TX(3)
```

Es wird das Zeichen D ausgedruckt. Die Zählung erfolgt wie bei Vektoren mit Ø beginnend.

Teilzeichenketten werden mit zwei Indizes bezeichnet. Der erste Index bezeichnet die Stelle, an der die Teilzeichenkette beginnt.
Der zweite Index gibt die Länge der Teilzeichenkette an.
Im obigen Beispiel wird mit

```
PRINT TX(1,4)
```

die Zeichenkette BCDE ausgedruckt.
Es können auch Teilzeichenketten verknüpft werden:

```
      STRING TR(13),TF(5)
1Ø    LET TR='TRANSFORMATOR'
      LET TF=TR(Ø,3);TR(5,2)
      PRINT TF
```

Es wird TRAFO ausgedruckt.

Vergleich von Zeichenketten

In einer IF - Anweisung kann der Vergleich von Zeichenketten als Bedingung benutzt werden. Auf der linken Seite des Vergleichsausdruckes kann ein beliebiger Textausdruck stehen (vgl. Syntaxdiagramme), auf der rechten Seite darf nur eine Textvariable oder eine konstante Zeichenkette stehen. Als Vergleichsoperatoren dienen die gleichen Operatoren wie für den arithmetischen Vergleich. Dabei bedeutet '<'bzw.'>' davorstehend oder nachfolgend in alphabetischer Reihenfolge.

Folgendes Beispiel druckt 8 eingegebene Worte in alphabetischer Reihenfolge aus:

```
      STRING NAME(8Ø),TX(1Ø),TY(1Ø)
1Ø    PRINT FORMAT Ø.Ø;
      FOR A=Ø TO 7
      PRINT 'NAME';A+1;'=';
      LET NAME(1Ø*A,1Ø)=INP
      PRINT
      NEXT A
```

```
20  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
```

Im Sinne der Kompatibilität mit anderen BASIC-Versionen sollten Textvariable mit nachgestelltem $ (als Bestandteil des Namens) gekennzeichnet werden.

Quellennachweis

Dieses interne Material wurde unter Verwendung des Handbuchs zur BASIC 80-Programmierung des VEB Elektronik Gera geschaffen.

Ingenieurschule für Elektrotechnik
"Hanno Günther"
Velten - Hohenschöpping

Schulinternes Lehrmaterial

B A S I C - 80

Für den Mikrocomputer MC 80 des VEB Elektronik Gera
(Teil 2)

Stand: April 1985

8. Programmtest

Die interaktive und interpretative Arbeitsweise von BASIC-80 ermöglicht einen umfassenden Programmtest zur Feststellung der pragmatischen Fehlerfreiheit. Ein Programm kann in Teilen geschrieben und getestet werden. In jedem Programmzweig können zur Kontrolle PRINT- und WAIT-Anweisungen eingefügt werden, die Zwischenergebnisse ausdrucken. Damit kann die Datenentwicklung beobachtet werden.

Die Anweisung STOP und die Kommando cN und cC

Mit der Anweisung STOP kann ein Programm an beliebiger Stelle angehalten werden. Um den Bildschirminhalt im Moment der Abarbeitung der STOP-Anweisung nicht zu zerstören, wird auf dem Display rechts unten nur STOP vermerkt. Erst nach Betätigen einer beliebigen Taste wird der Editormodus eingenommen. Voraussetzung für eine erfolgreiche Weiterarbeit ist, daß das Kommando RUN in dieser Prozedurebene (im allgemeinen im Hauptprogramm) eingegeben wurde.

Voraussetzung: STOP im Programm
RUN Programmstart

Durch einfaches Betätigen von ENTER kann nun wie gewohnt der Eingabemodus angenommen werden. Wenn man als Anweisung

!A

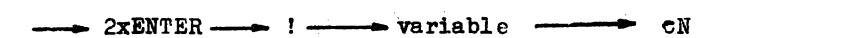
eingibt und nicht die ENTER-Taste, sondern die Taste cN betätigt (control-Taste und Taste N gleichzeitig), so wird diese Anweisung nicht in das Programm eingetragen, sondern sofort ausgeführt. Als Ergebnis der Ausführung erscheint auf dem Display der Wert der Variablen A. Diese Ausführung wird auch als Direktmodus bezeichnet.

Die Anweisung

!variablenname

dient also dem Ausschreiben eines Variablenwertes ähnlich der PRINT-Anweisung.

Syntaxdiagramm:

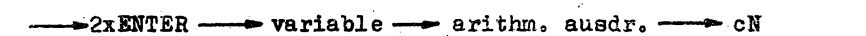


Als weitere Anweisung im Direktmodus läßt sich beispielweise eingeben:

A=5 cN

Damit wird die Anweisung LET a=5 sofort ausgeführt und die Variable erhält den Wert 5.
Auch jede beliebige andere Anweisung läßt sich so ausführen, was aber nicht in jedem Fall sinnvoll ist.

Syntaxdiagramm:

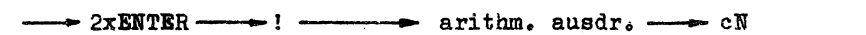


Wird z. B. geschrieben:

!3x4 cN

so wird der Wert 12 ausgedruckt. Damit läßt sich der Direktmodus für Taschenrechnerzwecke nutzen.

Syntaxdiagramm:



Wird jetzt die Taste OFF betätigt, so wird der Anzeigegrundzustand wieder eingenommen. Ein Betätigen von cN ohne Eingabe einer Anweisung löst die Ausführung der im Programm stehenden Anweisungen aus. Damit wird eine Abarbeitung des Programms im Einzelschritt erreicht. Prozeduraufrufe werden im Komplex abgearbeitet.
Gibt man das Kommando

cC

ein, so wird das Programm zur weiteren Abarbeitung an diesem Punkt gestartet, ggf. bis zum nächsten im Programm enthaltenen STOP.
Soll eine Prozedur selbst im Schritt getestet werden, so ist eine STOP-Anweisung in die Prozedur einzufügen. Das RUN-Kommando ist bei Anzeige der Prozedur einzugeben und die Prozedur ist im Hauptprogramm mit den nötigen aktuellen Variablen und Parametern aufzurufen.

9. Felder: Vektoren, Matrizen, Determinanten

BASIC-80 ermöglicht es, Felder zu vereinbaren und zu verarbeiten. Bei der Anfangsinitialisierung und nach NEW werden keine Felder vereinbart. Diese zusätzlichen Vereinbarungen können im lokalen oder globalen Vereinbarungsteil ergänzt werden, wie es dem Abschnitt 10 zu entnehmen ist.
Im Vereinbarungsteil steht nach dem Namen in Klammern die Zahl der Feldelemente. Die Felder werden nur als Vektoren mit einem Index geführt.
Feldelemente werden mit dem Feldnamen, gefolgt von einem in () stehenden Index bezeichnet. Die Zählung des Index beginnt mit Ø. Der Index kann auch von dem Ganzzahlanteil eines arithmetischen Ausdruck repräsentiert werden. Es ist zu beachten, das der Index die im Vereinbarungsteil festgelegte obere Grenze nicht überschreitet. Ansonsten können andere Variablenwerte zerstört werden.

Beispiel: Das Feld FEL wird mit 6 Elementen vereinbart. Diesen Elementen sollen über Eingabe 6 Werte zugeordnet werden.

