SUPERBASIC
FÜR
DEN MZ-80B

BEDIENUNGSANLEITUNG

Wir behalten uns vor, jederzeit und ohne Vorankündigung Änderungen an diesem Programm vorzunehmen, die seinem Optimierung dienen. In solchen Fällen wird die Programmdokumentation auf den neuesten Stand gebracht. Hierbei kann trotz aller Sorgfalt keine vollständige funktionelle Übereinstimmung gewährleistet werden.

INHALTSVERZEICHNIS

1. EINLEITUNG 4

2. KONVENTIONEN 5

3. DIE TASTATUR 6
3.1. Grundfunktionen 6
3.2. Kontrollfunktionen 7
3.3. Die Verwendung der Kontroll-Codes im Programm 8

4. DIE FUNKTIONSTASTEN [F1] BIS F[10] 9
4.1. Die Neubelegung der Funktionstasten 9

5. DIE VARIABLENTYPEN..... 10
5.1. Variablentyp STRING 10
5.2. Variablentyp INTEGER 10
5.3. Variablentyp SINGLE PRECISION 10
5.4. Variablentyp DOUBLE PRECISION 11
5.5. Die globale Variablendeklaration 11
5.6. Änderung von Variablentypen 11
5.7. Oktale und hexadezimale Zahlendarstellung 11

6. SUPERBASIC-BEFEHLE 12

7. STANDARDANWEISUNGEN 16

8. ANWEISUNGEN ZUR EIN- UND AUSGABE 23
8.1. Die PRINT-Anweisung 30
8.2. Die PRINT USING-Anweisung 31

9. DIE MATHEMATISCHEN FUNKTIONEN 34

10. DIE MATHEMATISCHEN UND LOGISCHEN OPERATOREN 38
10.1. Mathematische Operatoren 38
10.2. Vergleichsoperatoren 39
10.3. Logische Operatoren 41

11. DIE STRING-FUNKTIONEN 42

12. SONDERFUNKTIONEN 46

13. SCHLEIFENKONSTRUKTIONEN 52

14. FEHLERMELDUNGEN 54

15. DIE FEHLERVERARBEITUNG 57

16. BEFEHLSABKÜRZUNGEN 59

17. KASSETTENHANDLING 61

18. DIE GRAFIK-ANWEISUNGEN 64

19. INDEX 71

1. EINLEITUNG

SuperBASIC ist ein besonders schneller und leistungsfähiger BASIC-Interpreter, der in seinen Editierfunktionen einen bisher unbekannten Komfort bietet und in seinem Befehlsumfang sogar die CP/M BASIC-Version von Microsoft übertrifft.

Die Computertastatur wurde mit vielen Zusatzfunktionen ausgestattet. Unter anderem bietet sie nun "n-key rollover", 30 dedizierte Kontroll-Funktionen für die erweiterte Cursor-, Bildschirm- und Textsteuerung sowie 10 programmierbare Funktionstasten.

SuperBASIC-Variablen können als Strings, ganzahlig, mit einfacher oder mit doppelter Rechengenauigkeit deklariert werden. Eine globale Variablendeklaration a la Pascal ist möglich. Zahlen werden im Bereich von -10E38 bis +10E38 verarbeitet, und multidimensionale Felder können vereinbart werden.

Programmzeilen, die bis zu 240 Zeichen enthalten dürfen, können in Klein- oder Großschrift eingegeben werden. SuperBASIC sorgt dafür, daß alle Kleinbuchstaben, die nicht Teil eines Strings sind, automatisch in Großbuchstaben verwandelt werden.

Zu den weiteren Besonderheiten dieser BASIC-Implementierung gehören nützliche Elemente aus anderen Hochsprachen wie, z.B., REPEAT ... UNTIL und WHILE ... WEND, die Möglichkeit mit dem SCRN$-Befehl, Benutzereingaben direkt vom Bildschirm zu lesen, ohne daß eine GET- oder INPUT-Anweisung vorliegt, und die Fähigkeit Unterprogramme wie Prozeduren zu behandeln.

SuperBASIC bietet auch eine Reihe von Hilfsmitteln, die die Programmerstellung und Fehlersuche wesentlich erleichtern. Die TRACE-Funktion enthält sechs verschiedene Optionen für die Programmanalyse. Der integrierter Submonitor verfügt über eine große Befehlspalette für die Eingabe und Analyse von Maschinenprogrammen.

Dem MZ-80B-Besitzer bietet SuperBASIC ungeahnte Möglichkeiten für die Erstellung von Computergrafiken. Die SHARP-BASIC-Grafikebefehle wurden durch Anweisungen wie PAINT, SCROLL und CIRCLE erweitert und die Ausführungsgeschwindigkeit sämtlicher Grafikanweisungen wesentlich beschleunigt.

2. KONVENTIONEN

In dieser Bedienungsanleitung werden folgende Konventionen bei der Befehlserläuterung getroffen:

.....	Platzhalter für beliebige Ausdrücke
[n]	Tastensymbol
↑[n]	Kontrollfunktion mit [SFTLOCK] + Taste [n]
[Zahl]	Platzhalter für eine einzusetzende Zahl
[String]	Platzhalter für einen einzusetzenden String
[Variable]	Platzhalter für eine einzusetzende Zahlen- oder String-Variable
[Zahlenvariable]	Platzhalter für eine numerische Variable.
[Stringvariable]	Platzhalter für eine Stringvariable
(A)	Eine beliebige Zahlenvariable
(A$)	Eine beliebige Stringvariable
(aaaa)	Eine beliebige Zahlenfolge im zulässigen Bereich
▬ ▬ ▬	Zwingend vorgeschriebene Leerzeichen

3. DIE TASTATUR

3.1. Grundfunktionen

Im normalen und [RVS]-Modus, ohne daß [SHIFT] oder [SFTLOCK] betätigt wurden, erzeugen die alphanumerischen Tasten Großbuchstaben, Zahlen oder die entsprechenden Zeichen bzw. Funktionen.

Mit [SHIFT] werden Kleinbuchstaben und die entsprechenden Zeichen bzw. Funktionen der zweiten Tastatur-Ebene erzeugt.

Mit [SFTLOCK] erzeugen die Alpha-Tasten Kleinbuchstaben. Alle anderen Tasten bleiben hiervon unberührt. Die [SFTLOCK]-Funktion wird durch gleichzeitiges Drücken von [SHIFT] und [SFTLOCK] ein- und ausgeschaltet.

Im [RVS]-Modus, der mit [SFTLOCK] und [RVS] ein- und ausgeschaltet wird, werden die Buchstaben und Zeichen in negativer Darstellung erzeugt.

Das Ein- und Auschalten der Grafikebene geschieht durch Drücken der [SFTLOCK]- und [GRPH]-Tasten.

Alle Tasten außer den Funktionstasten [F1] bis [F10], den Modus-Tasten und dem Zehnerblock haben eine Repetierfunktion.

Die Tastatur ist mit einem "n key rollover" ausgestattet. Die Zeichen werden gepuffert, bevor sie auf dem Bildschirm erscheinen, wodurch ein besonders schnelles und flüssiges Schreiben ermöglicht wird.

Die Cursorsteuerung hat bereits ohne [SHIFT] eine Repetierfunktion. Mit [SHIFT] werden folgende Funktionen ausgelöst:

[SHIFT] + [Rechtspfeil] - Cursorsprung zum ersten Zeichen nach dem nächsten Trennzeichen (■ oder Interpunktionszeichen)

[SHIFT] + [Linkspfeil] - Cursorsprung rückwärts bis zum Zeichen vor dem letzten Trennzeichen

[SHIFT] + [Hochpfeil] - Verschiebt den Text ab Cursorposition nach unten und erzeugt Leerzeilen

[SHIFT] + [Tiefpfeil] - Verschiebt den Text ab Cursorposition nach oben und erzeugt Leerzeilen

Wird die [INST]-Taste über das Zeilenende hinaus betätigt, wird ab Cursorposition eine Leerzeile geöffnet.

Bei der [DEL]-Funktion wird die ganze logische Zeile nachgezogen, auch wenn sie sich über mehrere Bildschirmzeilen erstreckt.

3.2. Die Kontroll-Funktionen

Bei SuperBASIC wird die [SFTLOCK]-Taste als Kontrolltaste verwendet. In Verbindung mit anderen Tasten löst sie eine Reihe Sonderfunktionen aus:

↑[A] - Verbindet die aktuelle logische Zeile mit der Darunterstehenden

↑[B] - Bewegt den Cursor rückwärts bis zum Zeichen vor dem letzen Trennzeichen. Entspricht [SHIFT] + [Linkspfeil]

↑[C] - Bewegt den Cursor zum Anfang der nächsten logischen Zeile. Entspricht [SHIFT] + [BREAK] (siehe unten)

↑[D] - Keine Wirkung in dieser Implementierung von SuperBASIC.

↑[E] - Löscht die logische Zeile ab Cursorposition

↑[F] - Bewegt den Cursor auf das erste Zeichen nach dem nächsten Trennzeichen. Entspricht [SHIFT] + [Rechtspfeil]

↑[G] - Schaltet den Tasten-Knackton aus bzw. ein

↑[H] - Entspricht [DEL]

↑[I] - Löscht die Zeile bis zur nächsten Tabulatorposition

↑[J] - öffnet eine Leerzeile vor der Cursorposition

↑[K] - Entspricht [HOME]

↑[L] - Entspricht [CLR]

↑[M] - Entspricht [CR]

↑[N] - Verschiebt den Text ab Cursorposition nach oben. Entspricht [SHIFT] + [Tiefpfeil]

↑[O] - Verschiebt den Text ab Cursorposition nach unten. Entspricht [SHIFT] + [Hochpfeil]

↑[P] - Gibt den Bildschirminhalt auf dem Drucker aus

↑[Q] - Schaltet die Repetierfunktion ein und aus

↑[R] - Entspricht [INST]

↑[S] - Stop-Taste. Unterbricht die Programmausführung. Das Programm läuft nach dem nächsten Tastendruck weiter.

↑[T] - Setzt zusätzliche Tabulatorpositionen

↑[U] - Bewegt den Cursor zur nächsten Tabulatorposition. Entspricht [TAB]

↑[V] - Schaltet den [RVS]-Modus ein und aus. Entspricht [SFTLOCK] + [RVS]

↑[W] - Schaltet den [GRPH]-Modus ein und aus. Entspricht [SFTLOCK] + [GRPH]

↑[X] - Schaltet den Alpha-Lock ein und aus. Entspricht [SHIFT] + [SFTLOCK]

↑[Y] - Löscht Tabulatorpositionen

↑[Z] - Löscht den Bildschirm ab Cursorposition

↑[[] - Cursor nach oben

↑[\] - Cursor nach rechts

↑[]] - Cursor nach unten

↑[^] - Cursor nach links

↑[O] - Invertiert den Bildschirm

3.3. Verwendung der Kontroll-Codes im Programm

Die Kontroll-Codes können mit Hilfe des Befehls PRINT CHR$([Zahl]) in Programmen benutzt werden. Die zulässigen Werte sind:

ASCII	FUNKTION	ASCII	FUNKTION	ASCII	FUNKTION
1	↑[A]	11	↑[K]	21	↑[U]
2	↑[B]	12	↑[L]	22	↑[V]
3	↑[C]	13	↑[M]	23	↑[W]
4	↑[D]	14	↑[N]	24	↑[X]
5	↑[E]	15	↑[O]	25	↑[Y]
6	↑[F]	16	↑[P]	26	↑[Z]
7	↑[G]	17	↑[Q]	27	↑[[]
8	↑[H]	18	↑[R]	28	↑[\]
9	↑[I]	19	↑[S]	29	↑[]]
10	↑[J]	20	↑[T]	30	↑[^]
				31	↑[O]

4. DIE FUNKTIONSTASTEN [F1] BIS [F10]

Die aktuelle Belegung der Funktionstasten erfahren Sie durch Eingabe des Befehls KEYLIST¬. Diese Belegung kann beliebig umprogrammiert werde. Bei der Auslieferung sind die SuperBASIC-Funktionstasten folgendermaßen belegt:

```
[F1]  -  RUN[E][M]
[F2]  -  LIST[Z][M]
[F3]  -  SCRNSET
[F4]  -  CONT[M]
[F5]  -  CURSOR
[F6]  -  CHR$(
[F7]  -  LOCATE"
[F8]  -  SCALE80[M]
[F9]  -  SAVE"
[F10] -  LOAD[M]
```

Die in eckigen Klammern stehenden Zeichen entsprechen den in Abschnitt 3.2 erläuterten Kontrollcodes.

4.1. Die Neubelegung der Funktionstasten

Die Funktionstasten können jederzeit mit der Befehlsfolge

```
KEY [Zahl] , "Funktion" + CHR$([Zahl])
```

neu programmiert werden.

Die Befehlsfolge, die der Taste zugeordnet werden soll, darf maximal 15 Zeichen, inklusiv Kontroll-Codes, lang sein. Mehrere Kontroll-Codes können gleichzeitig verwendet werden. Ein Kontroll-Code entspricht einem Zeichen. Beispiel:

```
KEY 4 , "KEYLIST" + CHR$(5) + CHR$(13)
```

Nach dieser Eingabe bewirkt ein Druck auf [F4] die Anzeige der kompletten Belegung der Funktionstasten. Neben 4 steht nun:

```
4      KEYLIST[E][M]
```

Wie Sie sehen, hat SuperBASIC [F4] neu belegt. Dabei wurden die Controllcodes CHR$(5) und CHR$(13) in [E] (= Löschen der logischen Zeile) und [M] (= [CR]) umgewandelt. Das Anhängen von CHR$(13) hat die direkte Ausführung des Befehls zur Folge gehabt.

5. DIE VARIABLENTYPEN

In SuperBASIC-Programmen können folgende Variablentypen verwendet werden:

5.1. Variablentyp STRING

Stringvariablen werden durch Anfügen des [$]-Zeichens gekennzeichnet. Beispiel: A$, XY$, FA$, usw.

Strings dürfen maximal 255 Zeichen enthalten.

Die Variablenzuweisung erfolgt nach den Mustern:

```
A$ = "[String]"
C$ = A$ + B$
X$ = "[String]" + B$
I$ = "[String]" + "[String]"
```

Einer Stringvariablen kann auch eine BASIC-Funktion zugewiesen werden, deren Befehlswort mit einem [$]-Zeichen endet (siehe auch Abschnitt "STRINGFUNKTIONEN").

Beispiel: A$ = SPACE$(10)

5.2. Variablentyp INTEGER

INTEGER-Variablen stellen ganze Zahlen im Bereich von -32768 bis +32767 dar. Dieser Variablentyp wird durch Anfügen des [%]-Zeichens an den Variablennamen deklariert.

Beispiel: A% = 50

Versucht man bei Zuweisungen oder Rechenoperationen mit Integerzahlen, den Wertebereich zu verlassen, wird ein Überlauf gemeldet.

5.3. Variablentyp SINGLE PRECISION

Zu diesem Variablentyp gehören Fließkommazahlen im Bereich von -10E38 bis +10E38. Der Exponent darf auch negative Werte annehmen. Eine Zahl dieses Typs besteht aus bis zu 7.1 signifikanten Stellen.

Bei einer Zuweisung werden alle darüberhinausgehenden Stellen abgeschnitten.

Das Kennzeichen einer Single-Precision-Variablen ist das [!]-Zeichen.

```
Beispiel: A! = 1234.567
          B! = 911.17E+9
```

Bei Überschreitung des Wertebereichs wird ein Überlauf gemeldet.

Werden Fließkommavariablen ohne Kennzeichen verwendet, werden sie von SuperBASIC automatisch als Single-Precision-Variablen deklariert.

5.4. Variablentyp DOUBLE PRECISION

Mit Double-Precision-Variablen können Zahlen im Bereich von -10D38 bis +10D38 mit doppelter Rechengenauigkeit verarbeitet werden. In exponentieller Darstellung erscheint hier im Gegensatz zu den Zahlen mit einfacher Genauigkeit ein [D] als Exponentialzeichen.

Die Darstellung dieser Zahlenart darf bis zu 16.8 signifikanten Stellen aufweisen.

Das Kennzeichen einer Double-Precision-Variablen ist das [#]-Zeichen.

Beispiel: A# = 123456789.0123456D+33

5.5. Globale Variablen-Deklaration

SuperBASIC läßt neben der einfachen Variablenzuweisung auch eine globale Variablen-Deklaration zu. Dabei kann einer Gruppe von Variablennamen die gewünschte Typenbezeichnung gegeben werden. Hierzu dient die Anweisung DEF in Verbindung mit den Zusätzen STR, INT, SGN und DBL.

Beispiel: 10 DEF STR A-D

Nach dieser Deklaration betrachtet SuperBASIC die Variablen A, B, C und D als Stringvariablen, obwohl sie keinen [$]-Affix haben.

Der Befehl DEF wird im Abschnitt "ANWEISUNGEN" näher erläutert.

5.6. Änderung von Variablentypen

Operationen und Zuweisungen dürfen nur mit Variablen des gleichen Typs ausgeführt werden, andernfalls erfolgt die Fehlermeldung "Typenfehler".

Ist dennoch eine gemeinsame Verarbeitung von Variablen unterschiedlichen Typs erforderlich, können sie mit Hilfe der Funktionen CINT, CSGN, CDBL und VAL angepasst werden (siehe Abschnitt "ARITHMETISCHE FUNKTIONEN").

5.7. Oktale und hexadezimale Zahlendarstellung

SuperBASIC ermöglicht außer der Darstellung dezimaler Zahlen auch die Verwendung der oktalen und hexadezimalen Schreibweise. Bei dieser Zahlendarstellung muß der jeweiliger Wert mit dem Kennzeichen [&O] für oktal bzw. [&H] für hexadezimal versehen werden.

Beispiel: A% = &HAFFE

Zahlen des Typs INTEGER können mit den Funktionen OCT$ und HEX$ in die gewünschte Darstellung überführt werden.

6. SUPERBASIC-BEFEHLE

SuperBASIC verfügt über folgende Befehle, die im Direkt-Modus und teilweise auch im Programm-Modus eingesetzt werden können:

AUTO AUTO [1. Zeilennummer] , [Zeilenabstand]

Dient der automatischen Erzeugung von Zeilennummern bei der Programmerstellung.
Wird [1. Zeilennummer] weggelassen, beginnt die die Numerierung bei Zeile 10 .
Wird [Zeilenabstand] weggelassen, erhöhen sich die Zeilennummern in Zehnerschritten.
Das Weglassen beider Argumente entspricht dem Befehl AUTO 10 , 10.
Der AUTO-Modus wird mit [SHIFT] + [BREAK] abgeschaltet.

