NIXDORF
COMPUTER

Nixdorf 8870

Business BASIC

Nachdruck inkl. Ä 1 vom 01. 10. 86 · Bestell-Nr. 34692.00.2.93

Einleitung 1

Elemente der Syntax 2

Kurzliste der Anweisungen und BASIC-Kommandos 3

Die BASIC-Kommandos 4

Datenformate und Datendarstellung 5

Die BASIC-Anweisungen 6

Die BASIC-Funktionen 7

Die CALL-Anweisungen 8

Tastatur-Bildschirm-Ein-/Ausgabe 9

Die Magnetplatte 10

Dateibearbeitung 11

Die Drucker 12

Der Lochkartenleser 13

Lochstreifenleser / Lochstreifenstanzer 14

Das Magnetband 15

Die Magnetbandcassette 16

Programmstruktur 17

Testhilfen 18

TAMOS-Schnittstellen 19

Programmwartung 20

Programmierbeispiele 21

Fehlerbehandlung 22

Stichwortverzeichnis 23

Organisationsblatt

Organisationsblatt

Dieses Blatt gibt eine Übersicht über alle Änderungen, die seit der ersten Auflage an diesem Modul durchgeführt wurden. Es wird bei jeder Änderungsmitteilung mitgeliefert und ist jeweils auszutauschen.

Erstauflage:	01.06.85	Rel. 5.1
Änderung 1 :	01.10.86	Rel. 7.0

- Neue Kapitel:

  9.9
  8.19-1
  8.19-2
  8.28-1

- Diverse Ergänzungen und Korrekturen

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

* Änderungen gegenüber dem vorhergehenden Stand dieses Li-
* teraturmoduls sind auf den Seiten mit * am linken Rand
* gekennzeichnet.
*
* Das Ausgabedatum ist stets am Datum der jeweiligen Seite
* erkennbar.

Änderungswünsche/Fehler

Änderungswünsche/Fehler

Sollten Ihnen bei der Benutzung dieses Teils der Systemliteratur Fehler aufgefallen sein oder sollten Sie Vorschläge zur Verbesserung des Moduls haben, so bitten wir Sie, diese schriftlich zu formulieren und an folgende Anschrift zu schicken:

NIXDORF COMPUTER AG
Abt. ZSI
Fürstenallee 7

4790 Paderborn

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Inhaltsverzeichnis

Inhaltsverzeichnis

		Seite
1	Einleitung	1-1
1.1	BASIC und Business-BASIC	1-1
1.2	Hinweise zur Verwendung des Handbuchmoduls	1-3
1.3	Das BASIC-Programmiersystem	1-3
1.3.1	Neuanlegen eines BASIC-Programms	1-4
1.3.2	Ändern eines BASIC-Programms	1-4
1.3.3	Testen eines BASIC-Programms	1-4
1.3.4	Ablauf eines BASIC-Programms	1-4
1.3.5	Speichern von BASIC-Programmen auf der Platte	1-5
1.3.5.1	Die Wahl des geeigneten Programmtypes für Objektcode	1-5
1.3.5.2	Speichern und Laden von Quellprogrammen	1-5
2	Elemente der Syntax	2-1
2.1	Beschreibung der Metasprache	2-1
2.2	Liste der Syntaxelemente	2-4
3	Kurzliste der Anweisungen und BASIC-Kommandos	3-1
3.1	Kurzliste der Anweisungen	3-1
3.2	Kurzliste der BASIC-Kommandos	3-8
4	Die BASIC-Kommandos	4-1
4.1	Das Kommando ANALYSE	4-2
4.2	Das Kommando AUTOLN	4-4
4.3	Das Kommando COMMENT	4-4
4.4	Das Kommando CREF	4-6
4.5	Das Kommando DEBUG	4-6
4.6	Das Kommando DELETE	4-8
4.7	Das Kommando DUMP	4-9
4.8	Das Kommando EDIT	4-10
4.9	Das Kommando HASH	4-11
4.10	Das Kommando HELP	4-12
4.11	Das Kommando LIST	4-12
4.12	Das Kommando LOAD	4-13
4.13	Das Kommando NEW	4-14
4.14	Das Kommando PARTRENUM	4-14
4.15	Das Kommando PLIST	4-15
4.16	Das Kommando RENUMBER	4-19
4.17	Das Kommando RUN	4-20
4.18	Das Kommando SAVE	4-21
4.19	Das Kommando SIZE	4-22
4.20	Das Kommando TRACE	4-23
4.21	Das Kommando UNCOMMENT	4-25

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Inhaltsverzeichnis

5	Datenformate und Datendarstellung	5-1
5.1	Darstellung numerischer Daten	5-2
5.1.1	Einfache numerische Variablen	5-2
5.1.1.1	Aufbau einer BCD-Ganzzahl	5-2
5.1.1.2	Aufbau von Gleitkommazahlen	5-3
5.1.1.3	Arithmetischer Überlauf	5-4
5.1.2	Vektoren/Matrizen	5-5
5.1.3	Numerische Konstanten	5-6
5.2	Darstellung alphanumerischer Daten	5-7
5.2.1	String-Variablen	5-7
5.2.2	String-Literale	5-8
5.3	Deklaration (Dimensionierung) von Variablen	5-9
5.3.1	Aufbau der DIM- und der GLOBAL-Anweisung	5-10
5.3.2	Dimensionierung einfacher numerischer Variablen	5-11
5.3.3	Dimensionierung von Vektoren/Matrizen	5-13
5.3.4	Dimensionierung von String-Variablen	5-16
5.4	Adressierung von Variablen	5-17
5.4.1	Einfache numerische Variablen	5-17
5.4.2	Vektoren/Matrizen	5-17
5.4.3	String-Variablen	5-18
5.5	Hexadezimale Konstanten	5-21
6	Die BASIC-Anweisungen	6-1
6.1	Deklaration von Datenfeldern und Funktionen	6-2
6.1.1	Die DIM-Anweisung	6-5
6.1.2	Die DEALLO-Anweisung	6-5
6.1.3	Die GLOBAL-Anweisung	6-6
6.1.4	Die DEF-Anweisung	6-7
6.2	Datenmanipulation und Arithmetik	6-9
6.2.1	Die LET-Anweisung	6-9
6.2.1.1	Numerische Ausdrücke	6-14
6.2.1.2	Verknüpfung von Werten mit verschiedenen Genauigkeiten	6-15
6.2.1.3	Aufruf von Anwenderfunktionen	6-15
6.2.1.4	Aufruf von numerischen BASIC-Funktionen	6-16
6.2.1.5	Masken	6-17
6.2.1.6	String-Ausdrücke	6-21
6.2.1.7	Die CLEAR-Anweisung	6-21
6.2.2	Die DATA-Anweisung	6-22
6.2.3	Die READ-Anweisung	6-23
6.2.4	Die RESTOR-Anweisung	6-24
6.2.5	Die RANDOM-Anweisung	6-25
6.2.6	Die MAT-Anweisung	6-26
6.2.6.1	Zuweisung einer einfachen Matrix-Variablen	6-27
6.2.6.2	Matrizen-Addition und -Subtraktion	6-28
6.2.6.3	Matrizen-Multiplikation	6-29
6.2.6.4	Skalarmultiplikation mit einer Matrix	6-30
6.2.6.5	Die Funktion INV	6-31

Inhaltsverzeichnis

6.2.6.6	Die Funktion MIV	6-32
6.2.6.7	Die Funktion TRN	6-33
6.2.6.8	Die Funktion ZER	6-34
6.2.6.9	Die Funktion CON	6-35
6.2.6.10	Die Funktion IDN	6-36
6.2.7	Die Anweisung MAT READ	6-37
6.3	Steuerung des Programmablaufs	6-38
6.3.1	Die GOTO-Anweisung	6-38
6.3.2	Die GOSUB-Anweisung	6-39
6.3.3	Die RETURN-Anweisung	6-40
6.3.4	Die ON-Anweisung	6-41
6.3.5	Die IF-Anweisung	6-42
6.3.6	Die IF ERR 0-Anweisung	6-44
6.3.7	Die IF ESC-Anweisung	6-45
6.3.8	Die FOR-Anweisung	6-46
6.3.9	Die NEXT-Anweisung	6-47
6.3.10	Die CHAIN-Anweisung	6-48
6.3.11	Die LINK-Anweisung	6-49
6.3.12	Die STOP-Anweisung	6-51
6.3.13	Die END-Anweisung	6-51
6.4	Die Ein-/Ausgabeanweisungen	6-52
6.4.1	Die Anweisungen zur Tastatur-Bildschirm-Ein-/Ausgabe	6-53
6.4.1.1	Die INPUT-Anweisung	6-54
6.4.1.2	Die Anweisung MAT INPUT	6-56
6.4.1.3	Die PRINT-Anweisung	6-57
6.4.1.4	Die Anweisung MAT PRINT	6-60
6.4.2	Die Anweisungen des logischen IOCS	6-61
6.4.2.1	Die OPEN #-Anweisung	6-61
6.4.2.2	Die ENV #-Anweisung	6-69
6.4.2.3	Die READ #-Anweisung	6-71
6.4.2.4	Die WRITE #-Anweisung	6-73
6.4.2.5	Die PRINT #-Anweisung	6-76
6.4.2.6	Die MAT READ #-Anweisung	6-81
6.4.2.7	Die MAT WRITE #-Anweisung	6-84
6.4.2.8	Die SEARCH #-Anweisung	6-87
6.4.2.9	Die CLOSE #-Anweisung	6-93
6.4.2.10	Die BUILD #-Anweisung	6-94
6.4.2.11	Die KILL-Anweisung	6-97
6.5	Spezielle Anweisungen	6-98
6.5.1	Kommentare in BASIC-Programmen	6-98
6.5.1.1	Die REM-Anweisung	6-98
6.5.1.2	Kommentare in beliebigen Anweisungen	6-99
6.5.2	Inter-Task Kommunikation	6-100
6.5.2.1	Die SIGNAL 1-Anweisung	6-101
6.5.2.2	Die SIGNAL 2-Anweisung	6-102

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Inhaltsverzeichnis

6.5.3	Die SIGNAL 3-Anweisung	6-103
6.5.4	Die CALL-Anweisungen	6-104
6.5.5	Die LGERR-Anweisung	6-105
7	Die BASIC-Funktionen	7-1
7.1	Mathematische Funktionen	7-2
7.1.1	Trigonometrische Funktionen	7-2
7.1.1.1	Die Funktionen SIN, COS und TAN	7-2
7.1.1.2	Die Funktion ATN	7-3
7.1.2	Transzendente Funktionen	7-3
7.1.2.1	Die Funktion SQR	7-3
7.1.2.2	Die Funktion EXP	7-4
7.1.2.3	Die Funktion LOG	7-4
7.1.3	Arithmetische Funktionen	7-4
7.1.3.1	Die Funktion ABS	7-5
7.1.3.2	Die Funktion SGN	7-5
7.1.3.3	Die Funktion INT	7-5
7.1.3.4	Die Funktion NOT	7-6
7.1.3.5	Die Funktion FRA	7-6
7.1.3.6	Die Funktion RND	7-7
7.2	Funktionen zur Zahlenmanipulation	7-8
7.2.1	Die Funktion MAN	7-9
7.2.2	Die Funktion CHR	7-9
7.2.3	Die Funktion IXR	7-10
7.3	Sonderfunktionen	7-11
7.3.1	Die Funktion SPC	7-11
7.3.2	Die Funktion LEN	7-13
7.3.3	Die Funktion CHF	7-13
7.3.4	Die Funktion DET	7-14
7.3.5	Die Funktion KEY	7-15
7.3.6	Die Funktion LKY	7-16
7.3.7	Die Funktion CKY	7-17
7.3.8	Die Funktion CHN	7-17
7.3.9	Die Funktion FLN	7-18
7.3.10	Die Funktion PRN	7-18
8	Die CALL-Anweisungen	8-1
8.1	Übersicht über die verfügbaren Unterprogramme	8-2
8.2	CALL 1	8-5
8.3	CALL 2	8-9
8.4	CALL 3	8-10
8.5	CALL 4	8-11
8.5.1	Ausgabe von Nachrichten an anderen Arbeitsplätzen	8-11
8.5.2	Schreiben/Lesen der Archiv- und der LU-Nummer	8-12
8.6	CALL 6	8-15
8.7	CALL 7	8-17
8.8	CALL 20	8-17
8.9	CALL 21	8-20