```
      REAL FEL(6)
1 Ø   FOR A=Ø TO 5
      PRINT 'FEL(';A;')=';
      LET FEL(A)=INP
      PRINT
      NEXT A
      FOR A=Ø TO 5
      PRINT FEL(A)
      NEXT A
      END
```

Dieses Programm besteht aus zwei Laufanweisungen. Die erste ermöglicht die Eingabe der Werte der Feldelemente. Die zweite Laufanweisung druckt die übernommenen Werte auf dem Display zwecks nochmaliger Kontrolle aus.

Bei der Vereinbarung von Feldern wird keine Zeilen- und Spaltordung festgelegt wie es bei Matrizen erfolgt, sondern es gibt nur eine geordnete Folge von Feldelementen, ähnlich der von Vektoren. Durch geeignete Indizierung ist es möglich, Matrizen- und Determinantenrechnung durchzuführen.
Dabei bieten sich Laufaneisungen an.
Beispiel: Spiegeln einer Matrix an der Hauptdiagonalen (Stürzen der Matrix). Dabei werden die Zeilen zu Spalten und umgekehrt:

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}^T \quad \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

Im ersten Programmteil wird die Matrix folgendermaßen belegt und ausgedruckt:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
REAL MAT1(9), MAT2(9)
1Ø FOR A=Ø TO 8
   LET MAT1(A)=A+1
   NEXT A
   CLEAR
   PRINT MAT1(Ø);``;MAT1(1);``;MAT1(2)
   PRINT MAT1(3);``;MAT1(4);``;MAT1(5)
   PRINT MAT1(6);``;MAT1(7);``;MAT1(8)
```

In diesen einfachen Beispiel wird pro Matrixzeile eine PRINT-Anweisung verwendet.
Für größere Matrizen sind Laufanweisungen vorzuziehen.
Das Stürzen der Matrix geschieht nach folgenden Programm:

```
2Ø FOR I=Ø TO 2
   FOR J=Ø TO 2
   LET MAT2(J*3+I)=MAT1(I*3+J)
   NEXT J
   NEXT I
3Ø FOR A=Ø TO 8 STEP 3
   PRINT MAT2(A),MAT2(A+1),MAT2(A+2)
   NEXT A
```

Die Indizierung der Matrix kann auch direkt für die Ausgabe ausgenutzt werden. Deutlich wird das im folgenden Programm:

```
10 FOR J=Ø TO 2
   FOR I=Ø TO 2
   DPL J, 3xI FORMAT !2.Ø MAT2(Jx3+I)
   NEXT I
   NEXT J
   END
```

Mit dieser Indizierung lassen sich einfach Unterdeterminanten für die weitere Berechnung bilden.