CLEAR CLEAR CLEAR [Speicheradresse]

Die Verwendung des CLEAR-Befehls ohne Argument führt zum Löschen aller bisher benutzen Variablen. Eine eventuell vorhandene Speicherbegrenzung wird nicht aufgehoben.

CLEAR [Speicheradresse] begrenzt den Speicherbereich für BASIC-Programme und erlaubt das Anhängen von Maschinenunterprogrammen (vergl. LIMIT-Befehl in SHARP-BASIC).

Beispiel: CLEAR &HBF00

CONT

Mit CONT kann ein durch [SHIFT] + [BREAK] oder der STOP-Anweisung unterbrochenes Programm fortgesetzt werden.

Der CONT-Befehl wird nur ausgeführt, wenn hinter dem Bereitschaftszeichen [OK] ein Punkt steht. Ein unzuläßiger CONT-Befehl wird mit der Fehlermeldung "Geht nicht" quittiert.

DELETE DELETE [Anfangszeile] - [Endzeile]

DELETE dient zum gezielten Löschen von Programmzeilen.

Beispiel:	DELETE 100-200	- löscht Zeilen 100 bis 200
	DELETE -200	- löscht alle Zeilen bis 200
	DELETE 500-	- löscht alle Zeilen ab 500
	DELETE 190	- löscht Zeile 190
	DELETE -	- löscht das ganze Programm
	DELETE .	- löscht die aktuelle Zeile
	DELETE .-	- löscht ab der aktuelle Zeile
	DELETE -.	- löscht bis zur aktuellen Zeile

DUMP DUMP DUMP#

DUMP gibt alle bisher definierten Variablen, außer den Dimensionierten, mit ihren Inhalten auf dem Bildschirm oder, mit dem Affix [#], auf dem Drucker aus. Bei dimensionierten Variablen wird die Anzahl der vereinbarten Felder ausgegeben.

```
Beispiel: DUMP¬
          P $ = "ABCD"
          A # = 1.299999999D+18
          A $ (11,13)
```

Die Matrix A$ wurde mit DIM A$(10,12) dimensioniert. DUMP liefert die Menge der zur Verfügung stehenden Matrixfelder.

EDIT EDIT [Zeilennummer] EDIT.

Dieser Befehl druckt die gewählte Zeile aus und setzt den Cursor auf das erste Textzeichen. Wird EDIT mit dem Affix [.] eingegeben, wird die aktuelle Programmzeile geliefert. Dies ist in der Regel die letzte bearbeitete Zeile.

```
Beispiel: RUN¬
          Syntaxfehler in 750
          OK
          EDIT.¬
          750 PINT "ABBCDEFG"
              ↑
          Cursor blinkt hier
```

LIST LIST [Anfangszeile] - [Endzeile] LIST#

Mit LIST können beliebige Programmteile auf dem Bildschirm dargestellt werden.

```
Beispiel: LIST 50-90    -   listet die Zeilen 50 bis 90
          LIST -70      -   listet alle Zeilen bis 70
          LIST 80-      -   listet alle Zeilen ab 80
          LIST 20       -   listet Zeile 20
          LIST .        -   listet die aktuelle Zeile
          LIST .-       -   listet ab der aktuellen Zeile
          LIST -.       -   listet bis zur aktuellen Zeile
          LIST          -   listet das ganze Programm
```

Bei LIST# erfolgt die Ausgabe auf dem Drucker.

Das Auflisten von Programmzeilen kann mit [SPACE] angehalten und wieder fortgesetzt werden. Ein laufendes oder mit [SPACE] gestopptes Listing kann mit der Taste [Hochpfeil] in umgekehrter Richtung fortgeführt werden.

LOAD LOAD "[Programmname]"

LOAD dient zum Laden von BASIC- oder Maschinenprogrammen in den Speicher. Bei Maschinenprogrammen muß ein entsprechender Speicherbereich mit CLEAR [Speicheradresse] vom BASIC-Textbereich abgetrennt werden.

LOAD "[Programmname]" ermöglicht ein gezieltes Suchen nach dem zu landenden Programm. Folgt auf LOAD kein "[Programmname]", so wird das nächste auf der Kassette befindlichen Programm geladen.

LOCATE LOCATE "[Suchstring]" LOCATE# "[Suchstring]"

Dieser Befehl dient zur Auffindung von Befehlen bzw. Zeichen und Zeichengruppen, die im Programm verwendet wurden. Ausgegeben werden alle Programmzeilen, die den "[Suchstring]" enthalten. Mit dem Affix [#] erfolgt die Ausgabe auf dem Drucker.

Beispiel: LOCATE "REM"↵

```
 10 REM *** TESTPROGRAMM ***
 50 REM *** HAUPTPROGRAMM ***
100 REM *** SUBROUTINE 1 ***
200 REM *** SUBROUTINE 2 ***
300 REM *** DATEN ***
OK
```

Das Auflisten der gefundenen Zeilen kann mit [SPACE] gestoppt und wieder fortgesetzt werden. Mit der [Hochpfeil]-Taste wird ein laufendes oder angehaltenes Listing rückwärts fortgesetzt.

MERGE MERGE "[Programmname]"

Der Befehl MERGE dient zum Nachladen von Programmteilen in ein BASIC-Programm. Programmzeilen werden, wenn erforderlich, numerisch richtig einsortiert. Eventuell vorhandene Programmzeilen gleicher Nummerierung werden überschrieben. Nachzuladende Programme müssen als ASCII-Dateien aufgezeichnet worden sein (siehe SAVE). MERGE kann im Direkt- oder Programmodus verwendet werden.

NEW

Löscht den gesamten BASIC-Textbereich

RENUM RENUM RENUM [a] , [b] , [c]

Dient zur Neunummerierung der Programmzeilen, wobei [a] = neue Anfangszeile, [b] = alte Anfangszeile, [c] = neuer Zeilenabstand. Ein Weglassen der Argumente bewirkt die Verwendung von Standardwerten bei der Neunummerierung, wobei [a] = 10 , [b] = niedrigste vorhandene Zeilennummer, [c] = 10.

RUN RUN RUN [Zeilennummer]

Programmstartbefehl. Durch Anfügen einer Zeilennummer beginnt die Programmausführung an dieser Stelle.

SAVE SAVE SAVE "[Programmname]"
SAVE, A SAVE "[Programmname]", A

SAVE bewirkt die Aufzeichnung eines im Speicher befindlichen Programms. Dies kann mit "[Programmname]" (maximal 16 Zeichen) gekennzeichnet werden.

Durch Anfügen von [, A] wird das jeweilige Programm in ASCII- anstelle des Binär-Formats aufgezeichnet. Programmteile, die mit dem MERGE-Befehl nachgeladen werden sollen, müssen in ASCII-Format aufgezeichnet werden.

SCALE SCALE 40 SCALE 80

Löscht den Bildschirm und schaltet auf 40 bzw. 80 Zeichen pro Zeile.

SCRNSET SCRNSET [Anfangzeile] , [Zeilenzahl]

Definiert einen bestimmten Bildschirmbereich als "Fenster", in dem Text auf- bzw. abwärts gescrollt werden kann. Text, der in den Bildschirmbereichen ober- und unterhalb des "Fensters" steht, bleibt bei allen Bildschirmoperationen erhalten, es sei denn, der Bildschirmzugriff wurde mit einer CURSOR-Anweisung diesen Bereichen zugeordnet.

Beispiel: SCRNSET 10, 5↵

Mit diesem Befehl wird der Bildschirmarbeitsbereich auf die Zeilen 10 bis einschließlich 15 begrenzt. Alle Editierfunktionen, die über die Tastatur ausgeführt werden, sind nur in diesem Bereich wirksam. SCRNSET 0,25 gibt den gesamten Bildschirm wieder frei.

TRACE TRACE "[DHLNPS]" TRACE

Der TRACE-Befehl wird für die Fehlersuche bzw. die Programmanalyse eingesetzt. Insgesamt sind 6 Betriebsarten einzeln oder in beliebiger Kombination zulässig.

TRACE "D" (D = Dump)	-	Nach jeder abgearbeiteten Zeile werden die verwendeten Variablen mit ihren Inhalten ausgegeben.
TRACE "H" (H = High)	-	Die Anzeige der durch TRACE erzeugten Texte erfolgt in der obersten Bildschirmzeile.
TRACE "L" (L = Line)	-	Zeigt die gerade abgearbeitete Programmzeile an.
TRACE "N" (N = Number)	-	Zeigt die Nummer der gerade abgearbeiteten Zeile an.
TRACE "P" (P = Printer)	-	Die Trace-Ausgabe erfolgt auf dem Drucker.
TRACE "S" (S = Stop)	-	Die Programmausführung wird nach jeder abgearbeiteten Zeile gestoppt. Fortsetzung mit der [CR]-Taste.

Der TRACE-Modus wird durch Eingabe von TRACE ohne Argumente abgeschaltet.

Der TRACE-Vorgang kann durch Drücken der [BREAK]-Taste angehalten werden. Eine Fortsetzung erfolgt nach beliebigem Tastendruck.

7. DIE STANDARDANWEISUNGEN

SuperBASIC verfügt über alle Standardanweisungen und viele Sonderanweisungen, die das Programmieren westentlich erleichtern. Die Standardanweisungen sind:

CALL CALL [Adresse]

Diese Anweisung dient dem Aufruf von Maschinenunterprogrammen, die, falls eine Rückkehr zum BASIC erwünscht ist, mit RET (C9H) abgeschlossen sein müssen.

Beispiel: 500 CALL &HEFF0

Vergl. USR in SHARP-BASIC.

DATA DATA [Datum] , ... , [Datum]

Definiert numerische Werte bzw. Strings als Daten, die bei Bedarf Variablen zugewiesen werden können. Stringdaten, die Kommata enthalten, müssen in doppelten Anführungsstrichen stehen. Siehe auch READ und RESTORE.

Beispiel: 500 DATA 1, 5 , Buch, "veni, vidi, vici"

DEF DEF [Zusatz]

Mit DEF können Variablentypen oder Funktionen global deklariert werden. Solche Deklarationen sind eine Hilfe bei der Erstellung von strukturierten SuperBASIC-Programmen. Unter anderem ermöglichen sie eine Verkürzung des Quelltexts. Die DEF-Anweisungen sind:

DEF␣INT – Definiert spezifizierte Zahlenvariablen als Typ INTEGER.

DEF␣SNG – Definiert spezifizierte Zahlenvariablen als Typ SINGLE PRECISION.

DEF␣DBL – Definiert spezifizierte Zahlenvariablen als Typ DOUBLE PRECISION.

DEF␣STR – Definiert spezifizierte Variablen als Typ STRING.

Beispiel:
```
10 DEF INT A-F, X, Z
20 DEF SNG N, W
30 DEF DBL H-J
40 DEF STR M, O, T-V
```

DEF USR – Dient der Kennzeichnung von bis zu 10 verschiedene Maschinenunterprogrammen.

Beispiel
```
50 DEF USR 0 = &HB000
60 DEF USR 9 = &HCF18
```

Diese Unterprogramme können von SuperBASIC aus mit USR(n) aufgerufen werden.

DIM DIM [Variable ([Zahl] , [Zahl] , [Zahl])]

Mit DIM können Matrizen mit individuell ansprechbaren Feldern dimensioniert werden. Die Anweisung DIM A$(20,20,2) erzeugt eine dreidimensionale Matrize mit 21 x 21 x 3 Feldern. In SuperBASIC ist die Anzahl der Felder und Dimensionen nur durch die Speicherkapazität des Rechners begrenzt. Bei der Dimensionierung ist darauf zu achten, daß die Matrizenparameter in vernünftigen Grenzen bleiben, damit genug Platz für das Programm selbst zur Verfügung steht.

Matrizen können selbstverständlich auch mit Zahlenvariablen dimensioniert werden.

Beispiel: 10 DIM A$(20,6) , C#(27,6,2,2) , X(A,B,C)

END

Die Anweisung END beendet einen Programmlauf und schaltet den Direktmodus ein. Sie dient auch der Trennung von zwei oder mehr Programmen, die sich gleichzeitig im Speicher befinden.

FOR FOR [Laufvariable] = [Zahl 1] TO [Zahl2] STEP[Zahl 3]

FOR dient in Verbindung mit TO und STEP (siehe auch dort) zum Aufbau von Zählschleifen. Nach FOR wird die Laufvariable mit ihrem Anfangswert, z.B. als I = 1, definiert. Danach folgt TO mit dem Endwert der Schleife. Soll die Schrittweite (STEP) der FOR...TO-Schleife größer oder kleiner 1 sein, so muß sie mit dem Zusatz STEP festgelegt werden. Ohne STEP wird die Laufvariable bei jedem Durchlauf um 1 inkrementiert.

Beispiel: 10 FOR I = 100 TO 0 STEP -10

Im folgenden Beispiel wird die FOR...TO-Schleife infolge der Schrittweite von 0.1 insgesamt 991 mal durchlaufen, bevor der Endwert von I = 100 erreicht wird.

```
Beispiel: 10 FOR I = 1 TO 100 STEP 0.1
          20 Z = Z + 1: PRINT Z,
          30 NEXT I
          40 END
```

GOSUB GOSUB [Zeilennummer] GOSUB "[Label]"

GOSUB dient dem Aufruf von Unterprogrammen. Somit brauchen Programmteile, die immer wieder benötigt werden, nur einmal erstellt zu werden. Programmteile, die mit GOSUB angesprungen werden müssen mit RETURN abgeschlossen sein, um eine Rückkehr Hauptprogramm zu ermöglichen. Als Kennung des Anfangs von einem Unterprogramm kann entweder die entsprechende Zeilennummer oder die Anweisung LABEL (siehe unten) dienen. Unterprogramme, die mit GOSUB aufgerufen werden, können ineinander verschachtelt werden.

```
Beispiel: 100 GOSUB 120: GOSUB "UNTERPROGRAMM 2"
          110 END
          120 PRINT "UNTERPROGRAMM 1": RETURN
          130 LABEL "UNTERPROGRAMM 2"
          140 PRINT "LABELAUFRUF": RETURN
```

GOTO GOTO [Zeilenummer] GOTO "[Label]"

Mit GOTO kann ein unbedingter Sprung in ein anderes Programmteil erzielt werden. Der Programmlauf wird an der angesprungenen Stelle genauso fortgesetzt, als ob ein ununterbrochenes Programm vorliegen würde. Eine Rückkehr zur Zeile nach der GOTO-Anweisung ist nur mit einem weiteren GOTO möglich. Das Sprungziel kann auch hier durch Angabe der Zeilennummer oder eines Namens (siehe LABEL) definiert werden.

```
Beispiel: 100 GOTO 150
          110 END
          150 GOTO "Ende"
          160 LABEL "Ende"
          200 PRINT "Ende": END
```

IF IF [Bedingung] THEN [Option 1] ELSE [Option 2]
oder
GOTO

Mit der Anweisung IF wird der weitere Programmablauf einer oder mehreren Bedingungen unterworfen. Zuerst wird geprüft ob die Nach IF stehende [Bedingung] logisch "wahr" oder logisch "falsch" ist. Ist die [Bedingung] "wahr", wird die nach THEN stehende [Option 1] ausgeführt, andernfalls die nach ELSE stehende [Option 2]. Fehlt der Zusatz ELSE, ist nur eine [Option] gegeben. Wenn in diesem Fall die Prüfung der [Bedingung] logisch "falsch" liefert, wird die Programmausführung in der nächsten Zeile fortgesetzt. Ist die [Bedingung] wahr, so wird die angegebene Option ausgeführt.

```
Beispiel: 10 INPUT "Geben Sie eine Zahl ein"; X
          20 IF X > 5 THEN PRINT "X ist größer 5"
             ELSE PRINT "X ist kleiner 5"
          30 IF X >10 THEN PRINT "und größer 10"
             ELSE PRINT "und kleiner 10"
          40 GOTO 10
```

Die IF...THEN...ELSE-Konstruktion kann auch für bedingte Programmsprünge eingesetzt werden.

LABEL LABEL "[Name]"

Die LABEL-Anweisung erlaubt sowohl die Benennung von diskreten Programmteilen als auch die Behandlung von Unterprogrammen wie Prozeduren. SuperBASIC fördert damit ganz wesentlich die Übersichtlichkeit der einzelnen Routinen und erleichtert die Erstellung von logisch strukturierten Programmen. Die sogenannten "Spaghetti-Programme" gehören somit der Vergangenheit. Wird in einer Zeile die Anweisung LABEL verwendet, weist SuperBASIC diese Zeilennummer dem nach LABEL stehenden "[Namen]" zu. Dadurch kann in gesamten Programm dieser Name wie eine Zeilennummer verwendet werden. Zulässig sind daher GOTO "[Name]", GOSUB "[Name]", IF...THEN "[Name]", usw.

```
Beispiel: 10 GOSUB "Anfang"
          20 PRINT "Ende": END
          30 LABEL "Anfang"
          40 PRINT "Anfang und ";: RETURN
```

Anstelle von "Name" ist auch eine Stringvariable zulässig. Es können maximal 255 Namen mit LABEL vergeben werden.

```
Beispiel: 10 A$ = "Test"
          20 GOTO A$
          30 END
          40 LABEL A$
          50 PRINT "So geht es auch"
          60 END
```

LET LET [Variablenname] = [Wert]

LET ist eine Pseudoanweisung, die nur aus Gründen der Programmkompatibilität vorhanden ist. Sie dient der Variablenzuweisung. Da Variablen bei der ersten Verwendung im Programm automatisch den Wert Null erhalten und, wenn ein anderer Wert erwünscht ist, dies zum Beispiel mit " A = 2657 " erreicht werden kann, sind vorausgehende Definitionen und Zuweisungen mit LET in Super-BASIC unnötig und können weggelassen werden.