Inhaltsverzeichnis

8.10	CALL 22	8-22
8.11	CALL 23	8-24
8.12	CALL 24	8-27
8.13	CALL 25	8-28
8.14	CALL 26	8-30
8.15	CALL 28	8-31
8.16	CALL 29	8-33
8.17	CALL 31	8-36
8.18	CALL 32	8-37
8.19	CALL 35	8-41
*8.19-1	CALL 43	8-44-1
*8.19-2	CALL 44	8-44-3
8.20	CALL 51	8-45
8.21	CALL 59	8-47
8.22	CALL 60	8-49
8.23	CALL 61	8-51
8.24	CALL 62	8-53
8.25	CALL 63	8-56
8.26	CALL 64	8-57
8.27	CALL 65	8-59
8.28	CALL 66	8-61
*8.28-1	CALL 68	8-67-1
8.29	CALL 70	8-67
8.30	CALL 72	8-71
8.31	CALL 73	8-72
8.32	CALL 74	8-73
8.33	CALL 80	8-75
8.34	CALL 84	8-79
8.35	CALL 90	8-82
8.36	CALL 91	8-86
8.37	CALL 93	8-88
8.38	CALL 96	8-90
8.39	CALL 97	8-93
8.40	CALL 98	8-96
8.41	CALL 99	8-98
9	Tastatur-Bildschirm-Ein-/Ausgabe	9-1
9.1	Die INPUT-Anweisung	9-2
9.2	Die MAT INPUT-Anweisung	9-3
9.3	Die PRINT-Anweisung	9-3
9.4	Die MAT PRINT-Anweisung	9-3
9.5	Die Bildschirmfunktionen	9-4
9.5.1	Die Funktion TAB	9-4
9.5.2	Die Funktion Backspace	9-5
9.5.3	Die Funktion Mark Cursorposition	9-6
9.5.4	Die Funktion Back to marked Cursorposition	9-6
9.5.5	Die Funktion Carriage Return	9-7
9.5.6	Die Funktion Line Deletion	9-7
9.5.7	Die Funktion Line Insertion	9-8
9.5.8	Die Funktion Start Background	9-8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Inhaltsverzeichnis

9.5.9	Die Funktion Start Foreground	9-9
9.5.10	Die Funktion Clear Screen	9-9
9.5.11	Die Funktion Clear Foreground	9-10
9.5.12	Die Funktion Ring Keyboard Bell	9-10
9.5.13	Die Funktion Tab to next Foregroundfield	9-11
9.5.14	Die Funktion Clear Foregroundfield	9-11
9.5.15	Die Funktion Cursor Home	9-12
9.5.16	Die Funktion Dark	9-12
9.5.17	Die Funktion Hexadezimal-Ausgabe	9-13
9.5.18	Die Funktion Duplicate Field	9-13
9.5.19	Die Funktion Duplicate Characters Limited	9-14
9.5.20	Die Funktion Duplicate Characters	9-15
9.5.21	Die Funktion Display Memory	9-16
9.5.22	Die Funktion Cursor Position	9-17
9.5.23	Die Funktion Define Window	9-17
9.5.24	Die Funktion No Typewriter Keyboard	9-18
9.5.25	Die Funktion Typewriter Keyboard	9-18
9.5.26	Die Funktion 8-Bit-Übertragung einschalten	9-19
9.5.27	Die Funktion 8-Bit-Übertragung ausschalten	9-19
9.5.28	Übersicht über die verfügbaren Bildschirm- und Gerätefunktionen	9-20
9.6	Auslöse- und Rasttasten-Programmierung	9-23
9.6.1	Die Funktion KEY	9-23
9.6.2	Die Funktion LKY	9-24
9.6.3	Die Funktion CKY	9-25
9.7	Eingabe über OCR-A/B-Handleser, BCR, SCR, IDKG	9-26
9.7.1	Ablauf einer Eingabe	9-26
9.7.2	Besonderheiten der einzelnen Geräte	9-28
9.7.2.1	Besonderheiten des OCR-A/B-Lesers	9-28
9.7.2.2	Besonderheiten des BCR	9-30
9.7.2.3	Besonderheiten des SCR	9-31
9.7.2.4	Besonderheiten des IDKG	9-31
9.8	ASCII-Code-Tabelle	9-34
*9.9	I/O-Makro	9-99
*9.9.1	Aufruf des I/O-Makros	9-101
*9.9.2	Prinzip der Feldbehandlung beim I/O-Makro	9-105
*9.9.2.1	Behandlung von Vorbelegungen und Eingaben	9-105
*9.9.2.2	Behandlung von Auslösetasten	9-108
*9.9.3	Feldbeschreibungsparameter	9-113
*9.9.3.1	Spaltennummer	9-114
*9.9.3.2	Zeilennummer	9-114
*9.9.3.3	Skip-Status	9-114
*9.9.3.4	Ein-/Ausgabeart	9-120
*9.9.3.5	Maximale Eingabelänge (Anzahl Ziffern bzw. Bytes)	9-121
*9.9.3.6	Minimale Eingabelänge (Anzahl Ziffern bzw. Bytes)	9-122
*9.9.3.7	Anzahl Nachkommastellen	9-122
*9.9.3.8	Art des Vorzeichens	9-122
*9.9.3.8	Maske für Ausgabe	9-123

Inhaltsverzeichnis

*9.9.3.10	Hardkey-Zulässigkeiten	9-124
*9.9.3.11	Softkey-Zulässigkeiten	9-124
*9.9.3.12	HELP-Text-Definition	9-124
*9.9.3.13	Minimaler Eingabewert	9-125
*9.9.3.14	Maximaler Eingabewert/Erlaubte Zeichen	9-125
*9.9.3.15	USING-Maske	9-126
*9.9.3.16	Vorbelegung	9-126
*9.9.3.17	Attribute	9-127
*9.9.3.18	Automatische "?"-Funktion	9-129
*9.9.4	Feldbeschreibungs-(FB)-Generator	9-130
*9.9.4.1	Anlegen einer Feldbeschreibung (CRE)	9-133
*9.9.4.2	Ändern einer Feldbeschreibung (AME)	9-137
*9.9.4.3	Löschen einer Feldbeschreibung (DEL)	9-137
*9.9.4.4	Anzeigen einer Feldbeschreibung (REV)	9-137
*9.9.4.5	Listen von Feldbeschreibungen in Kurzform (LST).	9-138
*9.9.4.6	Drucken fon Feldbeschreibungen in Kurzform (PRT)	9-138
*9.9.4.7	Reorganisieren der Datei (REO)	9-139
*9.9.4.8	Vergleichen zweier Feldbeschreibungsdateien (VGL)	9-139
*9.9.5	CALL 9	9-141
*9.9.6	Beispiel für I/O-Makro-Aufruf	9-143
10	Die Magnetplatte	10-1
10.1	Das Ein-/Ausgabekonzept	10-1
10.1.1	Dateiorganisation	10-1
10.1.2	Record-Pointer	10-2
10.1.3	Das Kanalkonzept	10-2
10.1.4	Dateidefinitionen	10-2
10.1.5	Der Dateiname	10-3
10.1.6	Das Konzept der logischen Einheiten	10-4
10.2	Aufbau von Magnetplattendateien	10-5
10.3	File-Sharing	10-6
10.4	Erstellung von Magnetplattendateien	10-9
10.5	Der Zugriff auf Magnetplattendateien	10-10
10.5.1	Die Dateieröffnung	10-10
10.5.2	Der Dateizugriff	10-11
10.5.3	Das Schließen von Dateien	10-11
10.6	Das Löschen von Dateien	10-12
10.7	Die Funktion CHF	10-12
11	Dateibearbeitung	11-1
11.1	Dateiorganisationsarten	11-1
11.2	Formatierte Dateien	11-2
11.2.1	Erstellen formatierter Dateien	11-3
11.2.2	Zugriff auf formatierte Dateien	11-4
11.3	Relative Dateien	11-8
11.3.1	Erstellen relativer Dateien	11-8
11.3.2	Zugriff auf relative Dateien	11-9
11.3.3	Der EOF-Zeiger (End Of File)	11-15

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Inhaltsverzeichnis

11.4	Indexdateien	11-16
11.4.1	Erstellen von Indexdateien	11-17
11.4.2	Zugriff auf Indexdateien	11-19
11.4.3	Informationen über die Indexdatei	11-20
11.4.4	Datenbereich sequentiell lesen	11-21
11.4.5	Datenbereich in Verzeichnisordnung lesen	11-21
11.4.6	Wahlfreier Zugriff über einen Ordnungsbegriff	11-22
11.4.7	Datensatz mit Ordnungsbegriffen einfügen	11-22
11.4.8	Datensatz mit Ordnungsbegriffen löschen	11-23
11.4.9	Indexdatei löschen	11-23
11.4.10	Statusmeldungen	11-24
11.4.11	Das Schließen der Datei	11-24
11.4.12	Probleme bei der Bearbeitung von Indexdateien	11-25
11.5	Textdateien	11-28
11.5.1	Erstellen von Textdateien	11-29
11.5.2	Zugriff auf Textdateien	11-30
11.6	Sequentielle Dateien	11-35
12	Die Drucker	12-1
12.1	Zugriff auf Druckerdateien	12-2
12.1.1	Eröffnen von Druckerdateien	12-3
12.1.2	Ausgabe auf dem Drucker	12-8
12.1.2.1	Formularsteuerung	12-9
12.1.2.2	Einzelformularverarbeitung	12-9
12.1.2.3	Papiervorschub	12-11
12.1.2.4	Papiercassettenauswahl	12-12
12.1.2.5	Verbreiterung des Schriftbilds	12-12
12.1.2.6	Zeichendichte	12-13
12.1.3	Die Funktion CHF	12-16
12.1.4	Das Schließen des Druckers	12-16
13	Der Lochkartenleser	13-1
13.1	Zugriff auf Lochkartendateien	13-1
13.1.1	Eröffnen von Lochkartenleserdateien	13-1
13.1.2	Eingaben über Lochkartenleser	13-2
13.1.3	Schließen des Lochkartenlesers	13-2
13.2	Beispiel	13-2
14	Lochstreifenleser/Lochstreifenstanzer	14-1
14.1	Eingabe über den Lochstreifenleser	14-1
14.2	Ausgabe über den Lochstreifenstanzer	14-3
14.3	Code-Tabellen für Lochstreifenleser/-Stanzer	14-5
15	Das Magnetband	15-1
15.1	Organisation der Daten auf einem Magnetband	15-1
15.2	Zugriff auf Magnetbanddateien	15-6
15.2.1	Eröffnen des Magnetbandgerätes mit der OPEN-Anweisung	15-6