10. Vereinbarungsteil

Nach dem Kommando NEW bzw. nach Neuanwahl des BASIC-Programmes erfolgt eine Grundinitialisierung. Damit können einfache Programme (die verwendeten Beispiele) sofort notiert werden. Der globale Vereinbarungsteil steht zu Anfang des gesamten Programmsystem.
Der lokale Vereinbarungsteil gilt nur für das Hauptprogramm oder der jeweiligen Prozedur.

10.1. Lokaler Vereinbarungsteil, Variablentypen

Das Hauptprogramm und jede Prozedur beginnt mit der Anweisung

```
DEF  name
```

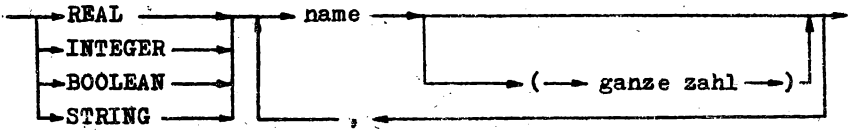
Dabei ist "name" eine beliebige Zeichenfolge. Bei Prozeduren wird nach dem Namen eine Liste formaler Variablen und Parameter in Klammern angegeben (vgl. Abschnitt 12.).
Die auf DEF folgenden Anweisungen gehören zum lokalen Vereinbarungsteil.
Der lokale Vereinbarungsteil des Hauptprogramms enthält nach der Grundinitialisierung die REAL-Vereinbarungen der Buchstaben A ... Z. Das bedeutet, daß jedem Buchstaben eine Gleitkommazahl (real) zugeordnet werden kann.

Im Abschnitt 9 wurde der lokale Vereinbarungsteil mit Feldvereinbarungen ergänzt.

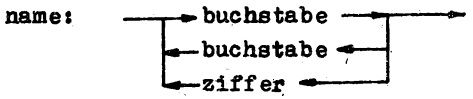
Um zusätzliche REAL-Variable mit eigenen Bezeichnungen zu erhalten, kann der lokale Vereinbarungteil z.B. folgendermaßen erweitert werden:

```
REAL U1,UØ,TAU
REAL U2
```

Die Anweisungen zum lokalen Vereinbarungsteil haben folgende Form:



Der Begriff "name" muß dabei noch definiert werden:



Der Begriff "buchstabe" kann ein Zeichen A ... Z haben und müßte im strengen Sinne hier auch definiert werden. Jedoch ist diese Definition mit Worten umschreibbar und allgemein bekannt und soll deshalb hier entfallen. Eine "ziffer" ist ein Zeichen Ø ... 9.
Ein "name" muß mit einem Buchstaben beginnen, gefolgt von einer beliebigen Anzahl von Buchstaben oder Ziffern.
Die Begriffe REAL, INTEGER, BOOLEAN und STRING bestimmen den Variablentyp. Die Variablentypen haben folgende Eigenschaften:

REAL

REAL-Variable sind Gleitkommazahlen. Sie haben einen Zahlenbereich von $-10^{38}...-10^{-38}$; Ø ; $10^{-38}...10^{38}$.
Die Verarbeitungsbreite beträgt 7 Dezimalstellen.

Beispiel:	Darstellung im Format Ø.4
1,234	1.234
Ø,125	125E-Ø3
$-3x1Ø^{2Ø}$	-3ØØE+18
Ø,1	Ø.Ø99

Eine REAL-Variable oder ein REAL-Feldelement belegt 4 byte im Speicher. Intern wird eine REAL-Variable in der binären Gleitkommadarstellung mit 24 bit Mantisse und 8 bit Exponent dargestellt. Das obere Mantissenbit ist immer 1. An dieser Stelle wird das Vorzeichen der Gesamtzahl vermerkt. Zum Exponenten im Zweierkomplementformat wird die Konstante 8Ø H addiert. Diese Zahlendarstellung ist in der Beschreibung der Gleitkommaarithmetik EPAS-80 beschrieben. Da in der binären Zahlendarstellung manche runde Dezimalbrüche unendlich periodisch sind, können sie in der dezimalen Darstellung nicht exakt wiedergegeben werden. Insbesondere trifft das auf die Zahlen Ø,1 ; Ø,2 usw, zu. Es wird mit abgerundeten Werten gearbeitet und diese werden auch dargestellt. In einer Laufanweisung

FOR A=1 TO 1ØØ STEP Ø.1

wird z. Bsp. nicht genau der Wert 1ØØ erreicht. Ist dies erforderlich, so ist günstiger zu programmieren:

FOR A=1Ø TO 1ØØØ STEP 1

Die Laufvariable kann intern dann mit der Bewichtung A/1Ø benutzt werden. Verbleibt man im Bereich der ganzen Zahlen, so treten keine Rundungsfehler auf. Wenn die letzte Ganzzahlstelle (Wertigkeit 1) noch aufgelöst werden soll, kann man maximal die Zahl

8 388 6Ø7

verarbeiten. Das entspricht der internen Auflösung mit 24 bit.