NEXT NEXT [Laufvariable A] , [Laufvariable B]

NEXT kennzeichnet das Ende eines Programmteils, welcher entsprechend des Werts der Laufvariblen , die durch die die Anweisungen FOR, TO und STEP definiert wurde, mit einer bestimmten Häufigkeit durchlaufen werden muß.
Da mehrere FOR-NEXT-Schleifen verschachtelt werden können, ist die Reihenfolge der Laufvariablen von Bedeutung. Sie müssen immer in umgekehrter Definitionsreihenfolge angeführt sein.
Mit einer NEXT-Anweisung können mehrere FOR-Schleifen geschlossen werden.

```
Beispiel: 10 FOR I = 1 TO 100
          20 FOR J = 1 TO 200
          30 PRINT I , J
          40 NEXT J , I
          50 END
```

ON ON [Variable] GOTO [Zeilennummer 1] , [Zeilennummer 2]
    oder
    GOSUB

Mit der Anweisung ON-GOTO oder ON-GOSUB lassen sich bedingte Sprünge oder Unterprogrammaufrufe programmieren. Der Wert der [Variablen] bestimmt einen Zeilennummer aus der nach GOTO oder GOSUB stehenden Liste. Der Wert entspricht der Position. Ist [Variable] gleich Null oder keine entsprechende Zeilennummer vorhanden, wird das Programm mit der folgenden Zeile fortgesetzt. In der Liste der Zeilennummern können auch mit LABEL definierte Namen verwendet werden.

```
Beispiel: 10 INPUT A
          20 ON A GOTO 40, 50, 60, "TEST", 900
          30 END
```

Die Anweisung ON wandelt den Wert von [Variable] zur Entscheidung über das Sprungziel in eine INTEGER-Zahl um (siehe INT).

PUSH
POP

PUSH bzw. POP [Variable 1],..., [Variable n]

Mit PUSH kann man die Inhalte von numerischen Variablen auf auf dem Stapelspeicher (Stack) ablegen. Man erhält dadurch die Möglichkeit, Werte vor der Veränderung durch eine Operation zwischenzuspeichern, um sie später mit Hilfe der POP-Anweisung zurückzurufen. Der Stapelspeicher funktioniert nach dem LIFO-Prinzip ("last in, first out"), d.h., mit POP wird der jeweils zuletzt auf den Stapelspeicher abgelegte Wert zurückgeholt. Da sich nur Werte auf dem Stapelspeicher befinden, können sie, falls erforderlich, neuen Variablennmen zugewiesen werden.

```
Beispiel: 10 A% = 1: B! = 2: C# = 3.9999
          20 PUSH A%, B!, C#
          30 POP D#, E!, F%     (← Zuweisung an neue Variablen)
          40 PRINT D#, E!, F%
```

REM

REM [Kommentar] ' [Kommentar]

REM ist eine Anweisung zur Einfügung von Kommentaren oder Erklärungen in ein Programm. REM-Anweisungen und nachfolgender Text werden bei der Programmausführung nicht beachtet. REM kann mit ' abgekürzt werden.

```
Beispiel: 235 REM Dies ist eine Beispielzeile
```

REPEAT

REPEAT [Programmteil] UNTIL [Bedingung]
oder REPEAT ON REPEAT OFF

Die REPEAT-Anweisung umfaßt zwei Funktionen:
1. Einleitung einer Schleifenkonstruktion
2. Ein- oder Ausschalten der Tastaturrepeatfunktion.

zu 1.:
Die Anweisung REPEAT ist die Einleitung der aus PASCAL entlehnten Schleifenkonstruktion REPEAT - UNTIL. Damit lassen sich sehr leistungsfähige und schnelle Programmschleifen aufbauen. Die Bedingung zum Verlassen der Schleife steht hier erst nach der Anweisung UNTIL. Eine Besonderheit dieser Konstruktion ist die Möglichkeit, ohne die Angabe von Zeilennummern zu programmieren.

```
Beispiel: 10 REPEAT
          20 A = A + 1
          30 UNTIL A = 20
          40 PRINT A
          50 END
```

Bei obigem Programm wird der Programmteil A = A + 1 so oft durchlaufen, bis A den Wert 20 erreicht hat. Erst danach wird das Programm fortgesetzt.
REPEAT - UNTIL-Anweisungen können verschachtelt werden. Bitte beachten Sie auch den Abschnitt "SCHLEIFENKONSTRUKTIONEN" dieses Handbuchs.

zu 2.
In Verbindung mit ON (REPEAT ON) oder OFF (REPEAT OFF) dient REPEAT dem Ein- oder Ausschalten der Tastenrepeatfunktion. Das Verhalten der Tastatur kann damit dem Wunsch des Benutzers angepasst werden. Vergleiche auch ↑Q ([SFTLOCK] + [Q]).

RESTORE RESTORE [Zeilennummer] RESTORE "[Label]"

Stellt DATA-Anweisungen, die vom Programm schon benutzt wurden, erneut zur Verfügung. Mit RESTORE [Zeilennummer] bzw. RESTORE "[Label]" kann der Beginn eines neuen Lesevorgangs durch READ festgelegt werden. RESTORE ohne Argument erlaubt das erneute Lesen beginnend mit der ersten DATA-Zeile des Programms.

Beispiel:
```
100 FOR I = 1 TO 9
110 READ I
120 NEXT I
130 DATA 3, 8, 10
140 DATA 6, 2, 1 <-------+
150 DATA 5, 7, 7         | Neuer DATA-Anfang
160 RESTORE 140 ---------+
```

Siehe auch DATA und READ.

RETURN

Beendet ein Unterprogramm, das mit GOSUB aufgerufen wurde. Die Anweisung bewirkt einen Rücksprung zu dem auf GOSUB folgenden Befehl.

Siehe auch GOSUB und ON GOSUB, sowie den Abschnitt "SCHLEIFEN-KONSTRUKTIONEN".

STOP

Bewirkt eine Unterbrechnung des laufenden Programms. Dient vornehmlich der Fehlersuche. Nach einer STOP-Anweisung kann der aktuelle Inhalt beliebiger Variablen untersucht werden. Anschließend kann die Programmausführung mit CONT↵ fortgesetzt werden. STOP hat dieselbe Wirkung wie die Tastenfolge [SHIFT]+[BREAK] und wird, wie diese, mit der Meldung "Break in [Zeilennummer]".

Eine Fortsetzung der Programmausführung nach STOP ist allerdings nicht möglich, wenn Änderungen im Programm vorgenommen worden sind. Siehe auch CONT.

SWAP SWAP [Variable 1] , [Variable 2]

Tauscht den Inhalt zweier Variablen aus. Die Variablen müssen gleichen Typs sein.

Beispiel:
```
10 A% = 5: B% = 7
20 SWAP A%, B%
30 PRINT "A% = "; A%
40 PRINT "B% = "; B%
RUN↵
A% = 7
B% = 5
```

UNTIL UNTIL [Bedingung]

UNTIL beendet eine mit REPEAT eingeleitete Bedingungsschleife. Trifft die nach UNTIL gegebenen [Bedingung] zu, wird das Programm mit der auf UNTIL [Bedingung] folgenden Anweisung fortgesetzt. Andernfalls wird der zwischen REPEAT und UNTIL stehender Programmteil wiederholt. Siehe auch REPEAT.

WHILE WHILE [Bedingung] [Programmteil] WEND

WHILE leitet die Bedingungsschleife WHILE ... WEND ein. Ist die nach WHILE stehende [Bedingung] logisch "wahr", wird der zwischen WHILE und WEND stehende Programmteil ausgeführt. Ist sie logisch "falsch", wird er übersprungen.

Beispiel:
```
10 A% = 20
20 WHILE A% > 0
30 A% = A% - 1
40 WEND
50 PRINT "A% ="; A%
RUN↵
A% = 0
```

Vergl. REPEAT ... UNTIL. Siehe auch Abschnitt "SCHLEIFEN-KONSTRUKTIONEN".

WEND

WEND schließt eine mit WHILE eingeleitete Bedingungsschleife ab (siehe oben).

8. ANWEISUNGEN ZUR EIN- UND AUSGABE

BEEP BEEP [Tonhöhe] , [Tondauer]

BEEP ohne Argument erzeugt einen anhaltenden Piepton mit einer Frequenz von 440 Hz.

Mit entsprechenden Argumenten können Tonhöhe und und -dauer variiert werden.

```
Beispiel: 10 BEEP 4, 5000
          20 BEEP 100, 100
```

Mit diesem Befehl können beliebige Ton- und Geräuscheffekte programmiert werden. Die Grenzwerte für die Argumente betragen:

[Tonhöhe] -32768 bis +32767 (teilweise außer Hörbereich)
[Tondauer] -32768 bis +32767

CLS CLS CLS [Argument]

Löscht den Bildschirm und stellt den Cursor in die HOME-Position. Die Verwendung von CLS mit Argumenten wird im Abschnitt "GRAFIK-ANWEISUNGEN" erläutert.

CURSOR CURSOR [X-Koordinate] , [Y-Koordinate]

Dient der Positionierung des Cursors auf dem Bildschirm. Für die Koordinaten sind folgende Werte zulässig:

Im 40-Zeichen-Modus: 0 <= X <= 39 und 0 <= Y <=24
Im 80-Zeichen-Modus: 0 <= X <= 79 und 0 <= Y <=24

CSRLIN

CRSLIN ist eine Pseudovariable, die die aktuelle Y-Koordinate des Cursors enthält. Sie liefert Werte zwischen 0 und 24.

```
Beispiel: 10 CURSOR 0, 10
          20 Y = CSRLIN
          30 PRINT Y
          RUN↵
           10
```

FRE(0) PRINT FRE(0)

FRE(0) ist eine Pseudofunktion. Sie liefert die Anzahl der noch freien Speicherstellen. Entspricht SIZE in SHARP-BASIC.

INKEY$ INKEY$ INKEY$([Zahl])

Mit INKEY$ können verschiedene Eingagbefunktionen laut folgender Tabelle ausgelöst werden:

Anweisung	Funktion
INKEY$	Entspricht der GET-Anweisung in SHARP-BASIC. Fragt die Tastatur ab und liefert, bei gedrückter Taste, das entsprechende Zeichen. Ist keine Taste gedrückt, wird ein Leerstring geliefert. INKEY$ hat keine Repetierfunktion.
INKEY$(-n)	Ermöglicht das gezielte Absuchen und Lesen einer Kassettendatei, ohne daß die Gesamtdatei geladen werden muß. Siehe Abschnitt "KASSETTENHANDLING"
INKEY$(0)	Wie INKEY$, jedoch mit sehr schneller Repetierfunktion.
INKEY$(n)	Wartet mit blinkendem Cursor auf eine Tasteneingabe. Bei einem Tastendruck wird das entsprechende Zeichen geliefert. Diese INKEY$-Funktion bedarf keiner Prüfung auf einen Leerstring. Das Pseudoargument (n) muß größer 0 und kleiner 255 sein.

Beispiel:

```
10 X = 0: Y = 0: Z = 0
20 A$ = INKEY$: IF A$ = "" THEN 20
30 PRINT A$; " ← 1. Eingabe": X = X + 1
40 IF X < 10 THEN 20
50 B$ = INKEY$ (0): IF B$ = "" THE 50
60 PRINT B$; " ← 2. Eingabe": Y = Y + 1
70 IF Y < 10 THEN 50
80 C$ = INKEY$ (1)
90 PRINT C$; " ← 3. Eingabe": Z = Z + 1
100 IF Z < 10 THEN 80
110 END
```

INPUT

```
INPUT [Varibable 1] , .... , [Variable n]
oder  INPUT "[Mitteilung]" , [Variable 1] , ......
oder  INPUT "[Mitteilung]" ; [Variable 1] , ......
```

Die Anweisung INPUT dient der Zuweisung von Eingaben an Variablen beliebigen Typs.

INPUT kann mit dem Zusatz "[Mitteilung]" versehen werden, um dem Anwender Hinweise zur erforderlichen Eingabe zu geben. Folgt nach [Mitteilung] ein Komma, erzeugt INPUT automatisch ein Fragezeichen.

Die Verwendung eines Semikolons unterdrückt die Anzeige des Fragezeichens.

INPUT kann zur Eingabe mehrerer Variablen verwendet werden. Diese Mehrfacheingaben müssen durch Kommata getrennt werden.

ACHTUNG: Sollen Stringdaten eingegeben werden, die Kommata enthalten, muß die Anweisung LINE INPUT verwendet werden (siehe dort).

INPUT findet auch in Verbindung mit Kassettendateien Anwendung. Die entsprechende Anweisung INPUT#-1 wird im Abschnitt "KASSETTENHANDLING" erläutert.

Beispiel:
```
10 INPUT"Eingabe: ", A$, B$, C$
20 INPUT"Soll die Eingabe wiederholt werden "; IN$
30 PRINT A$: PRINT B$: PRINT C$
40 IF IN$ ="JA" THEN 10 ELSE IF IN$ ="NEIN" THEN END
50 GOTO 10
```

LINE INPUT

```
LINE INPUT [Stringvariable]
LINE INPUT "[Mittteilung]"; [Stringvariable]
```

Ermöglicht die Eingabe von Strings, die Kommata enthalten. Nach Anbschluß der LINE INPUT-Eingabe enthält der [Stringvarible] die gesamte Eingabezeile, d.h. auch die [Mitteilung].

LINE INPUT erzeugt kein Fragezeichen. Es ist keine Mehrfacheingabe möglich.

Beispiel:
```
10 LINE INPUT"1. Eingabe: "; A$
20 LINE INPUT"2. Eingabe: "; B$
30 PRINT A$ : PRINT B$
40 LINE INPUT C$
50 PRINT C$ : END
```

INP

```
INP ([Kanalnummer])
```

Mit der Eingabeanweisung INP lassen sich die an den Peripheriekanälen des MZ-80 anliegenden Werte ablesen. [Kanalnummer] darf Werte von 0 bis 255, entsprechend den Z80-Portadressen, annehmen.

Beispiel:
```
10 FOR I = 0 TO 255
20 A = INP (I)
30 PRINT "Kanal Nummer:"; I ; " Wert:"; A
40 NEXT I
50 END
```

OUT

```
OUT [Kanalnummer] , [Wert]
```

Dient der Ausgabe eines Datenworts an einen Peripheriekanal des MZ-80. [Kanalnummer] darf, entsprechend den Z80-Portadressen, im Bereich von 0 bis 255 liegen. Die auszugebenden Werte müssen ebenfalls in diesem Bereich liegen.

Beispiel:
```
10 OUT 228, 0
20 X$ = INKEY$(1)
30 GOTO 10
```

PEEK

```
PEEK ([Adresse])
```

Mit dieser Anweisung können die Inhalte beliebiger Speicherstellen des MZ-80-Hauptspeichers gelesen werden. Der Wert [Adresse] muß im Bereich 0 bis 65535 bzw. &H0000 bis &HFFFF liegen.

Beispiel:
```
10 FOR I = 0 TO 100
20 A = PEEK (I)
30 PRINT A;
40 NEXT I
50 END
```

POKE POKE [Adresse] , [Wert 1] , ... , [Wert n]

Dient der Programmierung einer oder mehrerer Speicherstellen des Hauptspeichers.

ACHTUNG: Wird der POKE-Befehl in Bereichen verwendet, die nicht zuvor mit der Anweisung CLEAR [Adresse] freigegeben wurden, kann es zur Zerstörung des Programms kommen!

[Adresse] darf Werte von 0 bis 65535 bzw. &H0000 bis &HFFFF annehmen. [Wert] muß im Bereich 0 bis 255 liegen.

POKE erlaubt die Verarbeitung mehrerer Werte mit einer Anweisung. Dazu wird die [Adresse] automatisch erhöht.

Beispiel:

```
10 CLEAR &HEFFF
20 FOR I = 1 TO 200
30 A = PEEK (&H1000+I) : B = PEEK (&H2AF0+I)
40 POKE &HF000 + (I*2) , A , B
50 NEXT I
60 END
```

MEM$ MEM$ ([Adresse] , [Länge])

MEM$ dient, wie PEEK, dem Auslesen von Speicherstellen. Es können hier jedoch Bereiche von bis zu 255 Adressen gelesen und einer Stringvariablen zugewiesen werden. [Adresse] darf Werte von 0 bis 65535 bzw. &H0000 bis &HFFFF annehmen.

Beispiel:

```
10 A$ = MEM$(&H180,40)
20 PRINT A$
30 END
```

OFF OFF OFF OFF

OFF unterbindet die [SHIFT] + [BREAK] Funktion. Eine Tastaturabfrage findet nur noch bei Eingabeanweisungen statt. Hierdurch wird die Programmausführung beschleunigt.

Mit OFF OFF wird die Tastaturabfrage wieder aktiviert. [SHIFT] + [BREAK] ist dann wieder möglich.

Beispiel:

```
 10 OFF
 20 TIME$ = "00:00:00"
 30 FOR I = 1 TO 500
 40 BEEP I, 10000/I: NEXT I
 50 PRINT TIME$
 60 OFF OFF
 70 TIME$ = "00:00:00"
 80 FOR I = 1 TO 500
 90 BEEP I, 10000/I: NEXT I
100 PRINT TIME$: END
```

POS POS(n)

Liefert die horizontale Position des Cursors.

```
Beispiel: 10 SCALE 80
          20 FOR Y = 0 TO 24
          30 FOR X = 0 TO 79
          40 CURSOR X,Y
          50 PRINT POS(0)
          60 FOR I = 1 TO 50: NEXT I
          70 NEXT X, Y
          80 END
```

Das Pseudoargument (n) kann Werte von 0 bis 240 annehmen.

SYNT SYNT "[String]" SYNT [Stringvariable]
SYNT [Zahl] SYNT [Zahlenvariable]

SYNT (= Synthesize) entspricht den TEMPO- und MUSIC-Anweisungen von SHARP-BASIC. Das Tempo wird mit der Anweisung SYNT n festgelegt, wobei n Werte von 0 bis 7 annehmen kann. Die Melodie, die einem Stringvariablen zugewiesen werden kann, muß als String geschrieben werden. Der Notenbereich umfasst drei Oktaven:

```
Untere Oktave:    -C, -D, -E, -F, -G, -A, -B
Mittlere Oktave:   C,  D,  E,  F,  G,  A,  B
Obere Oktave:     +C, +D, +E, +F  +G  +A  +B
```

Pausen können mit R programmiert werden.

Die Noten- bzw. Pausendauer wird mit Zahlen im Bereich von 0 bis 9 festgelegt.