Inhaltsverzeichnis

15.2.2	Die READ-Anweisung	15-7
15.2.3	Die WRITE-Anweisung	15-7
15.2.4	Die ENV-Anweisung	15-8
15.2.4.1	Übergabe der Codetabellen	15-8
15.2.4.2	Die Status-Abfrage	15-9
15.2.4.3	Rückspulen	15-10
15.2.4.4	Rückspulen und Abschalten	15-10
15.2.4.5	Bandmarke schreiben	15-10
15.2.4.6	Blöcke vorsetzen	15-11
15.2.4.7	Blöcke zurücksetzen	15-11
15.2.4.8	Bandmarken vorsetzen	15-12
15.2.4.9	Bandmarken zurücksetzen	15-12
15.2.5	Die CLOSE-Anweisung	15-13
15.2.6	Eröffnen des Magnetbandgeräts mit dem CALL 70	15-13
15.2.7	Verarbeitung	15-14
15.2.7.1	Verwaltung des Datenträgers	15-14
15.2.7.2	Band vor- oder zurücksetzen	15-15
15.2.7.3	Block lesen/schreiben	15-16
15.2.8	Schließen des Bandgerätes	15-17
15.2.9	Ermitteln der Größe des Magnetbandpuffers	15-17
15.3	Fehlerbehandlung	15-18
15.3.1	Fehlerbehandlung der Magnetband-Schnittstelle	15-18
15.3.2	Fehlerstatuscodes bei Anwendung des CALL 70	15-19
16	Die Magnetbandcassette	16-1
16.1	Organisation der Daten auf der Cassette	16-1
16.2	Zugriff auf Cassettendateien	16-3
16.2.1	Eröffnen des Magnetbandcassettengeräts	16-3
16.2.2	Verarbeitung	16-4
16.2.2.1	Verwaltung des Datenträgers	16-4
16.2.2.2	Cassettenband vor- oder zurücksetzen	16-4
16.2.2.3	Block lesen/schreiben	16-5
16.2.3	Schließen des Magnetbandcassettengeräts	16-5
16.3	Fehlerbehandlung	16-6
17	Programmstruktur	17-1
17.1	Die Ausführung eines Programms	17-1
17.2	Die FOR-NEXT-Schleife	17-3
17.3	Die Unterprogrammtechnik	17-4
17.4	Unterprogramme in Maschinencode	17-6
17.5	Segmentierung/Verkettung von Programmen	17-6
17.6	Hinweise für laufzeitoptimale Programmierung	17-7
17.6.1	BASIC-Optimizer	17-7
17.6.2	Programm-Gliederung	17-7
17.6.3	Umstellen alter Programme	17-9
17.6.4	Weitere wichtige Hinweise	17-12
18	Testhilfen	18-1
18.1	Die BASIC-Fehleranzeige	18-1

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Inhaltsverzeichnis

18.2	Die IF ERR 0-Anweisung	18-2
18.3	Die IF ESC - Anweisung	18-4
18.4	Die Direktausführung von Anweisungen	18-5
18.5	Der Einzelschritt-Testlauf und das Ausführungsprotokoll	18-7
18.6	Testlauf des Programms	18-9
18.7	Die STOP-Anweisung	18-11
18.8	Die "ESC"-Taste	18-12
18.9	Der Variablen-Dump	18-13
18.10	Der Variablen-Stop	18-14
19	TAMOS - Schnittstellen	19-1
19.1	Vorbereitung von Programmen zur Einbindung in Selektoren	19-1
19.2	Starten eines Anwenderprogrammes	19-3
19.3	Kopfzeile und Nachrichtenzeile anzeigen	19-5
19.4	Beenden eines Anwenderprogramms	19-6
19.5	Logbuch und Nachrichtendatei	19-8
19.6	Common-Area	19-10
19.6.1	Übergabe von Standardparametern durch TAMOS	19-10
19.7	Fehlerbehandlung	19-12
19.8	Job-Spooler	19-13
19.8.1	Job in die Jobspooldatei eintragen	19-13
19.9	Parameter für den Druckspooler	19-19
19.9.1	Formularsteuerung	19-19
19.9.2	Spoolersteuerung	19-20
19.10	Start von Programmen auf einem Phantomport	19-21
19.11	Meldungen aus einem Phantomport	19-22
20	Programmwartung	20-1
20.1	Programmliste	20-2
20.2	Kommentare Auslagern	20-4
20.3	Kommentare Einlagern	20-6
20.4	Programmvergleich	20-8
20.5	Prüfsummenliste	20-11
20.6	Sichern unter anderem Filetyp (Programmoptimierung)	20-13
20.7	Programm-Analyse	20-15
21	Programmierbeispiele	21-1
21.1	Programmierbeispiele Indexdateibearbeitung	21-1
21.1.1	Direktes Lesen eines Datensatzes	21-2
21.1.2	Einfügen eines Ordnungsbegriffs mit zugehörigem Datensatz	21-2
21.1.3	Ordnungsbegriff löschen und zugehörigen Datensatz freigeben	21-3
21.1.4	Sequentielle Bearbeitung ab dem nächstgrößeren Begriff	21-3

Inhaltsverzeichnis

21.2	Beispiel einer Eingabe-Routine	21-4
21.3	Relative Datei auf Magnetband kopieren	21-7
21.4	Textdatei drucken	21-9
21.5	Druckerdatei in Textdatei zwischenspeichern ...	21-11
21.6	Beipiel zur CHF-Funktion	21-12
21.7	Lochkartendatei in relative Datei einlesen	21-12
21.8	CALL 97-Anwendung	21-13
21.9	Standard-Basisteil für Anwenderprogramme im TAMOS-System	21-15
21.9.1	Einführung und Handhabung	21-15
21.9.2	Beschreibung der einzelnen Routinen	21-17
21.9.3	Der Basisteil	21-22
22	Fehlerbehandlung	22-1
22.1	BASIC-Fehler	22-1
22.2	Fehlermeldungen des Arbeitsplatzes	22-8
23	Stichwortverzeichnis	23-1

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Einleitung

1 Einleitung

1.1 BASIC und Business-BASIC

Business-BASIC geht auf die Programmiersprache BASIC (Beginner's All-purpose Symbolic Instruction Code) zurück, die Mitte der sechziger Jahre am Dartmouth College in den USA entwickelt wurde. Die sinngemäße Übersetzung des Namens in "Symbolischer All-Zweck-Befehlscode für Anfänger" weist schon auf zwei wesentliche Eigenschaften dieser Programmiersprache hin:

1. BASIC ist einfach zu erlernen, da die Sprache symbolisch ist und einen unkomplizierten Aufbau hat.

2. BASIC ist für jeden Zweck geeignet, d.h. es handelt sich um eine leistungsstarke Programmiersprache mit breitem Funktionsspektrum.

Zur Darstellung der dritten wesentlichen Eigenschaft von BASIC muß folgendes vorangestellt werden:
Soll von einer elektronischen Datenverarbeitungsanlage ein Problem gelöst werden, müssen ihr die Lösungsschritte durch eine Folge von Instruktionen, dem Programm, eingegeben werden. Diese Instruktionen, ihr Aufbau und ihre Anwendung werden durch die Grammatik einer Programmiersprache vorgeschrieben. Man unterscheidet zwei Arten von Programmiersprachen:

"Maschinensprachen" und "symbolische Sprachen".

Maschinensprachen sind umständlich anzuwenden und außerdem von Computertyp zu Computertyp verschieden, d.h. nicht direkt übertragbar.

Aus diesem Grund werden die meisten Programme in einer (maschinenunabhängigen) symbolischen Sprache geschrieben, die erst später in Maschinensprache übersetzt wird. Aus jeder Instruktion einer Symbolsprache wird dabei meist eine Vielzahl von Maschinenbefehlen gebildet, womit ein weiterer Grund für das einfachere Programmieren in einer Symbolsprache gegeben ist.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Einleitung

Bei den meisten anderen höheren Programmiersprachen (PL/1, ALGOL, COBOL usw.) wird mit einem Editor ein Quellprogramm erfaßt und mit einem Compiler Objektcode erzeugt. Der Objektcode ist meistens Maschinencode und ist direkt vom Rechner ausführbar. Aus diesem Maschinencode kann kein Quellprogramm mehr hegestellt werden. Der Anwender muß also beide Programmformen Quell- und Objektprogramm verwalten.

Bei BASIC sind die Schritte, die zu einem ablauffähigen Programm führen, implizit in verschiedenen Kommandos enthalten. Der Editor enthält die lexikalische Analyse, den Syntaxcheck und einen Teil der Objektcodegenerierung, der üblicherweise in herkömmlichen Compilern liegt. Der BASIC-Objektcode ist kein Maschinencode und muß von einem Programm-Interpreter ausgeführt werden. Aus diesem Objektcode kann mit Kommandos das Quellprogramm wieder erzeugt werden, so daß der Anwender nur eine Programmform benötigt.

Die wesentlichen Vorteile eines Interpreter-Einsatzes sind:

- Interaktive Programmerstellung.
  Ein Programm oder auch Teile eines Programmes können unmittelbar, nachdem sie eingegeben wurden, ausgeführt, modifiziert, wieder ausgeführt, gelöscht oder erweitert werden. Jede Anweisung wird schon bei der Eingabe auf lexikalische und syntaktische Fehler untersucht.
- Speicherplatzersparnis und weniger Dateihandling.
  Es wird lediglich der Zwischencode gespeichert, der direkt interpretiert werden kann.

Die Sprache BASIC war ursprünglich für den wissenschaftlichen Gebrauch bestimmt, d.h. der Lösung mathematischer Probleme wurde Vorrang gegeben gegenüber der Ein-/Ausgabe und Manipulation großer Datenmengen. Bei der Weiterentwicklung von BASIC zum kommerzielleren Business-BASIC wurde die Vielzahl der Rechenmöglichkeiten sowie die einfache Sprachgrammatik beibehalten.
Die Anwendbarkeit auch auf kaufmännische Bereiche wurde vor allem durch die Unterstützung vieler Peripherie-Geräte erreicht: in erster Linie die Magnetplatte, aber auch Magnetband, Magnetbandcassette, Lochkartenleser, Diskette, Nadel- und Schnelldrucker sowie Datenfernübertragung, Lochstreifenleser, Lochstreifenstanzer und Plotter.

Einleitung

1.2 Hinweise zur Verwendung des Handbuchmoduls

Das vorliegende Handbuch ist für die folgenden drei Verwendungsmöglichkeiten vorgesehen:

1. Für den Teilnehmer eines BASIC-Einführungskurses als Begleitmaterial.

2. Für den erfahrenen Programmierer, der BASIC im Selbststudium erlernen möchte.

3. Für den erfahrenen BASIC-Programmierer als Nachschlagewerk. Hier sei besonders auf das Stichwortverzeichnis des Handbuchs verwiesen.

1.3 Das BASIC-Programmiersystem

Das Bearbeiten eines BASIC-Programms geschieht mit dem BASIC-Prozessor aus dem BASIC-Kommandomodus, den man mit TAMOS über den EXPERT-Selektor erreicht. Der BASIC-Prozessor ist ein Systemprogramm, das das Programmieren im Dialog ermöglicht und darüber hinaus mit verschiedenen Kommandos unterstützt. Der BASIC-Kommandomodus wird grundsätzlich dadurch angezeigt, daß sich der Cursor (Lichtzeiger am Bildschirm) am Anfang einer Bildschirmzeile befindet. Es wird vom Programmierer die Eingabe eines Kommandos oder einer Anweisung erwartet (Dialogbetrieb). Im folgenden soll unter Eingabe stets das Eingeben von Zeichen mit der Tastatur einschließlich des abschließenden Auslösens mit der "CR"-Taste verstanden werden.

Der BASIC-Prozessor kann durch gleichzeitiges Drücken der Tasten "CTL" und "C" wieder verlassen werden. Falls das Programm nach Erfassung oder erfolgten Änderungen nicht gesichert wurde (mit Hilfe des SAVE-Kommandos), wird die Warnung "SAVE?!" ausgegeben. Erst ein erneutes Drücken von "CTL" und "C" veranlaßt, daß der Prozessor verlassen wird.

Mit Hilfe des Syntaxanalysators, der Bestandteil des BASIC-Prozessors ist, werden BASIC-Anweisungen schon beim Editieren auf korrekte Syntax überprüft und erkannte Fehler auf dem Bildschirm angezeigt. Ein Statement wird bis zu einer Länge von 251 Zeichen angezeigt, wobei der Cursor auf der Position steht, an der der Fehler erkannt wurde.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Einleitung

1.3.1 Neuanlegen eines BASIC-Programms

Mit dem Kommando NEW wird das Programm im Speicher gelöscht, und es kann ein neues Programm eingegeben werden. Zum Schluß nicht vergessen, das Programm mit dem SAVE-Kommando zu sichern, wenn das neu eingegebene Programm später wieder genutzt werden soll!
Damit wird es in eine Programmbibliothek auf Magnetplatte ausgelagert.