INTEGER

INTEGER-Variablen sind Festkommazahlen. Sie umfassen einen Zahlenbereich von -32 768 ... 32 767.
Intern werden sie mit 2 Byte im Zweierkomplementformat abgespeichert. Alle Berechnungen innerhalb von BASIC-8Ø werden im Gleitkommaformat (REAL) ausgeführt. INTEGER-Zahlen werden deshalb vorher in REAL-Zahlen gewandelt. Das INTEGER-Format kann bei lokalen Variablen sinnvoll angewendet werden, wenn bei großen Feldern Speicherplatz gespart werden soll.
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.

BOOLEAN

BOOLEAN-Variablen sind Einzelbits mit dem Wertevorrat

Ø ; 1

Im Speicher wird auch nur ein Bit belegt, Auf ein Byte passen 8 BOOLEAN-Variablen. Es müssen immer 8 BOOLEAN-Variablen im Block vereinbart werden. In der Reihenfolge der Variablen wird zuerst das LBS belegt. Mittels BOOLEAN-Variablen kann insbesondere eine Einzelbitverknüpfung im RAM ausgeführt werden.

STRING

STRING-Variable können Zeichen im ASCII-Code speichern. Sie belegen als Einzelvariable 1 Byte im Speicher.
Wie im Abschnitt 9 dargestellt ist, sind im allgemeinen STRING-Felder notwendig, die dann entsprechend der Anzahl der Feldelemente Bytes benötigen.
Neue Vereinbarungen von lokalen Variablen dürfen nur nach den bereits getroffenen Vereinbarungen ergänzt werden. Das Streichen bereits verwendeter Vereinbarungen führt zu Anzeigefehlern.

10.2. Globaler Vereinbarungsteil

Im globalen Vereinbarungsteil werden globale Variable, Assemblerfunktionen, Assemblerausgabefunktionen und Assembleranweisungen declariert. Globale Variable werden vom Programm ebenso wie lokale Variable verarbeitet.
Ihre Speicheradresse wird vom Anwender festgelegt. Damit kann der Zugriff auch über ein Assemblerprogramm erfolgen.
Ein Block im globalen Vereinbarungsteil beginnt mit:

```
GLOBAL %EØØØ
```

EØØØ ist eine hexadezimale Speicheradresse, auf die sich alle weiteren Eintragungen beziehen. Die Typenbezeichnung der globalen Variablen erfolgt analog denen der lokalen Variablen mit nachgestellten # .

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

Die in diesen Block befindlichen Variablen werden ab der Speicheradresse EØØØ angelegt. Im Einzelnen haben die Variablen folgende hexadezimalen Adressen:

A1	EØØØ		4 Byte
VE	EØØ4		45 x 4 Byte = 18Ø Byte = B4H
B1	EØB8	Bit Ø	1 bit
B2	EØB8	Bit 1	
B3	EØB8	Bit 2 ... 7	
TEXT	EØB9	... EØFE	7Ø Byte

Ein nächster Vereinbarungsblock könnte z. B. beginnen:

```
GLOBAL %E8ØØ
INTEGER# F1, F2
```

Damit werden die INTEGER-Variablen F1 und F2 auf den Adressen E8ØØ bzw. E8Ø2 (á2 Byte) abgespeichert.

11. Softwareschnittstellen

11.1. Assembleranweisungen

Eine Assembleranweisung ist ein Unterprogramm in Assemblersprache ohne zusätzliche Schnittstellen. In BASIC wird eine Assembleranweisung wie folgt verwendet:

```
REGLER
```

Vereinbart wird diese Assembleranweisung:

```
GLOBAL %EE12
DCL REGLER
```

Wenn die Anweisung REGLER erreicht wird, wird ein Assemblerprogramm auf der Adresse EE12H gestartet.
Die Einbindung von Assemblerprogrammen in BASIC-8Ø ermöglichen die Ausnutzung des gesamten Befehlssatzes des Prozessors U88Ø und damit eine spezifische hardwarenahe Programmierung.

INPUT: DE = Zeiger auf die nächste Position des Zwischencodes BASIC, die abgearbeitet werden soll
IY = arithmetischer Stackpointer, arbeitet decrementierend
IX = Zeiger auf lokale Variable (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-Programms