Halbtonschritte können mit dem #-Zeichen erzeugt werden.

```
Beispiel: 10 SYNT 3
          20 M$ = "A3 A5 +#C3 +D +E +#F +G +#F0 +G3"
          30 SYNT M$
```

TIME$ TIME$ "[Stunden] : [Minuten] : [Sekunden]"

Mit TIME$ wird die interne Computeruhr eingestellt. Versucht man die zulässigen Zeitwerte zu überschreiten, wird ein falscher Funktionsaufruf gemeldet. Die Eingaben [Stunden], [Minuten] und [Sekunden] müssen mit einem Doppelpunkt voneinander getrennt werden

LPOS LPOS(0)

LPOS(0) liefert die aktuelle Position des Druckkopfs.

```
Beispiel: 10 PRINT#1, "Der Druckkopf ist bei Position";
          20 PRINT#1, LPOS(0)
```

Diese Funktion liefert nur dann Werte, wenn seit der letzten Zeichenausgabe an den Drucker noch kein Wagenrücklauf erfolgt ist.

READ

Die READ-Anweisung dient der Eingabe von Daten aus DATA-Anweisungen in das Programm. Das zu lesende Datum wird durch die Position des Datenzeigers bestimmt. Der Datenzeiger kann durch RESTORE beeinflusst werden. (Siehe auch DATA und RESTORE).

```
Beispiel: 10 DIM A$(9)
          20 FOR I = 0 TO 9
          30 READ A$(I): PRINT A$(I);" ";
          40 NEXT I
          50 DATA Dies, ist, ein, Beispiel, für, die,
             Dateneingabe, mit, READ, .
          60 END
```

SPC SPC([Zahl]) SPC([Zahlenvariable])

Mit der SPC-Anweisung können Leerzeichen auf dem Bildschirm ausgegeben werden. Sie wird benutzt, um bestimmte Bereiche einer Zeile bis zur gewünschten Cursorposition zu löschen.

Im Gegensatz zu SPACE$ kann SPC nicht als Stringvariable genutzt werden.

```
Beispiel: 10 CURSOR 0,0
          20 PRINT "SuperBASIC ist ein gutes BASIC"
          30 FOR I = 1 TO 1000: NEXT I
          40 CURSOR 15,0
          50 PRINT SPC(9)
          60 FOR I = 1 TO1000: NEXT I
          70 CURSOR 15,0
          80 PRINT "DAS BESTE"
          90 END
```

TAB TAB([Zahl]) TAB([Zahlenvariable])

Die TAB-Anweisung dient zur Festsetzung von Tabulationspunkten auf dem Bildschirm bzw. Drucker. Bei der Bildschirmausgabe unterscheidet sie sich von der TAB-Anweisung in SHARP-BASIC dadurch, daß alle Bildschirmpositionen zwischen der aktuellen Cursorposition und der nächsten TAB-Position gelöscht werden.

Diese Einrichtung ist besonders bei dynamischen Tabellen nützlich, deren Feldinhalte ständig wechseln.

```
Beispiel: 10 CLS: X=0
          20 FOR I = 1 TO 100
          30 X = X + 1
          40 CURSOR 0,0
          50 PRINT TAB(0); X/1.4;
          60 PRINT TAB(15); X/3.3;
          70 PRINT TAB(30); X/5.6;
          80 PRINT TAB(45); X/9.7
          90 FOR PAUSE = 1 TO 1000: NEXT PAUSE
         100 NEXT I
         110 END
```

VARPTR VARPTR([Variable])

VARPTR (= Variablen-Pointer) ermittelt die Speicheradresse des ersten Bytes einer Variablen. Der gelieferte Wert entspricht einer INTEGER-Zahl. Da dieser Zahlentyp die Zweierkomplement-Darstellung für negative Zahlen verwendet, muß die Adresse vor der absoluten Darstellung mit der Funktion HEX$ in Hexform umgewandelt werden.

VARPTR wird benötigt, um bei der Koppelung von SuperBASIC mit einem Maschinenprogramm direkt auf SuperBASIC-Variablen zugreifen zu können. Siehe auch CALL und USR.

Beispiel:
```
10 A# = 1.458: B$ = "TEST FÜR VARPTR"
20 W1 = VARPTR(A#)
30 W2 = VARPTR(B$)
40 PRINT "Dezimalwert von W1 =";  W1
50 PRINT "Hexwert von W1 =    ";  HEX$(W1)
60 PRINT
70 PRINT "Dezimalwert von W2 =";  W2
80 PRINT "Hexwert von W2     =";  HEX$(W2)
```

Der gelieferte Wert zeigt auf die Speicherstelle(n), die die Variable enthält.

8. 1. Die PRINT-Anweisungen

SuperBASIC enthält einige sehr leistungsfähigen Anweisungen für die formatierte Druckausgabe, sei es auf Bildschirm, Datenträger oder Drucker.

PRINT

Diese Anweisung kann laut folgender Tabelle verwendet werden:

Anweisung	Wirkung
PRINT [Zahl]	[Zahl] wird an der aktuellen Cursorposition auf dem Bildschirm gedruckt
PRINT [Variable]	Inhalt von [Variable] wird an der aktuellen Cursorposition auf dem Bildschirm gedruckt
PRINT "[String]"	Inhalt von [String] wird an der aktuellen Cursorposition auf dem Bildschirm gedruckt
PRINT A , B , C	Die Inhalte der Variablen A, B und C werden an den jeweils nächsten freien TAB-Positionen auf dem Bildschirm gedruckt
PRINT A$, B$, C$	Vergl. oben
PRINT A ; B ; C	Die Inhalte der Variablen A, B und C werden folgendermaßen auf dem Bildschirm gedruckt: <Vorzeichen (= Leer- oder Minuszeiche)> <Zahl> <Leerzeichen> Bitte beachten Sie den Unterschied zum SHARP-BASIC, das kein Leerzeichen hinter der Zahl ausgibt
PRINT A$; B$; C$	Wie oben aber mit dem Unterschied, daß keine Leerzeichen zwischen den ausgegebenen Strings stehen
PRINT	Erzeugt eine Leerzeile auf dem Bildschirm
PRINT 5 * SIN(A)	Druckt das Ergebnis von Berechnungen aus
PRINT#1, [...]	Bewirkt die Ausgabe auf dem Drucker. Alle obigen Argumentenformen sind zulässig
PRINT#-1,	Siehe Abschnitt "KASSETTENHANDLING"

8.2. Die PRINT USING-Anweisungen

PRINT USING ist ein äußerst nützliches und extrem leistungsfähiges Instrument zur formatierten Druckausgabe. In SuperBASIC sind folgende Formatierungen möglich:

PRINT USING "#####"; A

Der Inhalt der Variablen A wird dem Format "#####" rechtsbündig überlagert und dann gedruckt. Das Format "#####" kann um zusätzliche zu druckende Zeichen erweitert werden. Das Format selbst darf maximal 24 Zeichen, inklusiv Zusatzzeichen, enthalten.

Beispiel:
```
10 A = 1: B = 12 : C = 123: D = 1234
20 F$ = "Stückzahl ####"
30 PRINT USING F$; A
40 PRINT USING F$; B
50 PRINT USING F$; C
60 PRINT USING F$; D
RUN¬
Stückzahl       1
Stückzahl      12
Stückzahl     123
Stückzahl    1234
```

PRINT USING "#####.##"; A

Der Inhalt der Variablen A wird dem Format so überlagert, daß der ganzzahlige Teil rechtsbündig an das Punktzeichen anschließt und die Nachkommastellen auf die vorgegebene Länge gerundet werden. Ist die geforderte Anzahl der Nachkommastellen nicht oder nur teilweise gegeben, werden die Reststellen mit Nullen aufgefüllt.

Beispiel:
```
10 A# = 1.39961: B# = 1694.12309: C# = 22
20 F$ = "DM ####.##"
30 PRINT USING F$; A#
40 PRINT USING F$; B#
50 PRINT USING F$; C#
RUN¬
DM    1.40
DM 1694.12
DM   22.00
```

Die oben beschriebenen PRINT USING-Anweisungen erleichtern das Programmieren von numerischen Tabellen jeglicher Art (Preislisten, Kostenvoranschlägen, Berechnungsergebnisse, usw.). Da bei PRINT USING"####.##" die Dezimalbruchstellen richtig auf- oder abgerundet werden, werden alle Kalkulationen auf die gewünschte Anzahl von Nachkommastellen genau berechnet

```
PRINT USING "-#####.###"; A        PRINT USING"####.###+"; A
```

Diese Abwandlung des Standardformats bewirkt das Anfügen eines Vorzeichens an die formatierte Zahl. Stimmt das Vorzeichen der Variablen mit dem des Formats überein, wird es entsprechend der Formatanweisung angezeigt. Trifft diese Bedingung nicht zu, erscheint sein Leerzeichen. Die Vorzeichenfunktion kann mit allen anderen USING-Anweisungen verbunden werden. Das gewünschte Vorzeichen kann vor oder nach der Zahl stehen.

```
PRINT USING "******.##"; A
```

Die Verwendung von zwei [*]-Zeichen in der Formatanweisung bewirkt ein Auffüllen etwaiger Leerstellen mit Sternchen. Der Ausdruck von Zahlen oder Zahlenvariablen erfolgt rechtsbündig und fehlende Nachkommastellen werden mit Nullen aufgefüllt. Das Argument ** muß immer vorne stehen.

```
Beispiel: 10 A# = 1234.5555 : B# = 4353.345 : C# = 323.2
          20 PRINT USING "**********.##  ";A#;B#;C#
          RUN¬
          ******1234.56  ******4353.35  ********323.20
```

```
PRINT USING "######.###^^^^"; A
```

Das Einfügen von [^^^^] in die Formatanweisung bewirkt eine Zahlendarstellung in Exponentialform. Die zu formatierende Zahl wird dem Format [######.###] linksbündig überlagert und die Leerstellen mit Nullen gefüllt. Anschließend erhält die so formatierte Zahl den entsprechenden Exponenten.

```
Beispiel: 10 A# = 12345.2535D+12: B# = 345.454545: C# = 23.1D+10
          20 PRINT USING "######.###^^^^";A#;B#;C#
          RUN¬
          1234525.350D+10 3454545.450D-04 2310000.000D+05
```

```
PRINT USING "!"; ST$          PRINT USING "!####.##"; ST$ ;A
```

In dieser Formatanweisung wird das [!]-Zeichen als Platzhalter für das erste Zeichen der Stringvariablen ST$ verwendet. Das [!]-Zeichen muß immer links stehen. Werden mehrere [!]-Zeichen verwendet, muß eine gleiche Anzahl Stringvariablen angegeben sein. Eine Verwendung mit anderen Formaten ist zulässig.

```
Beispiel: 10 A# = 12345.6789 : ST$="%ABC"
          20 PRINT USING "!######.##";ST$;A#
          RUN¬
          % 12345.68
```

```
PRINT USING "&     &"; ST$
```

Mit der Formatanweisung ["& &"] ist es möglich, Stringvariablen linksbündig formatiert auszugeben. Stringvariablen, die länger als das Format sind, werden gekürzt. Die Formatlänge = 2 (für die beiden [&]-Zeichen) plus n, wo n = Anzahl der zwischen den [&]-Zeichen stehenden Leerzeichen.

```
Beispiel: 10 A$ = "STRING 1": B$ = "TESTSTRING": C$ = "ABCDEF"
          20 PRINT USING "&              &"; A$; B$; C$
          RUN¬
          STRING 1        TESTSTRING       ABCDEF
```

```
PRINT USING "¯#####.##"; A
```

Ein Hochstrich am Anfang einer Formatanweisung ermöglicht das Drucken von Zeichen, die normalerweise nur als Formatierungszeichen verwendet werden dürfen.

```
Beispiel: 10 A# = 123.564: B# = 3456.9: C# = 9097.94
          20 PRINT USING "¯#######.###    "; A#; B#; C#
          RUN¬
          #  123.564     # 3456.900     # 9097.940
```

9. DIE MATHEMATISCHEN FUNKTIONEN

ABS ABS([Zahl]) ABS([Variable])

Liefert den Absolutwert einer Zahl oder Zahlenvariablen.

Beispiel:
```
PRINT ABS(-17.39)¬
 17.39
PRINT ABS(35.9)¬
 35.9
```

ATN ATN([Zahl]) ATN([Variable])

Ermittelt den Arcustangens einer Zahl oder Zahlenvariablen.

Beispiel:
```
PRINT ATN(10)¬
 84.289407
```

CDBL CDBL([Variable]%) CDBL([Variable!])

Wandelt Zahlenvariablen des Typs INTEGER (%) bzw. SINGLE PRECISION (!) in Variablen des Typs DOUBLE PRECISION (#) um. Wenn in Berechnungen unterschiedliche Variablentypen gemischt werden, ohne daß eine Typanpassung vorgenommen wurde, ist für die Rechengenauigkeit die niedrigste verwendete Variablenebene maßgebend.

Beispiel:
```
10 BD# = CDBL(Y%)
20 CX# = CDBL(N!)
30 A# = CDBL(B)
```

CINT CINT([Variable]#) CINT([Variable!])

Mit dieser Funktion lassen sich Variablen des Typs DOUBLE oder SINGLE PRECISION in INTEGER-Variablen konvertieren.

Beispiel:
```
10 M% = CINT(X#)
20 K% = CINT(K!)
```

COS COS([Zahl]) COS([Variable])

Berechnet den Cosinus einer Zahl oder Zahlenvariablen. Die Genauigkeit wird durch den Zahlen- oder Variablentyp bestimmt.

Beispiel:
```
10 PRINT COS(0.5)
RUN¬
 1
```

CSGN CSGN([Variable]%) CSGN([Variable]#)

Konvertiert Zahlenvariablen des Typs INTEGER bzw. DOUBLE PRECISION in den Typ SINGLE PRECISION.

EXP EXP([Zahl]) EXP([Variable])

Die Funktion EXP(X) ermittelt den Wert e^{x} mit einer der dem Variablentyp X entsprechenden Genauigkeit.

Beispiel:
```
PRINT EXP(1#)¬
 2.71828182818181
```

FAC FAC([Zahl]) FAC([Variable])

Mit FAC(X) wird die Fakultät X! einer ganzen Zahl ermittelt. Ist die Variable(X) keine INTEGER-Variable, wird sie zur Berechnung in eine solche umgewandelt.

```
Beispiel: PRINT FAC(7)¬
           5040
```

FIX FIX([Zahl]) FIX([Variable])

Die FIX-Funktion dient dazu, alle Zahlen, die rechts von einem Dezimalpunkt stehen, abzuschneiden. Die Zahlen werden nicht gerundet (vergl. INT).

```
Beispiel: PRINT FIX(5.78)¬
           5
          PRINT FIX(-7.999)
          -7
```

INT INT([Zahl]) INT([Variable])

Die INT-Funktion schneidet alle Zahlen ab, die rechts von einem Dezimalpunkt stehen. Dabei werden negative Werte abgerundet.

```
Beispiel: PRINT INT(8.333)¬
           8
          PRINT INT(-3.7)¬
          -4
```

LGN LGN([Zahl]) LGN([Variable])

Berechnet den natürlichen Logarithmus einer Zahl oder Zahlenvariablen:

```
Beispiel: PRINT LGN(10)¬
```

RND RND(0) RND(1)

Ermittelt "Zufallszahlen" im Bereich zwischen 0 und 1. RND(0) liefert immer die gleiche "Zufallszahl", RND(1) ständing wechselnde Werte.

SGN SGN([Zahl]) SGN([Variable])

Liefert das Signum (Vorzeichen) einer Zahl oder Zahlenvariablen.

```
Beispiel: PRINT SGN(10)¬
           1
          PRINT SGN(0)¬
           0
          PRINT SGN(-27.9293)
          -1
```

1 bedeutet positives Vorzeichen (+)
0 zeigt an, daß die Zahl bzw. Variable gleich Null ist
-1 bedeutet negatives Vorzeichen (-)

SIN SIN([Zahl]) SIN([Zahlenvariable])

Berechnet den Sinus einer Zahl oder Zahlenvariablen. Die Genauigkeit wird durch den Zahlen- oder Variablentyp bestimmt.

Beispiel: PRINT SIN(1)¬
0

SQR SQR([Zahl]) SQR([Zahlenvariable])

Ermittelt die Quadratwurzel einer Zahl oder Zahlenvariablen.

Beispiel: PRINT SQR(1024)¬
32

TAN TAN([Zahl]) TAN([Zahlenvariable])

Berechnet den Tangens einer Zahl oder Zahlenvariablen.

Beispiel: PRINT TAN(1)¬
1.55741

PIX PIX([Zahl]) PIX([Zahlenvariable])

Berechnet das X-fache von π.

Beispiel: PRINT PIX(1#)¬
3.141592653589793
PRINT PIX(3)¬
9.42478

RAD RAD([Zahl]) RAD([Zahlenvariable])

Wandelt Altgrad in Radian um.

Beispiel: PRINT RAD(9)¬
.15708

SUM SUM([Zahl]) SUM([Zahlenvariable])

SUM(X) liefert die Summe aller ganzen Zahlen von 0 bis X.

Beispiel: PRINT SUM(9)¬
45

MKD$ MKD$([Zahl]) MKD$([Zahlenvariable])

Konvertiert eine Zahl bzw. eine Zahlenvariable des Typs DOUBLE PRECISION in einen 8-Byte-ASCII-String, die man einer Stringvariablen zuweisen kann. Solche Strings lassen sich mit der Anweisung CVD([Stringvariable]) in Zahlen zurückverwandeln. Die MKD$-Anweisung wird für die Konvertierung von numerischen Daten verwendet, die man im ASCII-Format aufzeichnen will. Für die Konvertierung anderer Zahlentypen stehen MKI$ und MKS$ zur Verfügung (siehe unten).

```
Beispiel: 10 X# = 123456789012345
          20 X$ = MKD$(X#)   ← Konvertierung
          30 Y# = CVD(X$)    ← Rückkonvertierung
```

MKI$ MKI$([Zahl]) MKI$([Zahlenvariable])

Konvertiert Zahlen bzw. Zahlenvariablen des Typs INTEGER in 2-Byte-ASCII-Strings. Anwendung: Siehe MKD$.