1.3.2 Ändern eines BASIC-Programms

Wenn in der Testphase eines Programms Fehler festgestellt wurden oder wenn das Programm gewartet werden soll, muß das bereits existierende Programm geändert werden. Zunächst wird es durch Anwahl über den EXPERT-Selektor in den Hauptspeicher geladen. Dabei ist die Angabe des Programmnamens und der LU# erforderlich.

Danach können Anweisungen gelöscht, überschrieben, hinzugefügt oder mit dem EDIT-Kommando gelistet und geändert werden.

Zum Schluß nicht vergessen, die neue Fassung des Programms mit dem SAVE-Kommando zu sichern!

1.3.3 Testen eines BASIC-Programms

Bevor die Bearbeitung eines BASIC-Programms endgültig abgeschlossen wird, wird in der Regel überprüft, ob es die gestellten Anforderungen erfüllt.

Dazu und zur gezielten Fehlererkennung stehen verschiedene Testhilfsmittel zur Verfügung, die im Kapitel "Testhilfen" beschrieben werden.

1.3.4 Ablauf eines BASIC-Programms

Den Ablauf eines BASIC-Programms erreicht man durch Anwahl aus dem EXPERT-Selektor mit Angabe des Programmnamens oder aus dem Kommandomodus mit dem Kommando RUN, wenn sich das gewünschte Programm bereits im Hauptspeicher befindet.

1.3.5 Speichern von BASIC-Programmen auf der Platte

1.3.5.1 Die Wahl des geeigneten Programmtypes für Objektcode

Wie schon beschrieben, reicht es in BASIC aus, nur den Objektcode für die Programmerfassung und Ausführung zu haben. Dieser Objektcode ist ein sogenanntes B-File (BASIC-Programm mit Filetyp B) und entsteht direkt nach dem Editieren und Speichern mit dem SAVE-Kommando. Durch die Einführung eines Optimizers (weitergehende Compilierung) kann ein besserer Objektcode erzeugt werden. Der Optimizer kann mit dem SAVE-Kommando aufgerufen werden, so daß ein D-File (BASIC-Programm mit Filetyp D) entsteht.

Achtung:

Da D-Files viel schneller ausgeführt werden, sollten nur noch diese verwendet werden.

Programme können andere Programme aufrufen und Variablen an diese übergeben. Solche Programme können unterschiedliche Programmtypen haben. Zu beachten ist jedoch, wenn eine möglichst tiefe Compilation erreicht werden soll (möglichst gute Laufzeiten), müssen die Programme D-Files sein.

1.3.5.2 Speichern und Laden von Quellprogrammen

Mit dem DUMP-Kommando ist es möglich, aus Objektprogrammen wieder Quellprogramme (Textfiles) und umgekehrt mit dem LOAD-Kommando aus dem Quellprogramm wieder Objektprogramme zu erzeugen.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

2 Elemente der Syntax

In diesem Kapitel werden Elemente der Sprache Business-BASIC eingeführt, die zur Bildung gültiger Anweisungen benötigt werden.
Die Angabe der Regeln zum Bilden dieser Elemente geschieht mit Hilfe einer Metasprache.

2.1 Beschreibung der Metasprache

Zur Beschreibung des Sprachumfangs wird im folgenden eine Metasprache verwendet, die angelehnt ist an die Backus-Naur-Form (BNF).

Ihre Zeichen und deren Bedeutung sind:

< > (spitze Klammer auf/zu)

Innerhalb dieser Klammersymbole steht in abgekürzter Form der Name einer Klasse von Sprachelementen, z.B. "NAusdr" als Abkürzung für "numerischer Ausdruck".

Falls es notwendig ist, in der Syntaxbeschreibung gleiche Klassen von Sprachelementen (z.B. numerische Ausdrücke) voneinander zu unterscheiden, so geschieht dies durch Indizierung der Abkürzungen innerhalb der spitzen Klammern (z.B. <NAusdr1>).

::= (...wird definiert als...)

Das links von diesem Definitionszeichen stehende Sprachelement wird durch den rechts davon angegebenen syntaktischen Ausdruck definiert (erklärt).

| (entweder...oder...)

Die vor und nach diesem Zeichen stehenden Sprachelemente schließen sich gegenseitig aus; z.B. <Bu>|<Zi> bedeutet: "entweder ein Buchstabe oder eine Ziffer" muß hier angegeben werden.

Elemente der Syntax

{ } (geschweifte Klammer auf/zu)

Sprachelemente, die innerhalb dieser Klammern aufgeführt sind, müssen an dieser Stelle auftreten ("Muß-Angabe").

[ ] (eckige Klammer auf/zu)

Sprachelemente, die innerhalb dieser Klammern aufgeführt sind, können an dieser Stelle auftreten, d.h. sie können hier auch weggelassen werden ("Kann-Angabe").

Für beide der eben genannten Klammertypen gilt:

Angaben, die untereinander stehen, schließen sich gegenseitig aus.
Beispiel: ... ⟨Bu⟩ ...
⟨Zi⟩

bedeutet, daß an dieser Stelle entweder ein Buchstabe oder eine Ziffer geschrieben werden muß.
(Bei eckigen anstelle der geschweiften Klammern entsprechend "kann" statt "muß".)

$\{\}_2^8$ $[]^4$ (minimale/maximale Mehrfachangabe)

Ein hochstehender Index an einer geschweiften bzw. eckigen "Klammer zu" gibt an, wie oft die Muß- bzw. Kann-Angabe an dieser Stelle höchstens hintereinander aufgeführt werden darf.

Ein tiefstehender Index an einer geschweiften "Klammer zu" gibt an, wie oft die Muß-Angabe an dieser Stelle mindestens hintereinander aufgeführt werden muß.

Wird kein Index angegeben, so bedeutet dies dasselbe wie Angabe "1".
Die Angabe "n" als hochstehender Index bedeutet "beliebig oft" ("n-mal").

Elemente der Syntax

Bsp.: ...$\{\langle Zi\rangle\}_1^4$ ...

bedeutet, daß an dieser Stelle mindestens eine Ziffer geschrieben werden muß, höchstens jedoch vier.

...$[\langle Bu\rangle]^4$ ...

bedeutet, daß an dieser Stelle nicht unbedingt Buchstaben geschrieben werden müssen (Kann-Angabe); wenn jedoch, dann dürfen höchstens vier auftreten.

Alle übrigen Zeichen außerhalb der spitzen Klammern sind Endsymbole und müssen stets geschrieben werden.

Anmerkung:

Falls für die Darstellung eines syntaktischen Elements eine Zeile nicht ausreicht, wird sie in der Folgezeile - eingerückt - fortgesetzt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Elemente der Syntax

2.2 Liste der Syntaxelemente

- ⟨Bu⟩ ::= A|B|C .... X|Y|Z

  ⟨Bu⟩ ist die Menge der Buchstaben des Alphabets.

- ⟨Zi⟩ ::= 0|1 ... 8|9

  ⟨Zi⟩ ist die Menge der Dezimalziffern.

- ⟨NOper⟩ ::= +|-|*|/|↑

  ⟨NOper⟩ sind die Operatoren für Addition (+), Subtraktion (-), Multiplikation (*), Division (/) und Potenzierung (↑).

- ⟨VglOper⟩ ::= ⟨|⟩|=|⟩=|⟨=|⟨⟩

  Die Vergleichsoperatoren bedeuten (von links nach rechts): "kleiner als", "größer als", "gleich", "größer als oder gleich", "kleiner als oder gleich" und "ungleich".

- ⟨Format⟩ ::= 1%|2%|3%|4%

  ⟨Format⟩ gibt die Speicherplatzgröße für zu deklarierende Variablen an.

- ⟨Ganzzahl⟩ ::= [+|-]{⟨Zi⟩}^{n}

  Eine ganze Zahl ist eine Aneinanderreihung von Dezimalziffern, eventuell mit Vorzeichen.

  Beispiele: 9 -123456789012345 0815

Elemente der Syntax

- ⟨Dezzahl⟩ ::= [+|-][{⟨Zi⟩}$_1^n$ [.[{⟨Zi⟩}^{n}]] | .{⟨Zi⟩}$_1^n$] [E[+|-]{⟨Zi⟩}$_1^2$]

  Eine Dezimalzahl ist entweder eine ganze Zahl oder eine Kette von Dezimalziffern, der ein Punkt vorausgehen oder folgen kann oder innerhalb derer an einer beliebigen festen Stelle ein Dezimalpunkt steht.
  Zusätzlich kann mit dem Buchstaben E und einer ein- oder zweistelligen Ganzzahl der Wert dieser Zahl mit 10 hoch dieser Ganzzahl multipliziert werden.

  Beispiele: 99 1. .6789E-5 0.05

- ⟨Zlnr⟩ ::= {⟨Zi⟩}^{5}

  Eine Zeilennummer ist eine aus 1 bis 5 Dezimalziffern bestehende Zahl.
  Es sind Anweisungsnummern von 1 bis 65535 möglich.

- ⟨NVarName⟩ ::= ⟨Bu⟩|⟨Bu⟩⟨Zi⟩
  ⟨SVarName⟩ ::= ⟨NVarName⟩$
  ⟨VarName⟩ ::= ⟨NVarName⟩|⟨SVarName⟩

  Ein numerischer Variablenname besteht aus einem Buchstaben oder einem Buchstaben gefolgt von einer Ziffer. Ein Stringvariablenname besteht aus einem numerischen Variablennamen, gefolgt von einem $-Zeichen. Ein Variablenname ist ein numerischer oder ein String-Variablenname.

  Beispiele: A0 P Z9 E$ E1$

- ⟨ProgName⟩ ::= [⟨LU⟩/] ⟨Bu⟩ [⟨Bu⟩|⟨Zi⟩|.|-]^{13}

  Ein BASIC-Programm wird durch seinen Namen und die Nummer der logischen Platteneinheit, auf der sich das Programm befindet, festgelegt. Die logische Platteneinheit 0 braucht nicht angegeben werden. Ein Programmname muß mit einem Buchstaben beginnen, gefolgt von bis zu 13 Buchstaben, Ziffern, Punkten oder Bindestrichen.

- ⟨DatName⟩ ::= ⟨ProgName⟩

  ⟨DatName⟩ ist syntaktisch identisch mit dem Programmnamen, bezeichnet jedoch eine Datendatei.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Elemente der Syntax

- <DriverName> ::= $ [<Bu>|<Zi>|.|-][13]

  Eine Driverdatei wird durch den Drivernamen beschrieben. Ein Drivername muß mit einem $-Zeichen beginnen, gefolgt von bis zu 13 Buchstaben, Ziffern, Punkten oder Bindestrichen.

- <Datei> ::= <ProgName> [!] / <Drivername>

  Eine Datei kann entweder eine Platten- oder eine Driverdatei sein. Wenn eine bereits bestehende Plattendatei ersetzt werden soll, ist die Angabe "!" erforderlich.

  Beispiele: CL1 TEST-S.0 $LPT

- <NVar> ::= <NVarName> [(<NAusdr> [,<NAusdr>])]

  Numerische Variablen stehen für dezimale Zahlenwerte. Sie bestehen aus einem numerischen Variablennamen, dem - in runde Klammern eingeschlossen - ein oder zwei durch ein Komma getrennte arithmetische Ausdrücke (Indexausdrücke) folgen können.

  Beispiele: A0 P(1) Z9(15,0) F(A*A,1+Y)

- <SVar> ::= <SVarName> [(<NAusdr> [,<NAusdr>])]

  String-Variablen stehen für Strings (Zeichenketten). Ihr syntaktischer Aufbau ist fast genau wie derjenige der arithmetischen Variablen, jedoch muß zur Kennzeichnung der String-Variablen ein Stringvariablenname angegeben sein. Die Indexausdrücke geben Positionen innerhalb der durch <SVarName> festgelegten Zeichenkette an.