11.2. Assemblerfunktionen

Eine Assemblerfunktion ist ein Unterprogramm in Assemblersprache (U88Ø), welche einen Wert als Ergebnis liefert. Assemblerfunktionen werden ähnlich wie Standardfunktionen verwendet. Sie können ohne Argument oder mit mehreren Argumenten auftreten. XE sei z. Bsp. eine Assemblerfunktion ohne Argument, die einen Analog-Digital-Wandler abfragt und seinen 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 %ECØØ
INTEGER! XE
```

Ab ECØØ beginnt dieses Assemblerprogramm, das als Unterprogramm formuliert das Ergebnis im INTEGER-(Festkomma-) Format im Registerpaar HL liefert.
Es gelten die gleichen Schnittstellen wie für die Assembleranweisungen. Zusätzlich müssen folgende OUTPUT-Parameter realisiert werden:

OUTPUT für	
REAL!	BCHL = Gleitkommazahl
INTEGER!	HL = Festkommazahl
BOOLEAN!	A = ØØ oder FF

Für die STRING-Funktion werden INPUT Parameter übergeben:

INPUT für STRING! :

HL = Adresse des Textpuffers

(IY)= Verfügbare Länge des Textpuffers

(IY+1) = Formatbyte

OUTPUT für STRING! :
Im Assemblerprogramm kann der Textpuffer nachfolgend mit beliebigen Zeichen in der ASCII-Codierung gefüllt werden. Dabei muß (IY) beim Einschreiben jedes Zeichens decrementiert werden. (IY) darf den Wert 1 nicht unterschreiten! HL muß bei Verlassen des Programms das nachfolgende Byte zeigern.

11.3. Assemblerausgabefunktionen

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

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

Das Ergebnis des arithmetischen Ausdruckes wird der Ausgabefunktion in den CPU-Registern übergeben. Vereinbart werden Assemblerausgabefunktionen genauso wie Assemblerfunktionen mit nachgestellten '?' (anstatt !).

```
GLOBAL %EEØF
INTEGER? VEN
```

Assemblerausgabefunktionen können sich nur auf der linken Seite einer LET-Anweisung befinden.
Es gelten die gleichen Schnittstellen wie für die Assembleranweisungen. Zusätzlich gelten folgende INPUT-Parameter:
INPUT für

REAL? : BCHL = Gleitkommazahl als Ergebnis des arithmetischen Ausdruckes der LET-Anweisung

INTEGER? : HL = Festkommazahl als Ergebnis des arithmetischen Ausdruckes der LET-Anweisung

BOOLEAN? : A = ØØ oder FF als Ergebnis des boolschen Ausdruckes der LET-Anweisung

STRING? : In dem STRING-Unterprogramm muß mit den INPUT-Parametern HL=Textpufferadresse und C=Textpufferlänge das im BASIC-Interpreter enthaltene Programm TTX aufgerufen werden!

Übersicht:

Einbinden Assemblerprogramme in BASIC 8Ø

Ass-funktionen	Ass-ausgabefunktionen	Ass-anweisungen
-UP in Assemblersprache	-UP in Assemblersprache	-UP in Assemblersprache ohne zusätzlichen Schnittstellen
-liefert einen Wert als Ergebn.	-benötigt einen Wert	
- wird wie eine Standarddfkt. benutzt	-BASIC-Ergebnis wird in Schnittstellen übergeben	
-Variable steht immer rechts in einer LET-Anweisung	-Variable steht immer links in einer LET-Anweisung	

11.4. BASIC-Programmaufruf aus Assemblerprogrammen

BASIC-Programme können als eigenständige Tasks inner- und außerhalb des BASIC-Editors verwendet werden. Mit dem Kommando PUT kann ein geschriebenes BASIC-Programm auf einen Pufferspeicher ausgelagert und dort als Assembleranweisung in BASIC aufgerufen werden. Dem mit PUT ausgelagertem Programm muß dann übergeben werden.

INPUT: IY = arithmetischer Stackpointer

Ein genügend großer RAM-Bereich mit IY als obere Grenze muß belegbar sein. Größe des RAM-Bereiches richtet sich nach der Zahl der lokalen Variablen, nach den aufgerufenen Prozeduren und der Schachtelungstiefe von Klammern.

DE - Zeiger auf:

BASIC-Zwischencode mit aktuellen Variablen und Parametern, wenn eine Liste formaler Variablen und Parameter Bestandteil des abzuarbeitenden BASIC-Programmes ist.

Das Auslagern des BASIC-Programms erfolgt mit PUT bzw. PUTD wie es in BASIC 8Ø Teil 1 beschrieben ist. Über den globalen Vereinbarungsteil wird die Anfangsadresse declariert.