MKS$ MKS$([Zahl]) MKS$([Zahlenvariable])

Konvertiert Zahlen bzw. Zahlenvariablen des Typs SINGLE PRECISION in 4-Byte-ASCII-Strings. Anwendung: Siehe MKD$.

CVD CVD([Stringvariable])

Erlaubt die Rückkonvertierung von Zahlen bzw. Zahlenvariablen, die mit der MKD$-Anweisung in 8-Byte-ASCII-Strings verwandelt und Stringvariablen zugewiesen wurden. Anwendungs: Siehe MKD$

CVI CVI([Stringvariable])

Wie CVD, aber nur bei Stringvariablen anwendbar, die mit der MKI$-Anweisung erstellt wurden. Anwendung: Siehe MKD$.

CVS CVS([Stringvariable])

Wie CVD, aber nur bei Stringvariablen gültig, die mit der MKS$-Anweisung erstellt wurden. Anwendung: Siehe MKD$.

10. MATHEMATISCHE UND LOGISCHE OPERATOREN

SuperBASIC verfügt über alle mathematischen und logischen Operatoren, die in den Hochsprachen gebräuchlich sind.

10.1. Mathematische Operatoren

Mathematische Operatoren dienen der Manipulation von numerischen Daten gemäß ihrer hierarchischen Ordnung.

= Zuweisungoperator. Dient der Zuweisung eines Werts bzw. des Inhalts einer Variablen an eine andere Variable. Die Zuweisung erfolgt IMMER von rechts nach links.

```
Beispiel: 10  AX = 12345    (Der Variablen AX wird der
                             Wert 12345 zugewiesen)

          20  AY = AX       (Der Inhalt des Variablen
                             AX wird der Variablen AY
                             zugewiesen)
```

\- Subtraktionsoperator. Subtrahiert Zahlen bzw. die Inhalte von Variablen.

```
Beispiel: 10 X = 10: Y = 5
          20 Z = X - Y
          30 PRINT Z
          RUN¬
           5
```

\+ Additionsoperator. Addiert Zahlen bzw. die Inhalte von Variablen.

```
Beispiel: 10 A = 100
          20 B = A + 10
          30 PRINT B
          RUN¬
           110
```

* Multiplikationsoperator. Multipliziert Zahlen bzw. die Inhalte von Variablen miteinander.

```
Beispiel: 10 K = 12: L = 5
          20 PRINT K * L
          RUN¬
           60
```

/ Divisionsoperator. Dividiert Zahlen bzw. die Inhalte von Variablen aller arithmetischen Variablentypen.

```
Beispiel: PRINT 60/3¬
           20
```

\ Divisionsoperator für Zahlen bzw. Variablen des Typs INTEGER. Hat die gleiche Funktion wie [/], ist in der Ausführung jedoch erheblich schneller.

```
Beispiel: 10 X% = 15: Y% = 5
          20 PRINT X% \ Y%
          RUN¬
           3
```

↑ Potenzoperator. Berechnet eine beliebige Potenz einer Zahl bzw. einer Zahlenvariablen.

```
Beispiel: PRINT 3 ↑ 5¬
           243
```

MOD Operator für die Modulodivision. Ermittelt den Rest einer Divisionsoperation.

```
Beispiel: PRINT 7 MOD 3¬
           1
```

Alle Berechnungen und Funktionen können mit SuperBASIC entsprechend der mathematischen Hierarchie (z.B. Punkt-vor-Strich-Rechnung) programmiert werden. Vielfache Klammerebenen sind zulässig.

10.2. Vergleichsoperatoren

Folgende Vergleichsoperatoren werden von SuperBASIC akzeptiert und liefern als Ergebnis entweder logisch "wahr" oder logisch "falsch":

= Liefert bei numerischem Gleichsein logisch "wahr"

```
Beispiel: 10 A% = 5: B% = 4
          20 IF A% = B% THEN PRINT "A% = B%" ELSE PRINT
             "A% <> B%"
          30 END
          RUN¬
          A% <> B%
```

Da in diesem Fall kein numerisches Gleichsein vorhanden war, wurde logisch "falsch" geliefert.

< Beim Vergleich A < B liefert dieser Operator logisch "wahr", dann und nur dann, wenn A kleiner B.

```
Beispiel: 10 IF 10 < 10 THEN PRINT 10: GOTO 30
          20 PRINT 999
          30 END
          RUN¬
           999
```

> Beim Vergleich A > B liefert dieser Operator logisch "wahr", dann und nur dann, wenn A größer B.

```
Beispiel: 10 IF 5 > 3 PRINT 5 + 3 ELSE PRINT 5 - 3
          20 END
          RUN¬
           8
```

<> Dieser Operator untersucht zwei Argumente auf Ungleich-
>< heit und liefert logisch "wahr", dann und nur dann, wenn eine Ungleichheit festgestellt wird.

```
Beispiel: 10 IF 10 <> 9 THEN 30
          20 PRINT "FALSCH": GOTO 40
          30 PRINT "WAHR"
          40 END
          RUN¬
          WAHR
```

>= Dieser Operator liefert beim Vergleich von A und B lo-
=> gisch "wahr", dann und nur dann, wenn A größer oder gleich B.

```
Beispiel: 10 INPUT "A ="; A%
          20 INPUT "B ="; B%
          30 IF A% >= B% THEN PRINT "WAHR"
             ELSE PRINT "FALSCH"
          40 END
          RUN¬
          A = ? 1¬
          B = ? 2¬
          FALSCH
```

<= Dieser Operator liefert beim Vergleich von A und B lo-
=< gisch "wahr", dann und nur dann, wenn A kleiner oder gleich B.

```
Beispiel: 10 INPUT "A ="; A%
          20 INPUT "B ="; B%
          30 IF A% <= B% THEN PRINT "WAHR"
             ELSE PRINT "FALSCH"
          40 END
          RUN¬
          A = ? 1¬
          B = ? 2¬
          WAHR
```

Die Begriffe logisch "wahr" und logisch "falsch" werden von Super-BASIC in der Form -1 (für "wahr") bzw. in der Form 0 (für "falsch") dargestellt. Bei Bedarf könne diese Werte Variablen zugewiesen werden.

```
Beispiel: 10 A% = 9: B% = 7
          20 C% = A%<>B%
          30 PRINT C%
          RUN¬
          -1
```

10.3 Logische Operatoren

Die logischen Operatoren vergleichen zwei Argumente bitweise auf logisch "wahr" oder "falsch". Dieser Vergleich wird gemäß nachfolgender Wahrheitstabellen durchgeführt.

Bezüglich logischer Entscheidungen und deren Auswertung siehe auch Abschnitt 10.2.

Operator	A	B	A Operator B
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
AND	1	1	1
	1	0	0
	0	1	0
	0	0	0
XOR	1	1	0
	1	0	1
	0	1	1
	0	0	0

NOT Der Operator NOT invertiert einen logischen Zustand.

A	NOT A
1	0
0	1

In obigen Wahrheitstabellen wurde aus Gründen der Übereinstimmung mit der gebräuchlichen Darstellung die Bezeichnung 0 für "falsch" und 1 für "wahr" verwendet.

SuperBASIC verwendet, wie schon erwähnt, -1 für "wahr" und 0 für "falsch". Diese Werte werden als Ergebnis einer logischen, bitweisen Operation erzeugt.

Als Alternative zu AND und OR kann man die in SHARP-BASIC üblichen logischen Operatoren * und + mit eingeklammerten Argumenten auch in SuperBASIC verwenden.

11. STRING-FUNKTIONEN

ASC ASC([String]) ASC([Stringvariable])

Liefert den ASCII*-Wert des ersten im Argument angegebenen Zeichens. Die Zuweisung erfolgt gemäß der ASCII-Code-Tabelle im SHARP BASIC-Handbuch. Die ASC-Funktion liefert immer einen Wert von 0 bis 255.

```
Beispiel: 10 PRINT ASC("A")
          20 PRINT ASC("ABC")
          RUN¬
           65
           65
```

* ASCII steht für "American Standard Code for Information Interchange" (= Amerikanischer Standard-Code für Informationsaustausch).

CHR$ CHR$([Zahl]) CHR$([Integervariable])

CHR$ ist die Umkehrfunktion von ASC und wandelt ASCII-Code-Werte von 0 bis 255 in die entsprechenden Zeichen bzw. Funktionen um. Siehe auch Abschnitt 3.2. KONTROLLFUNKTIONEN.

```
Beispiel: PRINT CHR$(66)¬
          B
```

Die Verwendung von unzulässigen Werten wird mit der Fehlermeldung "Falscher Funktionsaufruf" quittiert.

HEX$ HEX$([Zahl]) HEX$([Integervariable])

Konvertiert INTEGER-Zahlen in ihr hexadezimales Äquivalent und stellt sie als Strings zur Verfügung.

```
Beispiel: 10 A% = 64
          20 A$ = HEX$(A%)
          30 PRINT A$
          RUN¬
          FF
```

INSTR INSTR([Anfangsposition], [String], [Suchstring])

INSTR sucht einen [Suchstring] beginnend mit der [Anfangsposition] in dem spezifierten [String]. Als Ergebnis wird die Position geliefert, die dem Beginn der Übereinstimmung von [String] und [Suchstring] entspricht.

```
Beispiel: 10 ST$ = "       Computersysteme"  ← [String]
          20 SS$ = "systeme"                 ← [Suchstring]
          30 A1% = 2                         ← Beginn der
          40 A2$ = 19                        ← Suche beim 2.
                                               bzw. beim 19.
                                               Zeichen

          50 E1% = INSTR(A1%, ST$, SS$)
          60 E2% = INSTR(A2%, ST$, SS$)
          70 PRINT E1%, E2%
          RUN¬
           14          0
```

Da beim zweiten Suchbefehl keine Übereinstimmung gefunden wurde, lautet hier das Ergebnis "0".

LEFT$ LEFT$([String] , [Anzahl der Zeichen])

Liefert, beginnend mit dem ersten Zeichen von [String], einen Teilstring der Länge [Anzahl der Zeichen].

```
Beispiel: 10 ST$ = "LEFT$ DEMO"
          20 TS$ = LEFT$(ST$,5)
          30 PRINT TS$
          RUN¬
          LEFT$
```

LEN LEN([String]) LEN([Stringvariable])

Liefert die Länge von [String] bzw. [Stringvariable].

```
Beispiel: PRINT LEN("123456789")¬
          9
```

MID$ MID$([String],[Anfangsposition], [Länge])

Liefert, beginnend mit der [Anfangsposition], einen Teilstring der spezifizierten [Länge].

```
Beispiel: PRINT MID$("Computeritis perniciosa", 9, 4)¬
          itis
```

OCT$ OCT$([Integerzahl]) OCT$([Integervariable])

Konvertiert INTEGER-Zahlen in ihr oktales Äquivalent und stellt sie als Strings zur Verfügung.

```
Beispiel: 10 A% = 16: B% = 33
          20 A$ = OCT$(A%): B$ = OCT$(B%)
          30 PRINT A$, B$
          RUN¬
          20          41
```

PAR PAR([Stringvariable])

Prüft, ob eine Stringvariable einen Leerstring darstellt oder ob sie Zeichen enthält. Ist die Variable leer, wird "0", andernfalls "1" geliefert.

```
Beispiel: 10 A$ = ""
          20 B$ = "VOLL"
          30 IF PAR(A$) = 0 THEN PRINT "Leerstring"
             ELSE PRINT B$
          RUN¬
          Leerstring
```

RIGHT$ RIGHT$([String, Anzahl der Zeichen])

Liefert, beginnend mit dem äußersten rechten Zeichen, einen Teilstring der Länge [Anzahl der Zeichen].

```
Beispiel: PRINT RIGHT$("Demo-String", 6)¬
          String
```

SPACE$ SPACE$([Anzahl der Leerzeichen])

Erzeugt eine durch [Anzahl der Leerzeichen] definierte Menge von Leerzeichen, beginnend bei der aktuellen Cursorposition.

```
Beispiel: 10 PRINT "A";
          20 PRINT SPACE$(20);
          30 PRINT "B"
          RUN¬
          A                    B
```

STR$ STR$([Zahl]) STR$([Zahlenvariable])

Bildet eine Zahl bzw. den Inhalt einer Zahlenvariable als String ab und stellt sie in dieser Form für die Weiterverarbeitung zur Verfügung.

```
Beispiel: 10 A% = 10: B% = 20
          20 A$ = STR$(A%): B$ = STR$(B%)
          30 PRINT A% + B%
          40 PRINT A$ + B$
          RUN¬
           30
          1020
```

STRING$ STRING$([Zeichenanzahl], [ASCII-Code])

Liefert einen String bestehend aus einer durch [Zeichenanzahl] bestimmten Menge des durch seinen ASCII-Code spezifizierten Zeichens.

```
Beispiel: PRINT STRING$(10, 42)¬
          **********
```

VAL VAL([Zahlenstring])

Ermittelt den numerischen Wert eines Strings, der aus Zahlen besteht. Enthält der String andere Zeichen als zur Darstellung einer Zahl erforderlich, wird "0" geliefert.

```
Beispiel: 10 A$ = "123": B$ = "456"
          20 A% = VAL(A$ + B$)
          30 PRINT A%
          RUN¬
           123456
```

SCRN$ SCRN$([X-Koordinate],[Y-Koordinate],[Länge])

Die SCRN$-Funktion liest, beginnend bei den Cursor-Koorditen X und Y, einen String der spezifizierten [Länge] vom Bildschirm und weist ihn einer Variablen zu.

Beispiel:
```
10 CURSOR 10, 10
20 PRINT "Sehr geehrter Herr Kohl"
30 CURSOR 15, 15
40 PRINT "Sehr geehrter Herr Strauß"
50 A$ = SCRN$(10, 29, 4)
60 B$ = SCRN$(15, 34, 6)
70 C$ = "Blumen sind schön !"
80 A$ = LEFT$(C$, 6) + "k" + RIGHT$(A$, 3)
90 B$ = LEFT$(C$, 6) + "s" + RIGHT$(B$, 5)
100 PRINT A$; " statt ";B$
RUN↵
(Lassen Sie sich überraschen!)
```

Diese Funktion erweist sich bei der Maskenverarbeitung als besonders vorteilhaft. Folgender Programmablauf ist damit möglich:

a. Maskenaufbau

b. Bildschirmorientiertes Füllen der Maske durch den Benutzer

c. Lesen und Zuweisung aller Eingaben mit Hilfe der SCRN$-Funktion.

d. Auswertung der Eingaben durch das Programm.

Diese Eingabeart macht INPUT- und INKEY$-Anweisungen und die damit verbundenen umfangreichen Korrekturroutinen überflüssig.

12. SONDERFUNKTIONEN

Außer den bisher beschriebenen bietet SuperBASIC eine Reihe ungewöhnlicher Sonderfunktionen. Diese sind:

KEY ON

Die Anweisung KEY ON versetzt den Rechner in die Lage, Eingaben während des Programmlaufs anzunehmen -- auch dann, wenn keine INPUT- oder INKEY$-Anweisung vorliegt. Die Eingabe erfolgt "blind", d.h., vorerst ohne Bildschirmanzeige. Sie wird zwischengespeichert und erst in folgenden Fällen auf dem dem Bildschirm eingeblendet:

1. Wenn das Programm auf eine echte INPUT-Anweisung trifft. Der zwischengespeichterte Text erscheint genau so, als ob er erst jetzt über die Tastatur eingegeben wurde:

   Beispiel:
   ```
   10 KEY ON
   20 FORI = 1 TO 5000: NEXT I
   30 PRINT "Diese Eingabe erfolgte während er
      Warteschleife"
   40 INPUT A$
   RUN¬
   Diese Eingabe erfolgte während der Warteschleife:
   ? HALLO ⊠
        ↑
   Diese Tasten wurden gedrückt, während die
   Schleife abgearbeitet wurde. Das Programm
   wartet mit blinkendem Cursor auf eine Eingabe-
   bestätigung über die [CR]-Taste.
   ```

2. Nach der Programmausführung, wenn nach KEY ON keine INPUT-Anweisung im Programm steht. In diesem Fall wirkt die blinde Eingabe, wenn mit sie mit [CR] abgeschlossen wurde, wie eine Eingabe im Direktmodus:

   Beispiel:
   ```
   10 KEY ON
   20 FOR I = 1 TO 5000: NEXT I
   RUN¬
   LIST
   10 KEY ON
   20 FOR I = 1 TO 5000: NEXT I
   ```

   In diesem Fall wurde während des Schleifendurchlaufs das Wort "LIST" blind eingegeben und die [CR]-Taste gedrückt.

Nach einer KEY ON-Anweisung können auch einige Funktionstasten während des Programmlaufs eingesetzt werden:

↑[P] = Bildschirm-Hardcopy. Die Programmausführung wird unterbrochen, während der Bildschirminhalt auf dem Drucker ausgegeben wird.

↑[S] = Stop-Taste. Die Programmausführung wird bis zum nächsten Tastendruck unterbrochen.

↑[V] = RVS-Modus einschalten.

↑[W] = GRPH-Sperre einschalten.

↑[O] = Bildschirm invertieren.

KEY OFF

Hebt die KEY ON-Funktion wieder auf.

KEY O KEY O, [String 1] + [String 2]

Die KEY O-Anweisung ermöglicht die Erstellung von Programmen, die sich selbst erweitern können, zum Beispiel dadurch, daß Benutzereingaben automatisch in zeilenrichtig eingeordnete DATA-Anweisungen umgewandelt werden.

Diese Anweisung ist so leistungsfähig, daß man mit ihrer Hilfe sogar ein BASIC-Programmgenerator erstellen könnte.

Damit Programmzeilen generiert werden können, macht KEY O bei jeder Anwendung einen "Ausflug" in den Direktmodus. Um einen Rücksprung in den Programmodus zu bewerkstelligen, muß die Anweisung so formuliert werden, daß sie mit einem "GOTO [Zeilennummer]" bzw. mit einer "RUN [Zeilennummer]" abgeschlossen wird. Wenn man dies versäumt, kann unter Umständen die völlige Zerstörung des Programms die Folge sein.

Da der Befehl KEY O eine Benutzereingabe simuliert, kann er auch zum überspringen von INPUT-Anweisungen verwendet werden. Hierzu muß der einzugebende String so formatiert werden:

```
KEY O, "[String]" + CHR$(13)
```

Folgendes Beispielprogramm erzeugt aus Eingaben des Benutzers DATA-Zeilen mit fortlaufender Nummerierung.