  Beispiele: A0$ P$(1) Z9$(2,4) F$(X,X+10)

- <MVar> ::= <Bu>|<Bu><Zi>

  Matrix-Variable sind syntaktisch genauso aufgebaut wie Variablennamen und bezeichnen Matrizen in ihrer Gesamtheit. Die Matrix-Variable muß vor oder bei ihrem ersten Auftreten durch ihre Dimensionierung als Matrix kenntlich gemacht werden.

Elemente der Syntax

- <NAusdr> ::= [+|-]<Operand> [<NOper><Operand>]^{n}

  Ein numerischer Ausdruck gibt die Regel zur Berechnung eines numerischen Wertes an und wird gebildet durch Verkettung von Operanden mittels arithmetischer Operatoren. Nur der erste Operand kann mit einem Vorzeichen (+ oder -) versehen sein, ohne daß zusätzliche Klammern erforderlich sind.

  Hierbei bedeutet:

  <Operand>::= { FN<Bu>(<NAusdr>) | (<NAusdr>) | <NVar> | <NFkt> | <Dezzahl> }

  d.h. als Operanden können auftreten:
  - vom Anwender selbstdefinierte Funktionen
  - (in runde Klammern gesetzte) numerische Ausdrücke
  - numerische Variablen
  - numerische Funktionen
  - Dezimalzahlen

- <NFkt>

  Numerische Funktionen liefern stets einen numerischen Wert.
  Genaueres ist dem Kapitel "BASIC-Funktionen" zu entnehmen.

- <NumPar> ::= <NVar>|<Dezzahl>

  Numerische Parameter finden vor allem Verwendung als Parameter in CALL-Anweisungen.

- <Zeich>

  <Zeich> ist der Gesamtvorrat an gültigen Displayzeichen.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Elemente der Syntax

- ⟨OktalZahl⟩

  ⟨OktalZahl⟩ ist eine 3-stellige Oktalzahl von 000-377. Oktalcodes stehen für beliebige Zeichen des Zeichenvorrats und enthalten den ASCII-Code des Zeichens oktal.

- ⟨SLit⟩ ::=

  "[⟨Zeich⟩ außer '' und ← oder ←⟨OktalZahl⟩←]"

  Ein Stringliteral ist eine fest vorgegebene, in Anführungszeichen eingeschlossene, Zeichenkette, bestehend aus Displayzeichen mit Ausnahme von Anführungsstrich und Linkspfeil. Ein Anführungsstrich in einem ⟨SLit⟩ wird durch zwei aufeinanderfolgende Hochkommata ('') dargestellt. Ein Linkspfeil kann nur als ⟨OktalZahl⟩ eingegeben werden.

- ⟨String⟩ ::= ⟨SVar⟩|⟨SLit⟩

  Strings sind Stringvariablen oder Stringliterale.

- ⟨SAusdr⟩ ::= ⟨String⟩ [,⟨String⟩]^{n}

  Ein String-Ausdruck besteht aus einer Folge von String-Konstanten und String-Variablen, durch Kommata getrennt.

- ⟨logAusdr⟩::=

  {⟨NAusdr⟩[⟨VglOper⟩⟨NAusdr⟩]
  {⟨String⟩[,⟨String⟩][⟨VglOper⟩⟨String⟩[,⟨String⟩]]}

  Das Ergebnis der Auswertung eines logischen Ausdrucks ist "wahr" oder "falsch" gemäß den Regeln der Logik.

- ⟨DisFkt⟩

  Die Bildschirmfunktionen (engl.: display = Bildschirm) sind die Funktionen zur Aufbereitung des Bildschirms und zur Cursorpositionierung, wie sie im Kapitel "Tastatur-Bildschirm-Ein-/Ausgabe" beschrieben werden.

- ⟨DevFkt⟩

  Die Gerätefunktionen (engl.: device = Gerät) entsprechen syntaktisch den Bildschirmfunktionen (⟨DisFkt⟩) und werden bei der Programmierung von Gerätedateien zur Steuerung der Peripheriegeräte eingesetzt.

- ⟨StZeich⟩

  Steuerzeichen in DFÜ-Anweisungen zur Auftragsspezifikation. Syntaktisch entsprechen die Steuerzeichen den Bildschirmfunktionen (⟨DisFkt⟩), nur daß statt eines zweistelligen Literals ein dreistelliges in Hochkommata eingefaßt wird (s. "Nixdorf 8870 DFÜ-Programmierhandbuch").

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

3 Kurzliste der Anweisungen und BASIC-Kommandos

3.1 Kurzliste der Anweisungen

BUILD # ⟨NAusdr⟩, [+] ⟨String⟩ [[,#⟨NAusdr⟩],[+] ⟨String⟩]^{n}

Anlegen von Magnetplattendateien

CALL ⟨NAusdr⟩ $\left\{\begin{array}{l}\text{,⟨NumPar⟩}\\ \text{,⟨String⟩}\end{array}\right\}^{12}$

Aufruf von Maschinencode-Unterprogrammen

CHAIN ⟨String⟩

Aufruf eines beliebigen Programms oder Processors

CLEAR ⟨VarName⟩ [,⟨VarName⟩]^{n}

Löschen von Variablenwerten

CLOSE # ⟨NAusdr⟩ [,#⟨NAusdr⟩]^{n}

Schließen von Datenkanälen (Beenden der DFÜ-Verbindung s. "Nixdorf 8870 DFÜ-Programmierhandbuch")

CNTRL # ⟨NAusdr⟩;$\left\{\begin{array}{l}\text{⟨StZeich⟩}\\ \text{⟨StZeich⟩,⟨StZeich⟩}\end{array}\right\}$,⟨SVar⟩,⟨NVar⟩

Steuerung der Datenaufbereitung und der Empfangserlaubnis (DNT) bzw. Steuerung des Datenflusses (SNA), s. "Nixdorf 8870 DFÜ-Programmierhandbuch"

DATA [⟨Format⟩,] ⟨Dezzahl⟩ [,⟨Dezzahl⟩]^{n}

Abstellen von numerischen Konstanten

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Kurzliste der Anweisungen und BASIC-Kommandos

DEALLO ⟨VarName⟩ [,⟨VarName⟩]^{n}

Freigeben von Variablen und Feldern

DEF FN ⟨Bu⟩ (⟨Bu⟩)= ⟨NAusdr⟩

Definition von Anwender-Funktionen

DIM ⟨Feldbeschr⟩ [,⟨Feldbeschr⟩]^{n}

wobei:
⟨Feldbeschr⟩::=⟨NVarName⟩[(⟨NAusdr⟩[,⟨NAusdr⟩])]
⟨SVarName⟩(⟨NAusdr⟩)
⟨Format⟩

Dimensionierung von Variablen und Feldern

END

Beenden eines Programms

ENV # ⟨NAusdr⟩;⟨NVar⟩ [,⟨NVar⟩ / ,⟨SVar⟩]^{n}

Parameterübergabe an OPEN-, READ- oder WRITE-Anweisung
(s. auch Handbuch "Nixdorf 8870/U Datenfernübertragung")

FOR ⟨NVar⟩=⟨NAusdr⟩ TO ⟨NAusdr⟩ [STEP ⟨NAusdr⟩]

Beginn einer Programmschleife

GET # ⟨NAusdr⟩; [⟨StZeich⟩,] ⟨SVar⟩,⟨SVar⟩,⟨NVar⟩,⟨NVar⟩,⟨NVar⟩

Empfangen von Daten und Statusinformationen (s. "Nixdorf 8870 DFÜ-Programmierhandbuch")

Kurzliste der Anweisungen und BASIC-Kommandos

GLOBAL ⟨Feldbeschr⟩ [,⟨Feldbeschr⟩]^n

wobei:
⟨Feldbeschr⟩::= { ⟨NVarName⟩[(⟨Ganzzahl⟩[,⟨Ganzzahl⟩])] | ⟨SVarName⟩(⟨Ganzzahl⟩) | ⟨Format⟩ }

Dimensionierung von globalen Variablen

GOSUB ⟨Zlnr⟩

Unterprogrammsprung

GOTO ⟨Zlnr⟩

Unbedingter Sprung

IF ⟨logAusdr⟩ ⟨Anweisung⟩

Bedingte Ausführung von ⟨Anweisung⟩

IF ERR 0 [⟨Anweisung⟩]

Ausführung von ⟨Anweisung⟩ bei Fehler

IF ESC [⟨Anweisung⟩]

Ausführung von ⟨Anweisung⟩ bei Betätigung der Escape - Taste

INIT # ⟨NAusdr⟩; ⟨SVar⟩,⟨SVar⟩,⟨NVar⟩

Aufbau der physikalischen Verbindung zum gewünschten Partner mit Hilfe des Partner-ID's (s. "Nixdorf 8870 DFÜ-Programmierhandbuch")

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung Ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Kurzliste der Anweisungen und BASIC-Kommandos

INPUT

```
[ <SLit>,                                  ]^n  { <NVar> }
[ <DisFkt>,                                ]    { <SVar> }
[ TAB(<NAusdr> [,<NAusdr>]),               ]
[ <NVar>,                                  ]
[ <SVar>,                                  ]
```

Eingabe über Tastatur

KILL `<String> [,<String>]^n`

Löschen von Dateien

[LET]

```
{ <NVar>= { <NAusdr> }                                  }
{         { <String> }                                  }
{ <SVar>= { <SAusdr>                                    } }
{         { <NAusdr> [USING <String>]                   } }
```

Zuweisung an eine Variable (LET ist Muß-Angabe in einer IF-Anweisung)

LINK `[;<VarName>,] <String>`

Aufruf eines BASIC-Programms

LGERR

Eintrag von BASIC-Fehlern ins Systemlogbuch

LOFF # `<NAusdr>; <SVar>,<SVar>,<NVar>`

Abbau der logischen Verbindung (s. "Nixdorf 8870 DFÜ-Programmierhandbuch")

LON # `<NAusdr>; <SVar>,<SVar>,<NVar>`

Aufbau der logischen Verbindung (s. "Nixdorf 8870 DFÜ-Programmierhandbuch")

Kurzliste der Anweisungen und BASIC-Kommandos

MAT ⟨MVar⟩=
- ⟨MVar⟩
- ⟨MVar⟩ {+ | - | *} ⟨MVar⟩
- (⟨NAusdr⟩)*⟨MVar⟩
- INV(⟨MVar⟩)
- MIV(⟨MVar⟩)
- TRN(⟨MVar⟩)
- ZER[(⟨NAusdr⟩ [,⟨NAusdr⟩])]
- CON[(⟨NAusdr⟩ [,⟨NAusdr⟩])]
- IDN[(⟨NAusdr⟩,⟨NAusdr⟩)]

Matrix-Zuweisung

MAT INPUT ⟨MVar⟩ [(⟨NAusdr⟩ [,⟨NAusdr⟩])]
[,⟨MVar⟩ [(⟨NAusdr⟩ [,⟨NAusdr⟩])]]^{n}

Tastatureingabe in eine Matrix

MAT PRINT ⟨MVar⟩[,|;]

Anzeigen einer Matrix am Bildschirm

MAT READ ⟨MVar⟩ [(⟨NAusdr⟩ [,⟨NAusdr⟩])]
[,⟨MVar⟩ [(⟨NAusdr⟩ [,⟨NAusdr⟩])]]^{n}

Konstante in eine Matrix einlesen

MAT READ #⟨NAusdr⟩ [,⟨NAusdr⟩ [,⟨NAusdr⟩]]; ⟨VarName⟩ [;]

Aus einer Datei in eine Variable einlesen

MAT WRITE #⟨NAusdr⟩ [,⟨NAusdr⟩ [,⟨NAusdr⟩]]; ⟨VarName⟩ [;]

Aus einer Variablen in eine Datei übertragen

NEXT ⟨NVar⟩

Ende einer Programmschleife

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Kurzliste der Anweisungen und BASIC-Kommandos

ON

⟨NAusdr⟩ { GOTO / GOSUB } ⟨Zlnr⟩ [,⟨Zlnr⟩]^{n}

Berechneter Sprung

OPEN #

⟨NAusdr⟩ [;⟨String⟩], ⟨String⟩
[[,#⟨NAusdr⟩ [;⟨String⟩]], ⟨String⟩]^{n}

Eröffnung von Datenkanälen (Eröffnung eines DNT- bzw. SNA-Drivers s. "Nixdorf 8870 DFÜ-Programmierhandbuch")