```
GLOBAL %E000
DCL TASK
```

Aufgerufen wird dieser TASK aus dem Assemblerprogramm mit dem UP-Aufruf 'CALL'.

Dritte Möglichkeit ist, ein BASIC-Programm aufzurufen, welches sich innerhalb des BASIC-Editors (CØØØ H - EØØØ H) befindet. Der Aufruf erfolgt dabei mit der absoluten Adressierung zum Speicherplatz der mit ØB2 belegt ist.

CALL >B2<

Damit wird der Interpreter aufgerufen. Das BASIC-Programm muß mit END bzw. SUBEND abgeschlossen sein. IY ist wie oben beschrieben der Zeiger auf den arithmetischen Stack.

Übersicht:

Verwendung von BASIC-Programmen

außerhalb des BASIC-Editors		innerhalb des BASIC-Editors
Assembleranweisung	Prozedur I	Prozedur II
-BASIC-Progr. mit PUT auslagern Puffer	-BASIC-Progr.mit PUT auslagern in Puffer	-BASIC-Progr. nicht auslagern
-Aufruf von einem BASIC-Programm	-Aufruf von einem Ass.-Programm	-Aufruf von einem Ass.-Programm
-dazu global vereinbaren	-DE-Zeiger auf ZC IY-Zeiger auf SP von BASIC-Progr.	-IY-Zeiger auf SP von BASIC-Programm
	-Aufruf:	-CALL>B22 Adresse wo im RAM
	LD IY, SP	B2 steht
	CALL Anf-adr. des ausgelagerten Progr.	-Aufruf: LD IY, SP CALL>B2<

12. Prozeduren

Im Abschnitt 3.7. wurde bereits auf die Verwendung von Unterprogrammen hingewiesen, wenn eine gleiche Befehlsfolge mehrmals im Programm auftritt. Eine weitere Möglichkeit stellt dabei die Anwendung einer Prozedur dar. Gegenüber Unterprogrammen haben die Prozeduren Vorteile bei der Parameterübergabe. In einem Programm tritt z. Bsp. die Multiplikation von Matrizen mehrmals auf.

Ein Unterprogramm (GOSUB) kann man nur für Matrizen, die bei der Programmierung festgelegt werden, schreiben.
Sollen verschiedene Matrizen multipliziert werden, so sind die Werte dieser Matrizen zuvor auf jene Matrizen, mit denen im Unterprogramm gearbeitet wird, zu laden. Die Werte der Ergebnismatrix des Unterprogramms müssen wiederum auf die gewünschte Ergebnismatrix umgeladen werden. Wenn ein gleicher Algorithmus mit verschiedenen Variablen und Parameter ausgeführt werden soll, ist bei Verwendung eine Unterprogramms also jedesmal ein Umladen der Werte notwendig.
Die Prozeduren arbeiten nach außen hin mit formalen Variablen und Parametern. Diesen formalen Variablen und Parametern werden beim Aufruf die aktuellen Adressen und Werte zugewiesen. Im Prozedurkopf steht nach DEF der Prozedurname. Danach folgt in ( ) die Liste der formalen Variablen und Parameter. Dem Prozedurkopf darf keine Zeilennummer vorangestellt werden!
Zur Kennzeichnung formaler Variablen wird verwendet:

~~#~~	formale REAL - Variable
%	formale INTEGER - Variable
&	formale BOOLEAN - Variable
§	formale STRING - Variable

Diese Sonderzeichen sind dem Namen vorangestellt und gehören selbst nicht mit zur Variablenzeichnung. Die formalen Variablen werden in der formalen Liste des Prozedurkopfes nicht als Vektoren geführt. Nach der formalen Variablen mit den vorangestellten Vorsatzzeichen können in der formalen Liste weitere Variablenbezeichnungen folgen. Diese Variable sind immer REAL-Variable. Ihnen wird beim Prozeduraufruf in der aktuellen Liste einen Wert zugeordnet, der von einem beliebigen arithmetischen Ausdruck repräsentiert werden kann. Diese Variable werden formale Parameter genannt.