Beispiel:
```
 10 CLS: ZN=990
 20 ZN = ZN+10: PUSH ZN: CURSOR 0,10
 30 LINEINPUT "Daten: ";DA$
 40 DA$ = RIGHT$(DA$,LEN(DA$)-8): ZN$ = STR$(ZN)
 50 KEY O, ZN$+"DATA"+DA$+CHR$(13)+"GOTO70"+CHR$(13)
 60 END:REM ***Notwendig zur Ausführung von KEY O***
 70 POP ZN: CLS: CURSOR 0,10: PRINT "Weitere Eingaben ?";
 80 IF INKEY$(0)="N"GOTO 100 ELSE IF INKEY$(0)="J"GOTO 20
 90 GOTO 80
100 CLS: LIST: END
```

Ganz allgemein sind folgende Punkte bei der Verwendung von KEY O bei der Programmzeilengenerierung zu beachten:

1. Da SuperBASIC nach dem Einfügen einer neuen Zeile, bedingt durch die neue Speicherverteilung, alle Variablen vergisst, muß man Variablen, die weiterverwendet werden mit PUSH auf den Stapelspeicher legen und nach KEY O mit POP zurückholen.

2. KEY O kann nur dann eine Programmzeile erzeugen, wenn das Programm beendet wurde. Deshalb muß nach der KEY O-Befehlsfolge das Programm mit END abgeschlossen werden. Danach kann es durch Anhängen des Befehls "GOTO [Zeilennummer]" an die restliche KEY O-Folge weitergeführt werden.

3. Eine KEY 0-Anweisung ohne nachfolgendes Programmende hat das selbe Ergebnis wie eine Blindeingabe während eines Programmlaufs nach KEY ON (siehe auch dort). KEY ON-Eingaben benützen den selben Zwischenspeicher wie KEY 0, daher besteht die Möglichkeit, Eingaben, z.B. bei INPUT-Anweisungen , programmgesteurt zu simulieren. Damit lassen sich beispielsweise überflüssig gewordene Eingabeanweisungen trotz Bildschirmanzeige überspringen.

4. Desweiteren läßt sich KEY 0 dazu verwenden, Befehle, die normalerweise nur im Direktmodus zulässig sind, dem Programm zur Verfügung stellen.

5. Bei der Ausführung von KEY 0-Strings sind außer CHR$(13) als Startbefehl auch alle anderen Kontrollcodes zulässig, jedoch nicht immer sinnvoll.

6. Überprüfen Sie Programme, die KEY 0 enthalten, besonders sorgfältig, bevor Sie sie starten. Fehlprogrammierungen führen sehr leicht zum "Absturz" des gesamten Systems!

Der KEY - Puffer, das heißt der Bereich, der bei den Funktionen KEY ON, KEY 0, INPUT, LINEINPUT und INKEY$ alle Eingabedaten speichert, kann entweder durch eine bezuglose INPUT-Anweisung oder durch die Folge KEY 0, "" gelöscht werden. Dies ist notwendig, um eventuell folgende INPUT-Anweisungen von KEY-Funktionen freizuhalten.

SUB SUB

Die Anweisung MON übergibt die Systemkontrolle an einen eingebauten SUB-MONITOR. Dieser ist mit umfangreichen Befehlen zum "debuggen" von Maschinenprogrammen, sowie zu deren Erstellung und Sicherung versehen.

Der Befehlsumfang des Submonitors gliedert sich in vier Teile:

1. Befehle zur Darstellung von Speicherinhalten auf Bildschirm oder Drucker. Das Format der Ausgabe ist wählbar.

2. Lade- und Schreibbefehle zum Lesen und Aufzeichnen vom Maschinenprogrammen. Eine Verifikationsroutine wird ebenfalls zur Verfügung gestellt.

3. Befehle zur Ablaufkontrolle von Maschinenprogrammen. Dazu kann ein Breakpoint gesetzt und und damit ein Programm zur Prüfung des Programmlaufs unterbrochen werden.

4. Befehle zur Manipulation von Speicherinhalten. Als Eingabe sind 8- und 16-Bit-Werte oder ASCII-Zeichen möglich.

Als Zusatz wurde ein Befehl zur direkten Druckeransteuerung implementiert. Er erlaubt die Ausgabe der Werten von 00H bis FFH an den Drucker. Das Handshaking wird vom Submonitor übernommen.

Die Befehle werden in nachfolgender Tabelle erläutert:

D D[n]␣[Anfangsadresse]␣[Endadresse]

Diesee Befehl stellt einen Speicherbereich von [Anfangsadresse] bis [Endadresse] auf dem Bildschirm dar. Pro Zeile werden [n] Bytes ausgegeben.

Die Darstellung umfaßt die Adressenangabe (Adresse des ersten Bytes einer Zeile), die gewählte Anzahl Bytes in hexadezimaler Darstellung und, sofern darstellbar, deren ASCII-Äquivalente.

Die Auflistung kann mit der [SPACE]-Taste unterbrochen und mit einem beliebigen Tastendruck fortgesetzt werden. Die [SPACE]-Taste ermöglicht auch einen Einzelschrittmodus beim Auflisten.

P P[n]␣[Anfangsadresse]␣[Endadresse]

Befehlsformat und Ergebnis wie oben, jedoch mit Druckerausgabe.

M M[Anfangsadresse]

Mit M kann man, beginnend mit [Anfangsadresse], Speicherstellen modifizieren. Zuerst werden die zu modifizierende Speicherstelle ihren Inhalt angezeigt. Es kann nun entweder ein neuer Wert eingegeben werden oder durch einen Druck auf die [CR]-Taste die nächste Adresse gewählt werden. In diesem Fall bleibt der Inhalt der übersprungenen Adresse erhalten. Im M-Modus können pro Zeile mehrere Bytes, getrennt durch ein Leerzeichen, eingegeben werden.

Auch die direkte Eingabe von ASCII-Zeichen ist möglich. Dazu werden anstelle eines Bytes ein Semikolon, das gewünschte Zeichen und eine Leerzeichen eingegeben. Nach einem Druck auf die [CR]-Taste wird das Zeichen in einen ASCII-Wert umgewandelt.

Die M-Anweisung eignet sich auch zur Berechnung relativer Sprungweiten. Dazu wird an der entsprechenden Speicherstelle, statt eines aboluten Werts die Zieladresse angegeben. Diese Adressenangabe muß als Präfix ein [.]-Zeichen enthalten.

Beispiel:
```
*MF000↵
F000=00 ←——— alter Wert der Speicherstelle
     ↑
     └——————— Cursor blinkt hier

     FA↵←——— Eingabe ab Cursorposition

F001=00 ←——— Inhalt der nächsten Speicherstelle

     ;A↵←——— Eingabe des ASCII-Zeichens A in die
              Speicherstelle F001. A wird in 41H
              umgewandelt

F002=00 ←——— Inhalt der nächsten Speicherstelle

     20 0D ←— Eingabe von zwei Bytes

F004=00 ←——— Inhalt der nächsten Speicherstelle

     .F000 ←— Eingabe eines relativen Sprungs nach
              F000H. SuperBASIC setzt dazu den
              Wert FBH in die Speicherstelle ein
```

Der Programmiervorgang kann durch Betätigen von [SHIFT]+[BREAK] beendet werden. Fehleingaben werden durch Überschreiben und einen Druck auf die [CR]-Taste korrigiert.

T T[Anfangsadresse]␣[Endadresse]␣[Ziel]

Mit dem T-Befehl läßt sich ein Speicherbereich von [Anfangsadresse] bis [Endadresse] in einen Bereich, beginnend mit [Ziel], verschieben.

Beispiel: *TF000 F100 F200↵ ← Verschiebt einen Speicherblock von F000h bis F100h nach F200h

S S[Anfangsadresse]␣[Endadresse]␣[Wert1]␣[Wert2]....

Der Befehl [S] dient zum Suchen eines Bytes oder einer Bytekombination (z.B. [Wert1]) im Bereich von [Anfangsadresse] bis [Endadresse]. Als Ergebnis liefert SuperBASIC, wenn entsprechende Werte gefunden werden, Adresse, Wert (bzw. Kombination) und, wenn darstellbar, das ASCII-Äquivalent des Werts (bzw. der Kombination). Es werden alle Adressen aufgelistet, die die gesuchten Werte enthalten.

Beispiel: *SF000 F100 00 20↵ ← Sucht die Kombination 00 20 im Bereich von F000h bis F100h

O O=[Wert1]␣[Wert2]␣...␣[Wertn]

Mit [O] können beliebige Werte im Bereich 00h bis FFh an den Drucker ausgegeben werden. Die notwendige Handshake-Prozedur wird automatisch durchgeführt.

Beispiel: *O=41 42 43 44 0D 0A↵ ← Gibt die Zeichen A, B, C, D an den Drucker und erzeugt einen CRLF

R R

Bewirkt einen Rücksprung in die BASIC-Befehlsebene.

G G[Sprungadresse]

Bewirkt die Übergabe der Programmkontrolle an ein Programm, spezifiziert mit [Sprungadresse]

W W[Ladeadresse]␣[Endadresse]␣[Autostart]␣[Name]

Der W-Befehl dient zur Aufzeichnung von Maschinenprogrammen auf Kassette.

Beispiel: *WF000 F200 F010 Testpro↵

Obige Befehlsfolge bewirkt die Aufzeichnung eines Programms aus dem Speicherbereich F000h bis F200h. Dieses Programm hat eine Autostartadresse von F010h und den Namen [Testpro].

L L␣[Programmname]

Lädt ein Programm des Namens [Programmname] von der Kassette. [Programmname] kann weggelassen werden.

V V␣[Programmname]

Prüft die ordnungsgemäße Aufzeichnung eines Programms.

B B[Adresse] B

Mit dem B-Befehl kann in einem beliebigen Maschinenprogramm ein Breakpoint gesetzt werden. Dieser dient dazu, das Programm zu unterbrechen, um den aktuellen Inhalt der CPU-Register festzustellen. Die Ausgabe der CPU-Register erfolgt nach folgendem Muster:

```
        AF    BC    DE    HL
        ┌─┐   ┌─┐   ┌─┐   ┌─┐
00A0=ff aa cc bb ee dd ll hh / [ASCII-Äquivalente]
00A8=xl xh yl yh pl ph sl sh / [ASCII-Äquivalente]
       └─┘   └─┘   └─┘   └─┘
        IX    IY    PC    SP
```

Der Breakpoint kann durch Eingabe von B ohne Argument wieder gelöscht werden

Beispiel: *BF010↵ ←— Setzt einen Breakpoint auf die Adresse F010h

*GF000↵ ←— Startet die Programmausführung an der Adresse F000h

00A0=00 07 bb 02 06 17 14 F0 /
00A8=00 00 66 67 11 F0 12 14 /

*↵ ←— Rückmeldung des Systems. Das Programm wurde an der Adresse F010 unterbrochen

*B↵ ←— Löscht Breakpoint

I Entspricht KEY ON in BASIC. Erlaubt das Zwischenspeichern von Tastatureingaben, ohne direkt programmierte Eingabeanweisungen. Die Eingabe erfolgt vorläufig ohne Bildschirmanzeige. Die eingegebenen Zeichen erscheinen erst dann auf dem Bildschirm, wenn:

1. Eine Eingabanweisung eines Maschinenprogramms ausgeführt wird.

2. Der Submonitor wieder auf seine Befehlsebene zurückkehrt.

Für Näheres siehe KEY ON

N Entspricht KEY OFF in SuperBASIC (siehe dort).

Der Submonitor führt zwei Pointer mit, die die bei Befehlen zuletzt verwendeten Werte enthalten:

1. Adresspointer: Enthält die zuletzt bearbeitete Speicheradresse + 1

2. Formatpointer: Zeigt das zuletzt verwendete Format auf

Beim mehrfachen Aufruf der Befehle kann man die schon durch die Pointer festgelegten Werte wegzulassen. Der Submonitor arbeitet dann beginnend mit den Pointerwerten weiter.

Man kann auch die Endadresse bei auflistenden Funktionen weglassen. Der Submonitor arbeitet dann 128 Bytes ab und meldet sich zurück.

13. SCHLEIFENKONSTRUKTIONEN

Ein besonderes Merkmal von SuperBASIC ist seine Fähigkeit verschiedene Schleifenkonstruktionen zu verarbeiten. Neben der in BASIC üblichen FOR...NEXT-Anweisung findet man REPEAT... UNTIL und sogar WHILE...WEND. Hiermit lassen sich umfangreiche IF...THEN...ELSE-Bedingungsprüfungen vermeiden, wodurch das Programmieren vereinfacht und die Programmausführung beschleunigt werden. Dies bedeutet auch, daß man im "top-down"-Verfahren genau durchstrukturierte Programme erstellen kann.

In folgenden Beispielprogrammen wird aus einem einzugebenden String die Häufigkeit ermittelt, mit der ein bestimmter Buchstabe vorkommt. Dieses Problem wird mit Hilfe der oben genannten Anweisungen gelöst. Anhand des jeweiligen Lösungswegs sind die Vor- und Nachteile der einzelnen Schleifenkonstruktionen zu erkennen.

Beispiel 1. FOR...NEXT-Schleife

```
 10  CLS: DEF INT X
 20  CURSOR 0,10
 30    LINE INPUT "String eingeben: "; A$
 40    ST$ = RIGHT$(A$, LEN(A$) - 17)
 50  CURSOR 0,14
 60    PRINT "Welches Zeichen soll gesucht werden?"
 70      Z$ = INKEY$(1)
 80      X = 0
 90    FOR I = 1 TO LEN(ST$)
100      IF MID$(ST$, I, 1) = Z$ THEN X = X + 1
110    NEXT I
120  CURSOR 0,18
130    PRINT Z$; " kommt"; X; "Mal vor."
140  END
```

Beispiel 2. REPEAT...UNTIL-Schleife

```
 10  CLS: DEF INT X, Z
 20  CURSOR 0,10
 30   LINE INPUT "String eingeben: "; A$
 40   ST$ = RIGHT$(A$, LEN(A$) - 17)
 50  CURSOR 0,14
 60    PRINT "Welches Zeichen soll gesucht werden?"
 70       Z$ = INKEY$(1)
 80       X = 0 : Z =0
 90    REPEAT
100    Z = Z+1: IF MID$(ST$,Z,1) = Z$ THEN X = X + 1
110    UNTIL Z = LEN(ST$)
120  CURSOR 0,18
130    PRINT Z$; "kommt"; X; "Mal vor."
140  END
```

Beispiel 3. WHILE...WEND-Schleife

```
 10  CLS: DEF INT X, Z
 20  CURSOR 0, 10
 30    LINE INPUT "String eingeben: "; A$
 40    ST$ = RIGHT$(A$, LEN(A$) - 17)
 50  CURSOR 0, 14
 60    PRINT "Welches Zeichen soll gesucht werden?"
 70      Z$ = INKEY$(1)
 80      X = 0: Z = 0
 90    WHILE Z < LEN(ST$)
100      Z = Z + 1: IF MID$(ST$, Z, 1) = Z$ THEN X = X + 1
110    WEND
120  CURSOR 0, 18
130    PRINT Z$; " kommt"; X; "Mal vor."
140  END
```

In der täglichen Programmierpraxis empfiehlt es sich, die für die aktuelle Problemstellung günstigste Schleifenkonstruktion zu verwenden. Die Vorteile der REPEAT...UNTIL- und WHILE...WEND-Anweiweisungen kommen am besten in verschachtelten Bedingungsprüfungen zur Geltung. Bei der Erstellung von strukturierten Programmen lassen sie sich besonders elegant in Subroutinen verwenden, die mit GOSUB "LABEL" wie Prozeduren aufgerufen werden.

Bei der Entscheidung über die Verwendung von WHILE...WEND bzw. REPEAT...UNTIL muß sich der Programmierer darüber im Klaren sein, ob der eingeschlossene Programmteil vor oder nach der Bedingungsprüfung bearbeitet werden soll. Normalerweise hängt die Entscheidung von folgenden Kriterien ab:

1. Wenn die Bedingungsschleife immer mindesten einmal durchlaufen werden muß, sollte REPEAT...UNTIL verwendet werden, da die Bedingungsprüfung erst am Ende des Schleifendurchlaufs erfolgt.

2. Muß die Schleife nur dann durchlaufen werden, wenn die Bedingung, die sie einleitet, nicht erfüllt ist, sollte man der WHILE...WEND-Anweisung den Vorzug geben. Ist die Bedinggung schon gegeben, wird die Schleife übersprungen, wodurch die Verarbeitungszeit verkürzt wird. Hieraus wird ersichtlich, daß WHILE...WEND sich besonders in Subroutinen bewährt, die sporadisch zum Angleichen von Daten herangezogen werden.

14. FEHLERMELDUNGEN

SuperBASIC verfügt über 32 Klartext-Fehlermeldungen. Für die Fehlerverarbeitung im Programm sind die entsprechenden Fehlernummern zu verwenden.

Nr.	Klartext	Bedeutung
1	NEXT ohne FOR	Die FOR-Anweisung wurde in einer FOR...NEXT-Schleife vergessen.
2	Syntaxfehler	Eine für SuperBASIC nicht verständliche Konstruktion wurde verwendet.
3	RETURN ohne GOSUB	Das Programm trifft auf eine RETURN-Anweisung, zu der die entsprechende GOSUB-Anweisung fehlt.
4	Daten-Ende	Es wurde versucht, mit dem READ-Befehl nicht vorhandene DATA-Anweisungen in das Programm zu lesen.
5	Falscher Funktionsaufruf	Ein nicht zulässiger Parameter wurde an eine mathematische bzw. an eine String-Funktion übergeben.
6	Ueberlauf	Das Ergebnis einer Berechnung überschreitet den zulässigen Rechenbereich.
7	Speicherueberlauf	Das Programm hat die verfügbare Speicherkapazität überschritten.
8	Zeilennummer fehlt	Es wird versucht, in eine nicht existente Programmzeile zu springen.
9	Dimensionierungsfehler	Ein angesprochenes Matrixelement liegt außerhalb des dimensionierten Bereichs, oder die Matrixdefinition ist fehlerhaft.
10	Doppeldefinition	Es wird versucht, eine bereits dimensionierte Matrix noch einmal zu definieren.
11	Divisor = 0	Es wird versucht, einen numerischen Wert durch Null zu teilen.
12	Unzulaessiger Befehl	Ein Befehl, der nur im Programmmodus zur Verwendung kommt, wurde im Direktmodus eingegeben.