{PRINT / ;} [USING ⟨String⟩;] [⟨NAusdr⟩ / ⟨String⟩ / ⟨DisFkt⟩] [{;/,} {⟨NAusdr⟩ / ⟨String⟩ / ⟨DisFkt⟩}]^{n} [;/,]

Bildschirmausgabe (PRINT muß ausgeschrieben werden in einer IF-Anweisung)

{PRINT / ;}# ⟨NAusdr⟩ [,⟨NAusdr⟩ [,⟨NAusdr⟩]];

[USING⟨String⟩;] [⟨NAusdr⟩ / ⟨String⟩ / TAB(⟨NAusdr⟩) / ⟨DevFkt⟩] [{;/,} {⟨NAusdr⟩ / ⟨String⟩ / TAB(⟨NAusdr⟩) / ⟨DevFkt⟩}]^{n} [;/,]

Ausgabe in Magnetplatten- oder Gerätedateien (PRINT muß ausgeschrieben werden in einer IF-Anweisung)

PUT #

⟨NAusdr⟩; [⟨StZeich⟩,] [⟨StZeich⟩,] ⟨Svar⟩,⟨SVar⟩,⟨NVar⟩

Senden von Daten und Statusinformationen (s. "Nixdorf 8870 DFÜ-Programmierhandbuch")

RANDOM

⟨NAusdr⟩

RND-Funktion steuern

READ

⟨NVar⟩ [,⟨NVar⟩]^{n}

Konstante lesen

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Kurzliste der Anweisungen und BASIC-Kommandos

READ # ⟨NAusdr⟩ [,⟨NAusdr⟩ [,⟨NAusdr⟩]]; {⟨NVar⟩ / ⟨SVar⟩} [,⟨NVar⟩ / ,⟨SVar⟩]ⁿ [;]

Lesen aus Magnetplatten- und Gerätedateien

REM [⟨Zeich⟩]ⁿ

Im Anschluß an jede Anweisung kann hinter die Zeichenfolge /* ein Kommentar eingegeben werden

RESTOR [⟨Zlnr⟩]

Konstantenzeiger zurücksetzen

RETURN [⟨NAusdr⟩]

Rückkehr aus Unterprogrammen

SEARCH # ⟨NAusdr⟩,⟨NAusdr⟩,⟨NAusdr⟩;⟨SVar⟩,⟨NVar⟩,⟨NVar⟩ [:|;]

Schlüsselverzeichnisbearbeitung in Indexdateien

SIGNAL { 1,⟨NAusdr⟩,⟨NAusdr⟩,⟨NAusdr⟩ / 2,⟨NVar⟩,⟨NVar⟩,⟨NVar⟩ [,⟨NAusdr⟩] / 3,⟨NAusdr⟩ }

Signalliste bearbeiten und Warteanweisung

STOP

Programmunterbrechung

WRITE # ⟨NAusdr⟩ [,⟨NAusdr⟩ [,⟨Nausdr⟩]]; {⟨NAusdr⟩ / ⟨String⟩} [,⟨NAusdr⟩ / ,⟨String⟩]ⁿ

Schreiben in Magnetplatten- oder Gerätedateien

Kurzliste der Anweisungen und BASIC-Kommandos

3.2 Kurzliste der BASIC-Kommandos

[<Zlnr>] ANALYSE <Datei> [<Zlnr>] [↑<Ganzzahl>]

Auflisten des für Optimierungszwecke markierten Programms auf dem Drucker oder in einer Textdatei

---

[<Zlnr>] AUTOLN [<Ganzzahl>]

Automatische Erzeugung von Zeilennummern

[<Zlnr>] COMMENT [<ProgName> [!]] [<Zlnr>] [REM [<LU>]]

Eingliedern von Kommentaren in ein Programm

CREF { <Zlnr> / <VarName> / #<Ganzzahl> / #? / STAT } Ausgabe von Referenzen auf dem Bildschirm

---

[<Zlnr>] DEBUG [<Datei>]

Anweisungsfolge im Einzelschrittverfahren testen oder durchlaufene Anweisungen auf Drucker oder in Textdatei auflisten.

{ <Zlnr> / [<Zlnr>] DELETE [<Zlnr>] }

Löschen von Anweisungen

[<Zlnr>] DUMP <Datei> [<Zlnr>] [↑<Ganzzahl>]

Auflisten des Programmes auf dem Drucker oder in einer Textdatei

---

[<Zlnr>] EDIT

Anzeigen eines Statements auf dem Bildschirm

Kurzliste der Anweisungen und BASIC-Kommandos

HASH [<Zlnr>]

Prüfsummenberechnung mit Anzeige auf dem Bildschirm

HELP [<Ganzzahl>]

BASIC-Fehler im Klartext auf dem Bildschirm anzeigen

[<Zlnr>] LIST [<Zlnr>]

Auflisten von Anweisungen auf dem Bildschirm

LOAD <ProgName>

Laden eines Programms aus einer Textdatei

NEW

Löschen des aktuellen BASIC-Programms in der Partition

<Zlnr> PARTRENUM <Zlnr>

Teilweise Neunumerieren der Anweisungen eines Programms

[<Zlnr>] PLIST <Datei> [<Zlnr>] [,<Ganzzahl>] [REM[<LU>]]

Auflisten des Programms mit Crossreference auf dem Drucker oder in einer Textdatei

---

[<Zlnr>] RENUMBER [<Ganzzahl>]

Neunumerieren der Anweisungen eines Programms

[<Zlnr>]RUN [<Zlnr> [,<Zlnr>]^4]

Starten des aktuellen Programms

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Kurzliste der Anweisungen und BASIC-Kommandos

SAVE [,D,|,B,] [#<PartG>#] [<ProgName> [!]]

Auslagern des im Hauptspeicher stehenden Programms auf Magnetplatte (<PartG>: Partitiongröße in KByte)

SIZE

Anzeigen der Programmgröße

```
{[<Zlnr1>] TRACE [<Zlnr2>] {{,<VarName>}^5 }}
{                          {{,<Bedingung>}^2}}
{                                            }
{ TRACE [<Datei>]                            }
```

wobei:

```
                  {<NVarName1><VglOper>{<NVarName2>}                 }
                  {                    {<Dezzahl>  }                 }
<Bedingung>::={<NVarName>{=|<>}<Dezzahl1>:<Dezzahl2>}
                  {<SVarName1><VglOper>{<SVarName2>}                 }
                  {                    {<SLit>     }                 }
```

Programmstop bei vorgegebenen Variablenwerten. Ausgabe aller Variablenwerte.

[<Zlnr>] UNCOMMENT [<ProgName> [!]] [<Zlnr>] [REM [<LU>]]

Ausgliedern von Kommentaren aus einem Programm

Die BASIC-Kommandos

4 Die BASIC-Kommandos

Zur Unterstützung der Programmierung in Business-BASIC stellt der BASIC-Processor folgende BASIC-Kommandos zur Verfügung:

- ANALYSE
- AUTOLN
- COMMENT
- CREF
- DEBUG
- DELETE
- DUMP
- EDIT
- HASH
- HELP
- LIST
- LOAD
- NEW
- PARTRENUM
- PLIST
- RENUMBER
- RUN
- SAVE
- SIZE
- TRACE
- UNCOMMENT

Diese Kommandos werden aus dem BASIC-Kommandomodus aufgerufen, der durch die Positionierung des Cursors zu Beginn einer Zeile angezeigt wird.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

4.1 Das Kommando ANALYSE

Das Kommando ANALYSE unterstützt den Anwender bei der Umstellung von Programmen, die möglichst weit compiliert werden sollen. Es analysiert Programme statisch und markiert Programmierungen, die ungünstig für die Compilation sind, (Vergl. auch SAVE,D,...).

Syntax:

[<Zlnr1>] ANALYSE <Datei> [<Zlnr2>] [↑<Ganzzahl>]

<Zlnr1> : Zeilennummer der ersten Anweisung, die analysiert werden soll. Fehlt diese Angabe, beginnt die Ausgabe mit der ersten Anweisung des Programms.

<Datei> : Der Name der Textdatei, in die ausgegeben werden soll, oder der Name des Druckers, auf dem das Programm ausgedruckt werden soll.

<Zlnr2> : Die Nummer der letzten zu analysierenden Anweisung. Fehlt diese Angabe, endet die Ausgabe mit Programmende.

↑<Ganzzahl>: Anzahl pro Seite. (24-99)
Fehlt diese Angabe, wird die Standardhöhe 48 benutzt.

Die Programm-Analyse gibt eine Programmliste ohne Kommentare aus. Eine Kennzeichnung der referierten Zeilennummern in der Liste entfällt, ebenso wird keine Referenzliste für Anweisungen ausgegeben. In der Programmliste werden stattdessen folgende Markierungen angebracht:

DA Alle DEALLO-Anweisungen, die nicht direkt vor LINK im Programmlisting stehen, (deallocate).

Die betroffenen Variablen werden im Folgeprogramm unter dem gleichen Namen weiterverwendet.

LC Alle DEALLO-Anweisungen vor LINK im Programmlisting (local).

Solche Variablen sind bei der Umstellung eines Programmes als lokale Variablen zu definieren.

Die BASIC-Kommandos

Ferner wird die Cross-Reference-List der Variablen mit folgenden Markierungen ausgegeben, treffen mehrere zu, werden sie hintereinander gesetzt:

FP Alle explizit dimensionierten Variablen, die nicht am Programmanfang stehen, (false position).

VA Alle mit DIM dimensionierten Variablen, die keine ganzzahlige Feldbeschreibung haben oder in einer DIM-Anweisung gemeinsam mit solchen Variablen stehen, (variable argument).

DA Alle Variablen der DEALLO-Anweisung, die nicht direkt vor LINK stehen (deallocate).

LC Alle Variablen der DEALLO-Anweisung, die vor LINK stehen, (local).

- Beispiel:

  ANALYSE $LPT ↑ 72

  erzeugt eine Programmliste zur Analyse des Programms, in dem Sie sich befinden. Die Ausgabe der Liste erfolgt auf dem Drucker $LPT mit der Zeilenhöhe 72 Zeilen pro Seite.

Die Funktion der Kommandos kann für ein oder mehrere Programme auch über den TAMOS-Selektor angewählt werden (siehe Kap. "Programmwartung").

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

4.2 Das Kommando AUTOLN

Das Kommando AUTOLN löst die automatische Erzeugung von Zeilennummern auf dem Bildschirm aus.

Syntax:

[⟨Zlnr⟩] AUTOLN [⟨Ganzzahl⟩]

⟨Zlnr⟩ : Beginn der automatischen Zeilenerzeugung mit dieser Zeilennummer

⟨Ganzzahl⟩: Schrittweite

Fehlt die Angabe ⟨Zlnr⟩, beginnt die Zeilenerzeugung bei 10.
Fehlt die Angabe ⟨Ganzzahl⟩, gilt die Schrittweite 10.

4.3 Das Kommando COMMENT

Mit dem Kommando COMMENT werden Kommentare eines Programms, die zuvor mit dem Kommando UNCOMMENT ausgelagert wurden, wieder in den Programmtext eingefügt.

Syntax:

[⟨Zlnr1⟩] COMMENT [⟨ProgName⟩ [!]]
[⟨Zlnr2⟩] [REM [⟨LU2⟩]]

⟨Zlnr1⟩ : Zeilennummer der ersten Anweisung, von der mit der Einlagerung von Kommentaren begonnen werden soll. Fehlt diese Angabe, geht die Einlagerung vom Programmanfang aus.

⟨ProgName⟩: Mit dem Namen ⟨ProgName⟩ wird das in der Partition stehende Programm um die Kommentare erweitert und gesichert. Der eigentliche Name darf höchstens 12 Zeichen lang sein, da die Kommentare aus der Textdatei des gleichen Namens, aber mit den Anfangsbuchstaben Y. entnommen werden. Ist ⟨ProgName⟩ nicht angegeben, wird der Name des in der Partition liegenden Programmes genommen.