Die formalen Parameter existieren nur in der Prozedur. Ihnen wird beim Aufruf ein aktueller Wert zugewiesen. Diese Aufrufart wird `call by value`genannt. Im Gegensatz dazu werden den formalen Variablen aktuelle Variablen zugewiesen, die gelesen und beschrieben werden können.

Diese Aufrufart heißt 'call by reference'.
Über die formalen Variablen kann die Prozedur dem aufrufenden Programm Ergebnisse übermitteln. Das ist sonst nicht möglich. Die formale Liste des Prozedurkopfes muß mit der aktuellen Liste des Prozeduraufrufes in Anordnung und Art der Variablen und Parameter genau übereinstimmen. Ansonsten liegt ein Syntaxfehler bei der Abarbeitung vor. Dem Prozedurkopf muß ein lokaler Vereinbarungsteil folgen. In ihm werden lokale Variablen definiert, die nur in dieser Prozedur gültig sind.
Außer auf die in der Prozedur selbst definierten Variable kann in der Prozedur noch auf die globalen Variablen zurückgegriffen werden, die am Anfang des Programmsystems definiert wurden. Auf die lokalen Variablen des aufrufenden Programms kann nur indirekt oder über die formalen Variablen zugegriffen werden. Eine direkte Zugriffmöglichkeit besteht nicht. Prozeduren sind beliebig schachtelbar. In jeder Prozedur kann wiederum ein Prozeduraufruf, auch der eigenen Prozedur erfolgen.
Da bei jedem Prozeduraufruf ein weiterer lokaler Variablenbereich im arithmetischen Stack belegt wird, sind Prozeduren prinzipiell wiedereintrittsfähig (mehrfach geschachtelt aufrufbar, ohne Daten zu zerstören). Bei Verlassen der Prozedur wird der belegte Stackbereich wieder freigemacht. Damit verschwinden alle lokalen Variablenzuweisungen.

Funktionen

Als Prozedur geschriebene Funktionen sind ein Spezialfall einer Prozedur. Die Funktion übermittelt direkt einen Ergebniswert. Dazu muß am Ende der Funktion vor dem **RETURN** notiert werden.

FN = arithmetischer Ausdruck

Übersicht:

	Prozedur	Funktion
Organisation	DEF name(form.parameterliste)	DEF FNname(form. meterliste)
	REAL	REAL
Aufruf	CALL name(parameterliste)	LET D=FNname(parameterliste)
Wertzuweisung	(-den formalen Variablen)	FN=arithmet. Ausdruck
Rücksprung	RETURN	RETURN
	SUBEND	FNEND

Parameterlisten

Aufruf: CALL name(A, B, C, 5, 100)
Organisation: DEF name (#X,%Y,#Z, K, P)

13. Fehlerausschriften

Syntaxfehler bei der Eingabe:

Nach der Korrektur oder Neueingabe einer Anweisung und Betätigen von 'ENTER' erfolgt die Übersetzung der Zeile. Dabei wird ein Teilsyntaxtest durchgeführt. Jedes einzelne Wort wird in der Übersetzungstabelle und im Vereinbarungsteil gesucht. Wird es nicht gefunden, so wird der Kursor unter die unbekannte Zeichenfolge gesetzt. Es kann sofort korrigiert werden. Danach wird wieder 'ENTER' betätigt. Eine Überprüfung der Reihenfolge der Einzelfolge erfolgt bei der Eingabe nicht. Um den gesamten Syntaxtest des Programms auszuführen, ist die Abarbeitung des Gesamtprogramms notwendig.

≫ Syntax

Diese Ausschrift erfolgt nach RUN, wenn in einem Programm die Anordnung der Einzelworte einer Anweisung fehlerhaft ist und diese Anweisung abgearbeitet wird.

Die Programmabarbeitung wird unterbrochen, es kann nur mittels RUN neu gestartet werden. Die fehlerhafte Stelle wird im Anzeigegrundzustand mit ≫ gekennzeichnet.
Es ist zu beachten, daß bei der Abarbeitung z.T. nicht die eigentliche Fehlerstelle, sondern ein Folgefehler erkannt wird. Der eigentliche Fehler kann sich auch davor befinden.