Nr.	Klartext	Bedeutung
13	Typenfehler	Ein Stringvariablenname wurde einem numerischen Wert zugewiesen oder umgekehrt.
14	Kommt nie vor	Die verwendete Konstruktion kann im gegebenen Kontext nie vorkommen.
15	Stringueberlauf	Der verwendete bzw. ermittelte String ist länger als 240 Zeichen
16	Ausdruck zu komplex	Das Programm trifft auf einen Ausdruck (normalerweise eine Formel), der zu lang oder zu kompliziert ist.
17	Geht nicht	Es wurde versucht, mit dem CONT-Befehl ein Programm fortzusetzen, das: a) nach einem BREAK modifiziert wurde; oder b) wegen eines Fehlers abgebrochen wurde.
18	Nicht definierte USR-Funktion	Eine nicht definierte Benutzerfunktion wurde aufgerufen.
19	RESUME fehlt	Die RESUME-Anweisung wurde bei der Fehlerverarbeitung weggelassen.
20	RESUME ohne ERROR	Das Programm trifft auf eine RESUME-Anweisung, für die die Fehlerverarbeitungsanweisungen fehlen.
21	Formatfehler	Das Programm trifft auf eine falsch formatierte PRINT USING-Anweisung.
22	Operand fehlt	Es wird versucht, eine Operation durchzuführen, auf deren Operator kein Operand folgt.
23	REPEAT ohne UNTIL	Nach einer REPEAT-Anweisung fehlt die für den Schleifenabschluß erforderliche UNTIL-anweisung.
24	UNTIL ohne REPEAT	Es wurde versäumt, die REPEAT-Anweisung am Anfang einer REPEAT...UNTIL-Schleife zu setzen.
25	WHILE ohne WEND	In einer WHILE...WEND-Schleife fehlt die WEND-Anweisung.

Nr.	Klartext	Bedeutung
26	WEND ohne WHILE	Die WHILE-Anweisung wurde bei der Bildung einer WHILE...WEND-Schleife vergessen.
27	Bandlesefehler	Ein Fehler tritt auf beim Versuch, Daten von einer Kassette zu lesen.
28	Peripheriefehler	Ein angesprochenes Peripheriegerät ist nicht angeschlossen oder in einem Fehlerzustand.
29	Befehl nicht installiert	Der verwendete Befehl ist in dieser SuperBASIC-Version noch nicht verwendbar.
30	Stack leer	Es wurde versucht mit einer POP-Anweisung auf den leeren Stapelspeicher zuzugreifen.
31	Filefehler	Eine angesprochene Datei ist nicht lesbar.
32	Fehler	Das Programm befindet sich in einem Fehlerzustand, für den keine der oben genannten Fehlermeldungen zutrifft.

14. FEHLERVERARBEITUNG

SuperBASIC bietet umfangreiche Möglichkeiten zur Fehlererkennung und -beseitigung. Darüberhinaus lassen sich auch Fehler simulieren, beispielsweise, um das Verhalten von Programmen bei Benutzerirrtümern zu testen.

1. Das Prinzip der Fehlerbehandlung:

   Jede Fehlerbedingung, die während des Programmlaufs auftritt, führt im Normalfall zum Abbruch des Programms und zu Ausgabe einer Klartextfehlermeldung. Dies kann in vielen Fällen nicht sinnvoll sein, z.B. bei Fehleingaben, Datenüberlauf oder einer Druckerstörung. SuperBASIC kann dazu veranlaßt werden, in solchen Fällen in ein Unterprogramm zu springen, in dem die Fehlerursache behoben wird. Danach wird der Programmlauf an der Stelle fortgesetzt, die den Fehler verursachte.

2. Umfang der Fehlerbeseitigung mit SuperBASIC

   Fehler können natürlich nur dann beseitigt werden, wenn die Fehlerursache nicht auf prinzipielle Programmierfehler zurückzuführen ist. So lassen sich z.B. Syntaxfehler nur durch entProgrammänderungen beseitigen.

3. Programmierung

   Soll ein Programm mit Fehlerbehandlung ausgerüstet werden, sollte die entsprechende Anweisung bereits in der ersten Programmzeile stehen. Die Anweisung lautet:

   ON ERROR GOTO [Zeilennummer]

   Damit wird Superbasic angewiesen, nach dem Auftreten eines Fehlers die Klartextfehlermeldung zu unterdrücken und anstelle des Programmabbruchs zu einem Programmteil, beginnend mit [Zeilennummer], zu springen.

   Die beiden Pseudovariablen ERR und ERL enthalten dann Informationen über Fehlerart und -ort:

   ERR enthält die der Klartextfehlermeldung entsprechende Fehlernummer im Bereich zwischen 1 und 32 (siehe Fehlertabelle).

   ERL enthält die Nummer der Zeile, in der der Fehler auftrat.

   Tritt nun ein Fehler auf, z.b. durch eine falsche Eingabe, kann im durch "ON ERROR GOTO ..." angesprungenen Programm der Benutzer auf eine korrekte Eingabe hingewiesen werden. Danach kann SuperBASIC die Fehlerbedingung wieder aufheben und ins Hauptprogramm zurückkehren. Die Anweisungen dazu lauten:

   RESUME RESUME [Zeilennummer] RESUME NEXT

Die Anweisung RESUME dient dem Aufheben der Fehlerbedingung. Damit ist natürlich nicht die Fehlerursache beseitigt. Dazu kann folgendermaßen vorgegangen werden:

a. Es wird, eventuell nach entsprechendem Hinweis an den Benutzer, nur die Anweisung RESUME gegeben. SuperBASIC wiederholt dann noch einmal die fehlerverursachende Anweisung. Tritt nun kein Fehler mehr auf, wird das Programm fortgesetzt, andernfalls wird erneut in die Fehlerroutine gesprungen.

b. Es wird, z.B. nach Anweisungen an den Benutzer oder einem Programmteil zur Fehlerbeseitigung, die Anweisung RESUME [Zeilennummer] gegeben. Die Fehlerbedingung wird aufgehoben und die Abarbeitung des Programms wird, beginnend mit [Zeilennummer], wieder aufgenommen. Diese Form der RESUME-Anweisung ist dann sinnvoll, wenn die Fehlerursache nicht beseitigt werden kann und das Programm an anderer Stelle fortgesetzt werden muß.

c. Eine Vereinfachung von RESUME [Zeilennummer] ist die Anweisung RESUME NEXT. Die Ausführung ist fast wie in Punkt b , das heißt die fehlerverursachende Anweisung wird nicht erneut bearbeitet, sondern das Programm wird mit der nachfolgenden Anweisung fortgesetzt.

Fehlersimulation

SuperBASIC erlaubt es, Fehler an beliebiger Stelle des Programms zu simulierten. Das kann bei der Programmerstellung von Nutzen sein, um festzustellen, ob die Fehlerbehandlugsroutinen richtig arbeiten. Die Anweisung dazu lautet:

ERROR [Fehlernummer]

SuperBASIC leitet dann die der [Fehlernummer] entsprechende Fehlerbedingung ein. Wird ERROR [Fehlernummer] im Direktmodus angewandt, gibt SuperBASIC die jeweilige Klartextfehlermeldung aus.

Beispiel:
```
10 ON ERROR GOTO 1000
20 INPUT"Ihre Eingabe: "; A
30 PRINT 3/A
40 GOTO 20

1000 IF ERR = 13 THEN PRINT"Fehler !!  Nur Zahleneingabe !":
     RESUME
1010 PRINT "Fehler Nummer ";ERR;" in Zeile ";ERL
1020 RESUME 20
```

Wenn Sie bei obigem Beispiel anstelle einer Zahl einen Buchstaben eingeben, wird SuperBASIC "Fehler !! Nur Zahleneingabe !" ausgeben und eine neue Eingabe verlangen. Geben Sie jedoch als Wert eine Null ein, wird "Fehler Nummer 11 in Zeile 30 " auf dem Bildschirm erscheinen. Danach wird das Proramm mit Zeile 20 fortgesetzt.

Gerade das obige Beispiel zeigt die Nützlichkeit der Fehlerbehandlung, da zur Absicherung von mehreren INPUT-Anweisungen nur noch eine Routine nötig ist. Da eine Prüfung nach jeder INPUT-Anweisung entfallen kann, erhält man kürzere und damit schnellere Programme.

16. BEFEHLSABKÜRZUNGEN

Um die Tipperei bei der Programmerstellung auf ein Minimum zu halten, wurde SuperBASIC mit der Fähigkeit versehen, abgekürzte Befehle und Anweisungen zu verarbeiten. Da die Abkürzungen intern enstschlüsselt werden, erscheinen die Befehle und Anweisungen beim Auflisten eines Programms wieder in voller Länge.

Grundsätzlich ist zu bemerken, daß alle Befehlsabkürzungen mit einem Punkt [.] enden. R. bedeutet RUN, E. bedeutet EDIT, usw.

Da der Punkt auch als Zeiger auf die aktuelle Zeilennummer verwendet wird, kann es vorkommen, daß man Abkürzungen mit zwei darauffolgenden Punkten eingibt. Nehmen wir an, ein gerade laufendes Programm wird mit der Meldung "Syntaxfehler in 15110" abgebrochen. Wenn man die fehlerhafte Zeile a la SHARP-BASIC ausbessern will, muß man folgendermaßen vorgehen:

1. LIST 15110↵ eingeben.
2. Cursor in die Zeile bringen.
3. Cursor bis auf den Fehler bewegen.
4. Fehler ausbessern und [CR] drücken.

In SuperBASIC werden Schritt 1 und 2 mit der blossen Eingabe E..↵ erledigt, wo E. = EDIT und . = aktuelle Zeilennummer.

Die einzelnen SuperBASIC-Abkürzungen lassen sich selbstverständlich auch in zusammengesetzten Anweisungen des Typs ON ERROR GOTO (O.ER.G.) verwenden.

Die Kurzformen der SuperBASIC-Wörter bitten wird der nach folgenden Tabelle zu entnehmen.

Wort	Abk.	Wort	Abk.
ABS	AB.	OCT$	OC.
ASC	AS.	OFF	OF.
ATN	AT.	ON	O.
AUTO	A.	ON ERROR GOTO	O.ER.G.
BEEP	B.	PAINT	PA.
CALL	CA.	PEEK	PE.
CDBL	CD.	PEEK@	PE.@
CHADR	CH.	PLOT	PLO.
CINT	CIN.	POINT	POI.
CIRCLE	CI.	POKE	PO.
CLEAR	CLE.	POKE@	PO.@
CLOSE	CLO.	PRESET	PRE.
CLS	CL.	PRINT	P.; ?
CMT	CM.	PRINT USING	P.US.; ?US.
COLOR	COLO.	PSET	PS.
COLSET	COL.	PUSH	PU.
CONT	C.	RAD	RA.
CSGN	CS.	REM	'
CSRLIN	CSR.	RENUM	REN.
CURSOR	CU.	REPEAT	REP.
DATA	DA.	RESTORE	RES.
DELETE	D.	RESUME	RESU.
DUMP	DU.	RETURN	RE.
EDIT	E.	RIGHT$	RI.
ELSE	EL.	RND	RN.
ERROR	ER.	ROPEN	RO.
EXP	EX.	RUN	R.
FAC	FA.	SAVE	SA.
FIX	FI.	SCALE	SCA.
FOR	F.	SCREEN	SCRE.
GET@	GE.@	SCRNSET	SC.
GOSUB	GOS.	SCROLL	SCRO.
GOTO	G.	SGN	SG.
HCOPY	H.	SIN	SI.
HEX$	HE.	SPACE$	SPA.
INKEY$	INK.	SQR	SQ.
INPUT	I.	STRING$	STRI.
INSTR	INS.	SWAP	SW.
KEY	K.	SYNT	SY.
KEY LIST	K.L.	TAB	TA.
KEY OFF	K.OF.	THEN	TH.
KEY ON	K.O.	TIME$	TI.
LABEL	LA.	TRACE	T.
LEFT$	LEF.	UNTIL	U.
LINE	LIN.	VAL	VA.
LINE INPUT	LIN.I.	VARPTR	VAR.
LIST	L.	WEND	WE.
LOAD	LO.	WHILE	W.
LOCATE	LOC.	WOPEN	WO.
LPOS	LP.		
MEM$	MEM.		
MERGE	M.		
MID$	MI.		
NEXT	N		

17. KASSETTENHANDLING

SuperBASIC bietet dem Benutzer eine reichhaltige Befehlspalette für die Arbeit mit Kassettendateien. Verschiedene Aufzeichnungsformate und -optionen stehen zur Verfügung. Mit SuperBASIC ist es sogar möglich, eine Kassette nach Daten und Datengruppen gezielt abzusuchen, ohne die ganze Datei zu laden.

Die einzelnen Anweisungen für das Kassettenhandling lauten:

LOAD LOAD LOAD "[Programmname]"

Der LOAD-Befehl leitet das Laden eines SuperBASIC- bzw. eines Maschinenprogramms ein. Wenn ein Maschinenprogramm von der BASIC-Ebene aus geladen werden soll, muß der Speicherbereich, in dem es residieren wird, vorher mit der Anweisung CLEAR [Adresse] vom BASIC-Textbereich isoliert werden (siehe Seite 12).

LOAD ohne Argument bewirkt das Laden des nächsten auf der Kassette befindlichen Programms. Mit dem Zusatz "[Programmname]" zwingt man den Rechner, die Kassette nach dem spezifizierten Programm abzusuchen und, falls vorhanden, zu laden.

Der Ladevorgang kann mit [SHIFT] + [BREAK] abgebrochen werden.

MERGE MERGE MERGE "[Programmname]"

MERGE bewirkt das Nachladen eines Unterprogramms zu einem bereits geladenen SuperBASIC-Programm. Die nachzuladende Routine wird zeilenrichtig eingefügt, wobei residente Programmteile mit der selben Zeilennummerierung überschrieben werden. Die MERGE-Anweisung macht es möglich, Programme laufen zu lassen, die mehr als den verfügbaren Speicherplatz beanspruchen. Programmteile, die nur einmal benutzt werden (z.B. Initialisierungsroutinen, Programmbeschreibungen und ggf. DATA-Anweisungen), können mit Hilfe von MERGE während des Programmlaufs durch neue Programmteile ersetzt werden.

LOAD? LOAD? LOAD? "[Programmname"]

LOAD? entspricht dem VERIFY-Befehl in SHARP-BASIC. Es bewirkt einen Vergleich des gespeicherten Programms mit einem auf Kassette aufgezeichneten Programm.

Liegt eine 100-prozentige Übereinstimmung vor, meldet sich SuperBASIC mit "OK" zurück. Wenn nicht, so wird ein Bandlesefehler gemeldet.

SAVE SAVE SAVE "[Programmname]"

Mit dem SAVE-Befehl leitet man das Aufzeichnen eines SuperBASIC-Programms in Token-Format ein.

Der "[Programmname]" darf beliebige Zeichen enthalten. Er muß zwischen doppelten Anführungstrichen stehen und darf nicht länger als 16 Zeichen sein.

SAVE, A SAVE, A SAVE "[Programmname]", A

Diese Form des SAVE-Befehls bewirkt ein Programmaufzeichnung in ASCII-Format. Sie ist für die Aufzeichnung von Subroutinen zu verwenden, die mit der MERGE-Anweisung zu anderen Programmen nachgeladen werden sollen.

SAVEM SAVEM "[Programmname]",[Adresse 1],[Adresse 2]

Diese Variante gibt dem Benutzer die Möglichkeit, Maschinenprogramme von der SuperBASIC-Ebene aus aufzuzeichnen. Aufgezeichnet wird der Speicherbereich von [Adresse 1] (= Startadresse) bis [Adresse 2] (= Endadresse).

WOPEN WOPEN WOPEN "[Dateiname]"

WOPEN (= "write open") eröffnet eine Kassettendatei und ermöglicht die Datenaufzeichnung. Diese Anweisung verhält sich genauso wie in SHARP-BASIC.

ROPEN ROPEN ROPEN "[Dateiname]"

ROPEN (= "read open") öffnet eine vorhandene Kassettendatei und erlaubt das Programm, die darin enthaltenen Daten zu lesen. ROPEN verhält sich genauso wie in SHARP-BASIC.

PRINT#-1 PRINT#-1, [Datum], [Datum], ... , [Datum]

PRINT#-1 entspricht der PRINT/T-Anweisung in SHARP-BASIC und bewirkt die Eintragung (= Aufzeichnung) von Daten in eine mit WOPEN eröffnete Kassettendatei. Die [Daten] können mit einem Komma oder Semikolon voneinander getrennt werden. Das Aufzeichnungsformat entspricht genau dem Format, das die Daten bei der Ausgabe auf dem Drucker erhalten würden.

PRINT USING#-1 PRINT USING#-1,[Datum],....,[Datum]

Diese Anweisung erlaubt eine Aufzeichnung von Daten in sämtlichen zulässigen PRINT USING-Formaten. Für die die Formatierungsregeln siehe Abschnitt 8.2. "Die PRINT USING-Anweisung".

INPUT#-1 INPUT#-1, [Datum],...,[Datum]

Erlaubt das Lesen von Daten aus einer Kassettendatei, die mit ROPEN geöffnet wurde. Entspricht der INPUT/T-Anweisung in SHARP-BASIC.

Enthält ein zu lesendes Datum ein Komma, wird dies sowie das dahinterstehende Teildatum übersprungen.

LINE INPUT#-1 LINE INPUT#-1, [Datum],...,[Datum]

Wie INPUT#-1, aber mit dem Unterschied, daß Daten, die Kommata enthalten, vollständig gelesen werden. Vergl. LINE INPUT (Seite 25)

INKEY$(-n)

Liest sequentiell n Bytes aus einer Kassettendatei, beginnend an einer beliebigen Stelle. n darf Werte von -1 bis -32 annehmen.

Anwendung: Mit dieser Anweisung können innerhalb einer Banddatei gezielt Positionen gesucht werden. Die mit INKEY$(-n) gelesene Zeichen können dazu mit einem Suchbegriff verglichen werden.