! : Ein "!" ist erforderlich, falls ein Programm mit dem Namen ⟨ProgName⟩ schon existiert und überschrieben werden soll.

Die BASIC-Kommandos

4

<Zlnr2> : Zeilennummer der letzten Anweisung, bis zu der die Einlagerung von Kommentaren ausgeführt werden soll. Fehlt diese Angabe, geschieht die Einlagerung bis zum Programmende.

REM[<LU2>]: Nummer der logischen Platteneinheit, auf der die Textdatei mit den Kommentaren liegt. Ist <LU2>=0, kann diese Angabe entfallen.

Eine mit UNCOMMENT erzeugte Textdatei darf nicht mit dem Kommando LOAD zum reduzierten Programm dazugeladen werden, da Programmzeilen überschrieben werden können.

Werden Änderungen an einem Programm vorgenommen, dessen Kommentare ausgelagert sind, kann man für die Änderungen versehentlich Zeilennummern verwenden, die auch von ausgelagerten Kommentaren benutzt werden. Beim anschließenden Dazuladen mit "COMMENT" werden solche Kommentare als /*-Kommentare an die entsprechenden Änderungszeilen angehängt.

Achtung:

Mit dem COMMENT-Kommando wird die Textdatei mit den Kommentaren stets gelöscht, auch wenn durch Angabe von <Zlnr1> oder <Zlnr2> nicht alle Kommentare eingelagert werden. Falls die Gefahr des versehentlichen Löschens von Kommentaren besteht, sollte eine Kopie der Textdatei erstellt werden.

Die Funktion des Kommandos kann für ein oder mehrere Programme auch über den TAMOS-Selektor angewählt werden (siehe Kap. "Programmwartung").

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

4.4 Das Kommando CREF

Das Kommando CREF gibt die zugehörigen Referenzen zu jeweils einer Anweisungsnummer, einer Variabeln oder einer Kanalnummer auf dem Display aus. Bei der Angabe STAT wird die Programmstatistik ausgegeben.

Syntax:

```
        ⎧ <Zlnr>          ⎫
        ⎪ <VarName>       ⎪
CREF    ⎨ #<Ganzzahl>     ⎬
        ⎪ #?              ⎪
        ⎩ STAT            ⎭
```

Indizierungen bei <VarName> sind nicht zugelassen, da sie für die Referenzen nicht relevant sind.
Bei der Angabe #? werden alle Kanalnummern angezeigt, die durch Variablen oder numerische Ausdrücke definiert worden sind.

4.5 Das Kommando DEBUG

Das Kommando DEBUG ermöglicht dem Programmierer, eine Anweisungsfolge im Einzelschrittverfahren zu testen.

Syntax:

```
[<Zlnr>] DEBUG [<Datei>]
```

<Zlnr> : Zeilennummer der Anweisung, womit der Testlauf begonnen werden soll.
Ist <Zlnr> nicht angegeben, startet DEBUG mit der ersten ausführbaren Anweisung des Programms.

<Datei> : Angabe eines Druckers oder einer Textdatei als Ausgabemedium. Fehlt diese Angabe, wird die Meldung am Bildschirm angezeigt.

Die Angabe <Zlnr> ist nur dann sinnvoll, wenn bereits ein initialisierendes RUN- oder DEBUG-Kommando durchgeführt wurde.

Die BASIC-Kommandos

DEBUG ohne Angabe von ⟨DatName⟩ initialisiert den Einzelschritt-Test (DEBUG-Modus) und meldet:

STOP AT xxx ⟨Anweisung⟩

wobei mit xxx die Zeilennummer der nächsten Anweisung und die Anweisung selbst ausgegeben wird.

Der BASIC-Processor befindet sich weiter im DEBUG-Modus, und der Bediener kann durch Drücken der Taste "CR" die Ausführung der angezeigten Anweisung und die Ausgabe der nächsten Anweisung auslösen.

Die Eingabe jedes anderen Kommandos ungleich DEBUG oder TRACE oder die Eingabe einer BASIC-Anweisung mit Zeilennr. löschen den DEBUG-Modus. Direktausführung von Anweisungen oder Betätigung der ESC-Taste löschen den DEBUG-Modus nicht.
Das DEBUG-Kommando mit Angabe von ⟨DatName⟩ bewirkt ein Durchlaufen des Programms (ohne Zwischenstops wie beim Einzelschritt-Test) und die Ausgabe der durchlaufenen Anweisungen in die angegebene Datei bzw. auf dem angegebenen Drucker.

- Beispiel:

```
300 DEBUG
```

Beginnen des DEBUG-Modus bei der Zeilennummer 300 und Anzeigen dieser Anweisung. Nach Betätigung der 'CR'-Taste wird diese Anweisung ausgeführt und die folgende Anweisung angezeigt.

ERROR # 6 "ZEILENNUMMER NICHT EXISTENT"

wird gemeldet, wenn die im DEBUG-Kommando angegebene Zeilennummer nicht vorhanden ist.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

4.6 Das Kommando DELETE

Das Kommando DELETE ermöglicht es, aus einem BASIC-Programm eine beliebige Anweisungsfolge zu löschen.

Syntax:

```
{ <Zlnr>                              }
{[<Zlnr1>] DELETE [<Zlnr2>]           }
```

<Zlnr1> : Zeilennummer der Anweisung, die als erste gelöscht werden soll.
Fehlt diese Angabe, wird ab der ersten Anweisung im Programm gelöscht.

<Zlnr2> : Zeilennummer der letzten zu löschenden Anweisung.
Fehlt die Angabe <Zlnr2>, wird bis zur letzten Anweisung des Programms (einschließlich) gelöscht.

Ist nur DELETE angegeben, wird das gesamte Programm aus dem Hauptspeicher gelöscht. Es erzeugt in dieser Form den gleichen Zustand wie das Kommando "NEW".

Wird nur <Zlnr> angegeben (ohne DELETE), wird nur diese eine Anweisung gelöscht.

Beim Löschen größerer Programmteile kann die Ausführungszeit mehrere Sekunden betragen. Das Ende der Funktion ist daran zu erkennen, daß der Cursor auf der ersten Stelle der nächsten Bildschirmzeile positioniert ist.

- Beispiele:

  - 490
    Löschen der Anweisung mit der Zeilennummer 490.

  - 100 DELETE 200
    Löschen der Anweisungen von 100 bis 200 (einschließlich).

Die BASIC-Kommandos

4

4.7 Das Kommando DUMP

Das Kommando DUMP gibt das im Hauptspeicher stehende BASIC-Programm in eine Textdatei auf der Platte oder auf einen Drucker aus.

Syntax:

[<Zlnr1>] DUMP <Datei> [<Zlnr2>] [↑<Ganzzahl>]

<Zlnr1> : Zeilennummer der ersten Anweisung, die ausgegeben werden soll. Fehlt diese Angabe, beginnt die Ausgabe mit der ersten Anweisung des Programms.

<Datei> : Der Name der Textdatei, in die ausgegeben werden soll, oder Name des Druckers, auf dem das Programm ausgedruckt werden soll.

<Zlnr2> : Die Nummer der letzten auszugebenden Anweisung.

<Ganzzahl>: Anzahl Zeilen pro Seite.
Zu berücksichtigen ist, daß die ersten 3 Zeilen immer Leerzeilen sind.
Fehlt diese Angabe, wird die Standardseitenhöhe 48 benutzt.

Wird "↑" und die Anzahl Zeilen pro Seite angegeben, erfolgt der "DUMP" mit Seitentrennung. Fehlt diese Angabe, wird der "DUMP" ohne Seitentrennung durchgeführt. Dies gilt sowohl für das Ausdrucken als auch für die Ausgabe in eine Textdatei.

In einer Textdatei wird die Seitentrennung durch Abstellen des Oktal-Codes "←214←" (Top of Form) realisiert.

Der "Top of Form"-Code wird vom BASIC-Processor ausgegeben, sobald die angegebene Seitenhöhe -3 Zeilen erreicht ist.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

- Beispiel:

```
500 DUMP $LPT↑48
```

Ausdrucken des Programmes ab der Anweisung mit der Zeilennummer 500 auf dem Nadeldrucker mit einer Blatthöhe von 48 Zeilen und einem Seitenwechsel nach 45 Zeilen.

Fehlermeldungen, die auf der Nachrichtenzeile erscheinen, sind im im Handbuchmodul "Fehlermeldungen" beschrieben.

4.8 Das Kommando EDIT

Mit EDIT wird ein einzelnes BASIC-Statement, das sich im Speicher befindet, als Voreingabe für die Tastatur auf dem Display angezeigt, so daß es unmittelbar manipuliert werden kann. Es werden maximal 251 Zeichen auf dem Bildschirm dargestellt. Der Cursor steht am Anfang der Bildschirmzeile.

Syntax:

```
[<Zlnr>] EDIT
```

Falls kein Statement mit dieser Zeilennummer existiert, wird das mit der nächsthöheren Zeilennummer bearbeitet.

Wenn die Zeilennummer bei einem Statement geändert wird, so existiert dieses sowohl unter der alten als auch unter der neuen Zeilennummer.

An ladbaren BA's in alten Design (RAP) werden maximal 80 Zeichen in den Voreingabepuffer übernommen.

4

4.9 Das Kommando HASH

Dieses Kommando bewirkt die Berechnung des HASH-Total des im Hauptspeicher befindlichen BASIC-Programms ohne Benutzung eines Textfiles. Der Hashcode ist identisch mit dem, den der LIBR-Prozessor über das zugehörige Textfile ermittelt.

Syntax:

HASH [<Zlnr>]

Die Berechnung wird bis zum Statement mit der angegebenen Zeilennummer (einschl.) durchgeführt. Fehlt diese Angabe, werden alle Anweisungen bis Zlnr=49999 berücksichtigt.

Das Ergebnis wird in den Bytes 4 bis 10 rechtsbündig im Common-Bereich des benutzten BA's im ASCII-Format abgestellt und dezimal am Bildschirm angezeigt.

Das HASH-Total eines B-Files ist identisch mit dem eines D-Files.

Zu beachten ist jedoch folgendes:

Ursprünglich erfaßte BASIC-Quellen unterscheiden sich von Quellen, die über Compilation und anschließender Decompilation entstehen (Erfassung eines B-Files - SAVE,D,... - DUMP ...). Durch diese Eigenschaft kann der HASH der Ersterfassung von den Quellen, die über eine Decompilation entstanden sind, abweichen. Um eindeutige HASHes zu gewährleisten, muß daher immer erst ein SAVE,D... gemacht werden. Natürlich haben diese Unterschiede in dem Quellprogramm keinen Einfluß auf die Semantik.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

4.10 Das Kommando HELP

Mit HELP wird die Ausgabe des Fehlertextes der angegebenen Fehlernummer (ERROR #) bzw. der zuletzt aufgetretenen Fehlernummer erzeugt.

Syntax:

HELP [<Ganzzahl>]

<Ganzzahl> : Fehlernummer

Ist eine Fehlernummer angegeben, erscheint der Text, der dieser Nummer entspricht.
Wird keine Fehlernummer angegeben, erscheint der Fehlertext der zuletzt aufgetretenen ERROR-Nummer.

- Beispiel:

```
HELP 1

Ausgabe: SYNTAKTISCHER FEHLER
```

Ist die eingegebene Nummer nicht belegt, erfolgt die Meldung:

NACHRICHT NICHT VORHANDEN ODER FALSCHE NUMMER

4.11 Das Kommando LIST

Das Kommando LIST erzeugt das Anzeigen des im Hauptspeicherbereich des aufrufenden Teilnehmers stehenden BASIC-Programms am Bildschirm in aufsteigender Reihenfolge.

Syntax:

[<Zlnr1>] LIST [<Zlnr2>]

<Zlnr1> : Zeilennummer der ersten anzuzeigenden Anweisung. Fehlt diese Angabe, beginnt die Ausgabe mit der ersten Anweisung des Programms.