Nach Erscheinen dieser Fehleranzeige muß die Taste 'OFF' betätigt werden. Danach kann die betreffende Anweisung korrigiert werden (Überschreiben, ENTER).
Das Programm muß mit RUN neu gestartet werden.

Zuweisung

Diese Ausschrift erfolgt nach RUN, wenn in einem Programm die Anweisungsklammern fehlerhaft gesetzt sind.
Mit Schachtelungsmöglichkeiten muß folgen auf:
IF ... DO .... ELSEDO ... DOEND
IF ... DO .... DOEND
IF ... THEN....Zeilennummer mit Anweisung
FOR...TO .... NEXT
WHILE ... DO .... LOOP
GOTO Zeilennummer .... Zeilennummer
Fehlt ein Anweisungsklammerabschluß (DOEND, LOOP, NEXT, Zeilennummer), so erfolgt die Fehleranzeige am Programmende bzw. bei einer END-Anweisung. Ist ein Anweisungsklammerabschluß zuviel vorhanden, so wird dieser als fehlerhaft angezeigt. Nach Fehlerkorrektur und erneutem Programmstart mit RUN, kann ein weiterer Fehler dieser Art angezeigt werden usw. Erst nach Korrektur aller Anweisungsklammerfehler wird das Programm nach RUN gestartet.

Speicher Voll

Diese Ausschrift erfolgt, wenn eine Anweisung eingegeben werden soll und der Programmspeicherbereich würde dabei überfüllt werden.
Der vom BASIC-Programm insgesamt verfügbare Speicherbereich wird zu Beginn beim Aufruf (Abschnitt 3.1., Anwahl) angegeben. Bei der Programmabarbeitung wird

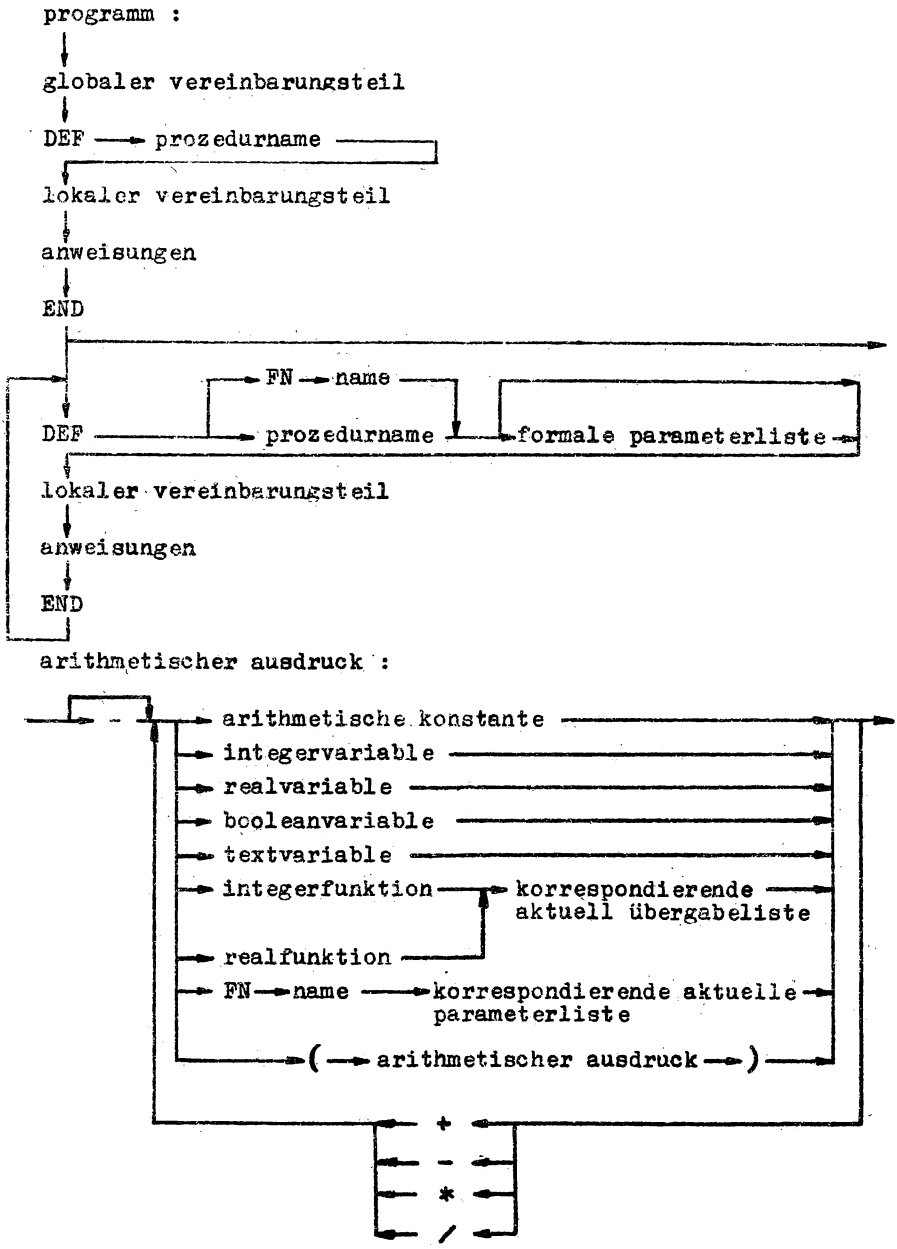
von oben (von der Endadresse aus rückwärts) der Variablenspeicher in Form eines arithmetischen Stacks angelegt.
Von unten her wird das Programm eingeschrieben. Eine Fehleranzeige des Programmspeicherüberlaufs erfolgt, ein möglicher Überlauf des Variablenspeichers wird nicht kontrolliert.
Das ist bei umfangreichen Programmen mit mehreren geschachtelten Prozeduraufrufen zu beachten.

DEF-EBENE

Diese Fehleranzeige erfolgt beim Auslösen der Kommandos cN oder cC, wenn das Programm nicht im Lauf unterbrochen wurde oder eine falsche Prozedurebene angewählt wird.

14. Syntaxdiagramme

In den Syntaxdiagrammen wird die Zulässigkeit der Kombinationen von Bezeichnungen definiert. Sie erklären nicht die Wirkung der Bezeichnungskombinationen im Programm.
Feste Bezeichnungen, die genauso geschrieben werden müssen, werden mit Großbuchstaben bzw. den entsprechenden Sonderzeichen dargestellt. Bezeichnungen in Kleinbuchstaben stehen für einen Begriff, der in einem weiteren Syntaxdiagramm definiert wird. Dabei kann dieser Begriff selbst wieder in der Definition enthalten sein.
Unterstrichene Bezeichnungen in Kleinbuchstaben sind Namen, die hier vereinbart werden. Diese Namen beginnen mit einem Buchstaben A ... Z und enthalten danach eine beliebige Folge von Ziffern und Buchstaben.
Die Bezeichnungen können in Pfeilrichtung kombiniert werden.
Die Syntaxdiagramme sind von oben nach unten, von der Gesamtstruktur zum Detail gegliedert.


programm :
globaler vereinbarungsteil
DEF
prozedurname
lokaler vereinbarungsteil
anweisungen
END
DEF
FN
name
prozedurname
formale parameterliste
lokaler vereinbarungsteil
anweisungen
END
arithmetischer ausdruck :
-
arithmetische konstante
integervariable
realvariable
booleanvariable
textvariable
integerfunktion
realfunktion
korrespondierende aktuell übergabeliste
FN
name
korrespondierende aktuelle parameterliste
(
arithmetischer ausdruck
)
+
-
*
/