Dateien können nach dieser Methode nach bestimmten Zeichen oder Zeichengruppen abgesucht werden, ohne die die gesamte Datei in den Arbeitsspeicher zu laden. Eine Kenntnis der in der Kassettendatei verwendeten Variablentypen ist nicht erforderlich.

Beispiel:
```
 10 A$="Test":B$="Bestehen":C$="Western":D$="@"
 20 WOPEN "Datei 1"
 30   PRINT#-1, A$, B$, C$, D$
 40 CLOSE "Datei 1"
 50   SB$ = "e": N = 0
 60   CMT 5
 70 ROPEN "Datei 1"
 80   IN$=INKEY$(-1)
 90    IF IN$="@"THEN 120
100    IF IN$=SB$THEN N=N+1
110   GOTO 80
120 PRINT "'e' kommt";N;"Mal in Datei 1 vor."
```

CMT CMT n

Diese Anweisung dient dazu, die verschiedenen Funktionen der Kassettenstation zu steuern. Das Argument n kann Werte von 0 bis 5 annehmen.

CMT 0 öffnet die Kassettenschacht
CMT 1 stoppt den Kassettenlauf
CMT 2 spult die Kassette vor
CMT 3 spult die Kassette zurück
CMT 4 spult die Kassette bis zum nächsten File Header vor
CMT 5 spult die Kassette bis zum nächsten File Header zurück

18. DIE GRAFIKANWEISUNGEN

Die SuperBASIC-Implementierung für den MZ-80B bietet eine Reihe von Zusatzbefehlen, die die Grafikfähigkeiten des Rechners um ein Vielfaches erweitern und die Erstellung von Computergrafiken wesentlich beschleunigen.

SCREEN SCREEN o,i,(xa, ya)-(xe, ye),(xm,ym),yb

Die SCREEN-Anweisung dient dazu, die verschiedenen Grafik- und Bildschirmbetriebsarten zu verwalten. Das Argument o, dem Werte von 0 bis 15 zugewiesen werden können, bestimmt die jeweilige Ausgabebetriebsart.

SCREEN 0 Schaltet den Grafikmodus aus.

SCREEN 1 Blendet den Grafikbereich 1 ein.

SCREEN 2 Blendet den Grafikbereich 2 ein.

SCREEN 3 Überlagert Grafikbereiche 1 und 2 (OR-Verknüpfung).

SCREEN 4
bis wie SCREEN 0 bis 3 aber im Inversmodus.
SCREEN 7

SCREEN 8
bis wie SCREEN 0 bis 3 aber ohne Bildschirmanzeige.
SCREEN 11

SCREEN 12
bis wie SCREEN 4 bis 7 aber ohne Bildschirmanzeige.
SCREEN 15

Die Argument 0 bis 15 können einer Zahlenvariablen zugewiesen werden.

Das Argument i bestimmt den Speicher, auf den die Grafikfunktionen zugreifen sollen. Werte von 0 bis 2 sind zulässig.

0 Grafikinformationen und ggf. Farbinformationen werden in beiden Grafikbereichen geschrieben.

1 Die entsprechenden Informationen werden nur im Grafikbereich 1 geschrieben.

2 Die entsprechenden Informationen werden nur im Grafikbereich 2 geschrieben.

Das Argument (xa,ya) definiert den oberen linken Eckpunkt des Bildschirmbereichs, auf den Grafikbefehle zugreifen können. Der untere rechte Eckpunkt wird mit dem Argument (xe,ye) definiert.

Die zulässigen Wertebereiche dieser Koordinaten sind:

für alle x-Koordinaten 0 <= xa < xe < 320

für alle y-Koordinaten 0 <= ya < ye < 200

Werden obige Werte bei der ersten Verwendung der SCREEN-Anweisung nicht spezifiziert, erhalten sie folgende Werte:

xa = 0, ya = 0, xe = 319, ye = 199 (= ganzer Bildschirm)

Sollen später komplexe grafische Funktionen ausgeführt werden, ist es sinnvoll, diese Werte entsprechend der jeweiligen Anwendung genau zu spezifizieren.

Das Koordinatenpaar (xm,ym) definiert den Koordinatenursprung für alle später verwendeten Grafikfunktionen.

Die zulässigen Werte lauten: 0 <= xm < 320
0 <= ym < 200

Werden xm bzw. ym nicht spezifiziert, erhalten sie den Wert 0.

Das Argument yb, das die Werte 0 oder 1 annehmen kann, bestimmt die Zählrichtung y-Achse. Für yb = 0 sind alle y-Werte oberder x-Achse positiv; für yb = 1 negativ.

Alle mit SCREEN definierten Werte bleiben bis zu einer Neudefinierung gespeichert. Wird eine neue SCREEN-Anweisung gegeben, die nicht alle Parameter enthält, werden für die nicht aufgeführten Parameter die alten Werte angenommen. Wenn einzelne Parameter bei der Neudefinierung weggelassen werden, muß das Anweisungsformat beibehalten werden. Dies wird am folgenden Beispiel veranschaulicht:

Grunddefinition: SCREEN 1,1,(0,0)-(319,199),(160,100),0

Soll nun der Koordinatenursprung geändert werden, genügt folgende Eingabe:

Neudefinition: SCREEN ,,,(0,0)

CLS CLS [Zahl]

CLS ohne Argument löscht den Textbildschirm. Mit einem Argument im Bereich 0-15 werden folgende Funktionen ausgelöscht:

CLS 0 Löscht beide Grafikspeicher mit Initialisierung
CLS 1 Löscht Grafik 1
CLS 2 Löscht Grafik 2
CLS 3 Wie 0 ohne Initialisierung
CLS 4-7 Wie 0-3 aber mit Löschen des Textbereiches
CLS 8-11 Wie 0-3 aber mit Vollschreiben der Speicher mit FFH
CLS 11-15 Wie 4-7 aber mit Vollschreiben der Speicher mit FFH

PSET PSET(x,y,f)

Mit PSET-Anweisung wird am Koordinatenpunkt x,y ein Pixel gesetzt. Das Argument f steht für "Farbwert" und kann, wenn keine Farbkarte angeschlossen ist, weggelassen werden. Ist eine Farbkarte vorhanden, können die Werte 0 bis 3 eingesetzt werden.

Die Wertebereiche, in der die PSET-Anweisung wirksam ist, lauten:

-30000 < x < +30000
-30000 < y < +30000

Der Koordinatenpunkt x,y bezieht sich auf den Koordinatenursprung. Für Koordinatenwerte, die über den Bildschirmbereich hinausgehen, ergibt sich der Wert aus einer Modulo-Division mit den jeweiligen Bildschirmendwerten.

PRESET PRESET(x,y,f)

PRESET unterscheidet sich von PSET nur dadurch, daß ein eventuell am Koordinatenpunkt x,y stehendes Pixel gelöscht wird.

PLOT PLOT(x,y),PSET
PLOT(x,y),PRESET
PLOT(x,y),XOR
PLOT(x,y),CHR$(n)
PLOT(x,y),"[String]"

Die PLOT-Anweisung verändert den am Koordinatenpunkt x,y herrschenden Zustand in Abhängigkeit vom angefügten Argument. Für die Grafikbereiche sind die Anweisungen PSET, PRESET und XOR gültig, für den Textbereich CHR$(n) und "[String]".

Die Argumente PSET und PRESET wirken wie die entsprechenden Anweisungen (siehe oben).

XOR invertiert den Zustand am Koordinatenpunkt.

CHR$(n) und "[String]" erzeugen die entsprechenden Werte oder Funktionen im Textbereich. Die Koordinaten sind in diesem Fall als Cursorkoordinaten zu verstehen.

SCROLL SCROLL n

Die SCROLL-Anweisung bewirkt eine Verschiebung des Grafik- oder Textspeicherinhalts in einer durch n bestimmten Richtung. Die Speicherbereiche sind hierbei als "kugelförmig" zu verstehen, d.h., aus dem Bildschirm geschoben Werte erscheinen auf der gegenüberliegenden Seite erneut.

SCROLL kann Werte zwischen 0 und 15 annehmen.

SCROLL 0 rollt den Textbildschirm nach oben.

SCROLL 1 rollt ihn nach rechts.

SCROLL 2 rollt ihn nach unten.

SCROLL 3 rollt ihn nach links.

SCROLL 4
bis wie 0-3 aber für Grafik 1
SCROLL 7

SCROLL 8
bis wie 0-3 aber für Grafik 2
SCROLL 11

SCROLL 12
bis wie 0-3 aber für Grafik 1 und 2
SCROLL 15

LINE

```
LINE (xa,ya)-(xe,ye), PSET, f
LINE (xa,ya)-(xe,ye), PRESET, f
LINE (xa,ya)-(xe,ye), XOR, f
LINE (xa,ya)-(xe,ye), CHR$(n), f
LINE (xa,ya)-(xe,ye), "[String]", f
```

Diese Anweisung zieht eine Gerade von Koordinatenpunkt xa,ya bis Koordinatenpunkt xe,ye. Diese Gerade kann mit den oben angeführten Funktionen gesetzt werden. Die Funktionen verhalten sich wie bei PLOT (siehe Seite 66). Das Argument f bezieht sich auf den Farbwert und kann ggf. weggelassen werden.

Die Koordinaten beziehen sich auf den vorher mit SCREEN definierten Koordinatenursprung.

Sollen von einer bereits gesetzten Geraden weitere Geraden ausgehen, müssen nicht mehr alle Argumente neu spezifiziert werden. Es genügt nun das Anweisungsformat:

LINE-(xe,ye) oder LINE (xa,ya)

Die bei der letzten LINE-Anweisung vorgegebenen Funktionsargumente bleiben erhalten, so daß fortlaufende Linien gezeichnet werden können.

Sollen fortlaufende Linienzüge gezeichnet werden, können alle Eckpunkte in einer LINE-Anweisung stehen. Format:

LINE (x1,y1)-(x2,y2)-...-(xn,yn)

```
LINE (xa,ya)-(xe,ye),[Funktionsargument], B
LINE (xa,ya)-(xe,ye),[Funktionsargument], BF
```

Das Anfügen des Arguments B (=Block) an eine LINE-Anweisung bewirkt die Erzeugung eines Rechtecks mit xa,ya als oberen linken und xe,ye als unteren rechten Eckpunkt.

Das Anfügen von BF hat dieselbe Wirkung, wobei der Inhalt des Rechtecks ausgemalt wird.

POINT POINT(x,y)

Mit dieser Funktion kann festgestellt werden, ob ein Pixel Koordinatenpunkt x,y gesetzt ist oder nicht.

POINT liefert folgende Werte:

0 ein Pixel ist weder in Grafik 1 noch 2 gesetzt
1 ein Pixel ist in Grafik 1 gesetzt
2 ein Pixel ist in Grafik 2 gesetzt
3 ein Pixel ist am gleichen Punkt in 1 und 2 gesetzt

POINT-Werte können Zahlenvariablen zugewiesen werden.

HCOPY HCOPY n

HCOPY 0 = Bildschirm-Hard-Copy vom Textbereich
HCOPY 1 = Bildschirm-Hard-Copy von Grafik 1
HCOPY 2 = Bildschirm-Hard-Copy von Grafik 2

CIRCLE

```
CIRCLE (x,y), r, a, w1, w2, PSET, f
CIRCLE (x,y), r, a, w1, w1, PRESET, f
CIRCLE (x,y), r, a, w1, w2, XOR, f
CIRCLE (x,y), r, a, w1, w2, CHR$(n), f
CIRCLE (x,y), r, a, w1, w2, "[String]", f
```

Mit der CIRCLE-Anweisung können beliebige rotationssymetrische Figuren gezeichnet werden.

(x,y) gibt den Mittelpunkt der Figur bezogen auf den Koordinatenursprung an.

r (Radius) legt den Abstand der Eckpunkte der Figur vom Mittelpunkt fest.

a bestimmt den Abstand der Eckpunkte voneinander in Grad.

w1 definiert den Anfangswinkel der Figur.

w2 definiert den Endwinkel der Figur.

f legt den Farbwert fest.

Die Grundwerte der CIRCLE-Anweisung sind:

CIRCLE (x,y), 40, 1, 0, 360, PSET

Bei einer Anweisung im Format CIRCLE (x,y) werden die Grundwerte 40, 1, 0, 360, PSET bei der Erstanwendung angenommen, andernfalls werden die zuletzt festgelegten Werte weiterverwendet.

PAINT PAINT(x,y),f

Die Anweisung PAINT dient zum Ausmalen abgegrenzter Flächen.

Die Koordinaten x,y legen den Anfangspunkt in der Fläche fest, an dem der Malvorgang beginnen soll. Von diesem Punkt aus werden horizontale Linien gezogen bis auf ein gesetztes Pixel getroffen wird. Dieser Vorgang wird für die gesamte Umgebung des Ausgangspunkt nach oben und unten fortgezetzt, bis die Fläche der umgebenden Figur ausgefüllt ist.

ACHTUNG: Ist in irgeneinder Richtung um (x,y) kein Begrenzungspunkt vorhanden, kommt es zu einem Überlauf.

PRINT@ PRINT@ (x,y), n, "[String]"
PRINT@ (x,y),-n, "[String]"

Entspricht der PATTERN-Anweisung in SHARP-BASIC.

(x,y) ist der Ausgangspunkt für das zu erstellende Bitmuster. (x,y) ist von der SCREEN-Anweisung unabhängig.

n bestimmt Höhe und Richtung des Musters bezogen auf (x,y). Bei positivem n ist die Richtung nach oben, bei negativem n nach unten.

"[String]" enthält die darzustellenden Bytes. Dabei stellt jedes ASCII-Zeichen eine der binären Wertigkeit entsprechendes Bitmuster dar. (Siehe auch SHARP-BASIC-Handbuch.

POKE@ POKE@ [Adresse], [Wert 1],...,[Wert n]

Ermöglicht trotz des nicht im Hauptspeicher befindlichen Bildschirmspeichers direkte Bildschirm-POKE-Anweisungen.

Die Adresseangabe muß in folgenden Bereichen liegen:

Textspeicher: &HD000 - &HD7CF
Grafik 1 & 2: &HE000 - &HFF3F

POKE@-Anweisungen an die Grafikspeicher wirken sich entsprechend der SCREEN-Anweisung aus.

PEEK@ PEEK@ [Adresse]

Ermöglicht "direkte" Bildchirm-PEEKs. Adressenbereiche: siehe POKE@

◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆

Anmerkung:

Da die hochauflösende Grafik für den SHARP MZ-80B nur als Option angeboten wird, haben wir in dieser Anleitung lediglich die wichtigsten Grafikfunktionen von SuperBASIC erläutert. Bei entsprechender Nachfrage, wollen wir eine spezielle Anleitung für Grafikfreunde mit vielen Beispiel anfertigen. Sollten Sie selbst interessante Applikationen mit SuperBASIC entwickeln, bitten unverbindlich um Mitteilung.

19. INDEX

Folgender Index enthält alle SuperBASIC-Befehle, Funktionen und Anweisungen in alphabetischer Reihenfolge:

Wort	Seite	Wort	Seite	Wort	Seite
ABS	34	ELSE	18	LOAD	13, 61
AND	41	END	17	LOAD?	61
ASC	42	ERL	57	LOCATE	14
ATN	34	ERR	57	LPOS	27
AUTO	12	ERROR	58	M	49
B	51	EXP	34	MEM$	26
BEEP	23	FAC	25	MERGE	13, 61
CALL	16	FIX	35	MID$	43
CDBL	34	FOR	17	MKD$	36
CHR$	42	FRE(0)	23	MKI$	37
CINT	34	G	50	MKS$	37
CIRCLE	68	GOSUB	17	MOD	38
CLEAR	12	GOTO	18	N	51
CLOSE	12	HCOPY	68	NEW	14
CLS	23	HEX$	42	NEXT	19
CLS 0-15	65	I	51	NOT	41
CMT	63	IF	51	O	50
CONT	12	INKEY$	23	OCT$	43
COS	34	INKEY$(-n)	63	OFF	26
CSGN	34	INP	25	OFF OFF	50
CSRLIN	23	INPUT	25	ON	19
CURSOR	23	INSTR	43	ON ERROR	57
CVD	37	INT	35	OR	41
CVI	37	KEY	9	OUT	25
CVS	37	KEY LIST	9	P	49
D	48	KEY OFF	47	PAINT	69
DATA	16	KEY ON	46	PAR	43
DEF	16	KEY 0	47	PEEK	25
DEF DBL	16	L	50	PEEK@	70
DEF INT	16	LABEL	18	PIX	36
DEF SNG	16	LEFT$	43	PLOT	66
DEF STR	11, 16	LEN	43	POINT	68
DEF DBL	16	LET	19	POKE	20, 26
DEF USR	16	LGN	35	POKE@	69
DELETE	12	LINE	67	POP	20
DIM	17	LINE INPUT	25	POS	27
DUMP	12	LINE INPUT#-1	36	PRESET	66
EDIT	13	LIST	13	PRINT	30

Wort	Seite	Wort	Seite
PRINT USING	31	SUM	36
PRINT USING#-1	62	SWAP	21
PRINT#1	30	SYNT	27
PRINT#-1	62	T	50
PRINT@	69	TAB	28
PSET	66	TAN	36
PUSH	20	THEN	18
R	50	TIME$	27
RAD	36	TRACE	15
READ	28	UNITL	21
REM	20	V	50
RENUM	14	VAL	44
REPEAT	20	Variablentyp	
REPEAT OFF	20	DOUBLE PREC.	11
REPEAT ON	20	Variablentyp	
RESTORE	21	INTEGER	10
RESUME	57	Variablentyp	
RETURN	21	SINGLE PREC.	10
RIGHT$	44	Variablentyp	
RND	35	STRING	10
ROPEN	62	VARPTR	29
RUN	14	W	50
S	50	WEND	22
SAVE	14, 62	WHILE	22
SCALE	15	WOPEN	62
SCREEN	64	XOR	41
SCRNSET	15	=	38, 39
SCRN$	45	-	38
SCROLL	66	Y	38
SGN	35	*	38
SIN	36	/	38
SPACE$	44	\	38
SPC	28	^	38
SQR	36	<	39
STOP	21	>	39
STRING$	44	<>, ><	40
STR$	44	<=, =<	40
SUB	48	>=, =>	40