<Zlnr2> : Zeilennummer der letzten anzuzeigenden Anweisung. Fehlt diese Angabe, wird bis zur letzten Anweisung angezeigt.

Fehlen die Angaben ⟨Zlnr1⟩ und ⟨Zlnr2⟩, wird das gesamte Programm angezeigt.

Durch Drücken der Taste "ESC" kann die laufende Anzeige beendet werden.
Mit der Leertaste wird die laufende Anzeige unterbrochen (weiter mit "CR").

- Beispiel:

  100 LIST

  Anzeigen des Programms ab der Anweisung mit der Zeilennummer 100 bis zum Programmende bzw. bis zum Abbruch durch Drücken der "ESC"-Taste.

4.12 Das Kommando LOAD

Mit dem Kommando LOAD wird ein BASIC-Programm oder ein Teil eines BASIC-Programms, das mit dem Kommando DUMP in einer Textdatei abgestellt wurde, in den Hauptspeicher geladen.

Syntax:

LOAD ⟨ProgName⟩

⟨ProgName⟩: Name der Textdatei, aus der das Programm geladen werden soll.

Befindet sich zum Zeitpunkt der Ausführung ein anderes Programm im Hauptspeicher, werden beide Programme gemäß ihren Zeilennummern vermischt. Treten in beiden Programmteilen identische Zeilennummern auf, werden die betreffenden Anweisungen des im Speicher befindlichen Programms durch die der Textdatei ersetzt.

Das mit LOAD dazugeladene Programm wird in den Code desselben Filetyps (B oder D) übersetzt wie der des in der Partition liegenden Programms. Befindet sich kein Programm im Hauptspeicher, wird ein Programm vom Typ B erzeugt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

Diese Möglichkeit des Mischens von im Hauptspeicher befindlichen Programmteilen mit in Textdateien befindlichen Programmteilen unterstützt das Einbinden von Standardroutinen in Anwenderprogramme.
Bei der Ausführung des LOAD-Kommandos werden Format-Prüfungen durchgeführt.
Werden dabei Fehler festgestellt, erfolgt am Bildschirm eine Anzeige der entsprechenden Fehlernummer.
Die Routine arbeitet trotzdem weiter, lädt aber die falschen Anweisungen nicht in den Hauptspeicher.

4.13 Das Kommando NEW

Dieses Kommando löscht und initialisiert den Hauptspeicherbereich (Partition) des aufrufenden Teilnehmers.

Syntax:

```
NEW
```

Warnung:

Jede Eingabe im BASIC-Kommandomodus, die mit NEW beginnt, löscht den Hauptspeicherbereich des Anwenders!

4.14 Das Kommando PARTRENUM

Dieses Kommando ermöglicht ein Teil-RENUMBER. Es wird eingesetzt, wenn es nicht mehr möglich ist, zwischen zwei Anweisungen eine weitere einzufügen.

Syntax:

```
<Zlnr1> PARTRENUM <Zlnr2>
```

<Zlnr1> : Zeilennummer, bei der das Teil-RENUMBER beginnen soll

<Zlnr2> : Zeilennummer, bei der das Teil-RENUMBER enden soll

Die Schrittweite der neuen Zeilennumerierung kann nicht angegeben werden. Diese wird automatisch nach der folgenden Formel berechnet.

Step = (⟨Zlnr2⟩ - ⟨Zlnr1⟩) / (Anzahl der ⟨Zlnr⟩ - 1)

Der errechnete Wert wird auf eine ganze Zahl abgerundet. Das Ergebnis ist der maximale, konstante Zeilennummernabstand der Anweisungen zwischen den beiden angegebenen Zeilennummern.

4.15 Das Kommando PLIST

Dieses Kommando bewirkt die Ausgabe der Programmliste mit Crossreference und Statistik auf einen Drucker oder in eine Textdatei.

Syntax:

[⟨Zlnr1⟩] PLIST ⟨Datei⟩

[⟨Zlnr2⟩] [,⟨Ganzzahl⟩] [REM [⟨LU⟩]]

⟨Zlnr1⟩ : Zeilennummer der ersten Anweisung, die ausgegeben werden soll. Fehlt diese Angabe, beginnt die Ausgabe mit der ersten Anweisung des Programms.

⟨Datei⟩ : Der Name der Textdatei, in die ausgegeben werden soll, oder der Name des Druckers, auf dem das Programm ausgedruckt werden soll.

⟨Zlnr2⟩ : Die Nummer der letzten auszugebenden Anweisung. Fehlt diese Angabe, endet die Ausgabe mit Programmende.

⟨Ganzzahl⟩: Anzahl Zeilen pro Seite. (24-99) Fehlt diese Angabe, wird die Standardhöhe 48 benutzt.

REM [⟨LU⟩]: Die Nummer der logischen Platteneinheit, auf der die Textdatei der Kommentare liegt. Ist ⟨LU⟩ = 0, kann diese Angabe entfallen.

Falls keine REM-Angabe gemacht wurde und das aktuelle BASIC-Programm schon gesichert wurde (d.h. der Programmname existiert schon), wird die LU mit der Nr. 0 nach einer zugehörigen Textdatei mit Kommentaren durchsucht.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

Bei der Ausgabe des Programmtextes werden ausgelagerte Kommentare nach rechts herausgerückt. Sie beginnen standardmäßig auf Spalte 81 und belegen u.U. mehrere Zeilen.

Da diese Ausgabeform manchmal unerwünscht ist, kann sie für Kommentare, die als REM-Anweisung formuliert sind, unterdrückt werden. Der Kommentar wird in der Form REM@[<Zeich>]^n eingegeben. Die Ausgabe beginnt dann auf Spalte 1.

Weiterhin können Zeilen- bzw. Seitenvorschub im Ausdruck durch speziellen REM-Anweisungen erreicht werden:

REM↓	[<Zeich>]^n	Zeilenvorschub und Kommentar
REM↓↓	[<Zeich>]^n	Seitenvorschub und Kommentar

Tritt bei der Ausführung des Kommandos PLIST ein Fehler auf (z.B. Papierende), so wird eine Fehlermeldung ausgegeben. Dann kann zwischen Abbruch und Neustart gewählt werden.

Erläuterung des Ausdrucks:

Auf jeder Seite erscheinen drei Kopfzeilen mit folgenden Angaben:

Zeile 1 : Datum, Programmname, Programmierer und Seitennummer.
Zeile 2 : Branch, Release Level, From Master, To Master (Diese Angaben werden aus dem Dateiheader gelesen.)
Zeile 3 : LINE, BASIC STATEMENT, LINE und COMMENT als Überschriften.

Liste der Anweisungen

Die Anweisungen werden mit den zugehörigen Zeilennummern aufgelistet. Kommentare werden auf der rechten Seite des Ausdrucks getrennt aufgelistet. Die Angabe "X=>" vor einer Zeilennummer zeigt an, daß diese Nummer im Programm referiert ist. "X" ist die Anzahl der Referenzen.

Die BASIC-Kommandos

4

Variablen "CROSS-REFERENCES"

Folgende Angaben werden in einer Tabelle ausgegeben:

- Variablenname (VAR)
- Variablengröße (SIZE) bei Strings : Länge in Bytes
  bei Vektoren : Anz. Elemente
  bei Matrizen : Anz. Reihen,Spalten

  Bei einzelnen Indizes größer 999, bei Dimensionierungen zur Laufzeit oder bei komplexen Ausdrucken wird unter SIZE "******" ausgegeben. Die genaue Dimensionierung ist dem DIM-Statement unter nachfolgend angegebenen LINE (Dimensionierung der Variablen) zu entnehmen.

  Bei der Angabe der Variablengröße werden besonders gekennzeichnet: mehrfach dimensionierte (MULTDIM), implizit dimensionierte (IM DIM) und nicht dimensionierte (= Strings, die in einem Programm, das dieses aufruft, dimensioniert und durch LINK an dieses übergeben wurden) (NO DIM).

- Zeilennummer der Dimensionierung (LINE)
- Variablentyp (TYPE) bei Strings : $
  bei num Vars : 1% - 4%

  Diese Angabe wird statisch über das Quellprogramm ermittelt, so daß Abweichungen zur Interpretationszeit möglich sind (siehe auch "DATA AREA" in der Programm-Statistik).
  Bei der Angabe des Variablentyps sind Globals zusätzlich mit G gekennzeichnet.

- Zeilennummern der Anweisungen, die diese Variable ansprechen. Die Zeilennummern können mit einem Kennzeichen M oder R versehen sein:
  -- M Die Variable steht in dieser Zeile auf der linken Seite einer LET-Anweisung (sie wird modifiziert).
  -- R Die Variable wird in dieser Zeile referiert. Sie steht in dieser Zeile auf der rechten Seite einer LET-Anweisung.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

Unter der Tabelle erscheint folgende Programmstatistik:

```
UNDIMENSIONED VARS : (nicht deklarierte Variablen)
*                                                    *
VARIABLES REFERENCED = xxx OUT OF 340
SIMPLE VARIABLES = xxx  STRING VARIABLES = xxx
DIMENSIONED ARRAYS = xxx  TOTAL REFERENCES = xxx
PROGRAM B-FILE SIZE xxxxx WORDS
PROGRAM D-FILE SIZExxxxx WORDS + DATA AREA xxxxx WORDS
                                  = RUNSIZE xx KB
COMPILED VARIABLES = xxx OUT OF xxx TOTAL USED
*                                                    *
```

D-FILE SIZE und COMPILED VARIABLES werden nur bei BASIC-Programmen, die als D-File "gesaved" wurden, ausgegeben.

Zeilennummern "CROSS-REFERENCES"

In einer Tabelle werden von links nach rechts ausgegeben:

- Referierte Zeilennummern (STMT)
  "U" kennzeichnet die Zeilennummern, die referiert werden, aber im Programm nicht existieren.
- Die Zeilennummern der Anweisungen, in denen die Referenz erfolgt. Alle diese Zeilennummern tragen eines der Kennzeichen S oder T.
  -- S Unterprogrammsprung
  -- T GOTO-Anweisung

Kanalnummern "CROSS-REFERENCES"

In einer Tabelle werden ausgegeben:

- Referierte Kanalnummern (CHNL)
  An der letzten Stelle sind unter "?" die Kanalnummern zusammengefaßt, deren Zuweisung erst zur Laufzeit erfolgt (z.B. A), deren Kanalnummer >99 ist oder die aus mehreren Elementen eines Ausdrucks bestehen.
- Die Zeilennummern der Anweisungen, in denen die Referenz erfolgt.

Die Funktion des Kommandos kann für ein oder mehrere Programme auch über den TAMOS-Selektor angewählt werden (siehe Kap. Programmwartung).

Die BASIC-Kommandos

4

4.16 Das Kommando RENUMBER

Das Kommando RENUMBER erzeugt eine Neunumerierung der Zeilennummern eines BASIC-Programms.

RENUMBER wird im allgemeinen nach umfangreichen Änderungen in BASIC-Programmen durchgeführt, besonders dann, wenn durch Einfügen neuer Anweisungen keine Lücken mehr zwischen den Zeilennummern vorhanden und weitere Einfügungen dadurch nicht möglich sind.

Syntax:

[⟨Zlnr⟩] RENUMBER [⟨Ganzzahl⟩]

⟨Zlnr⟩ : Die Zeilennummer, die die erste Anweisung im Programm erhalten soll.
Ist ⟨Zlnr⟩ nicht angegeben, erhält die erste Anweisung die Zeilennummer ⟨Ganzzahl⟩.

⟨Ganzzahl⟩: Schrittweite.

Ist die Schrittweite nicht angegeben, wird als Standard-Schrittweite 10 eingesetzt.

Entsteht bei Neunumerierung bedingt durch einen Parameter eine Zeilennummer ⟩ 65535, werden die Anweisungen mit einer Schrittweite von 1 numeriert.

- Beispiel:

```
1000 RENUMBER 5
```

Die Zeilennummern beginnen nach der Umnumerierung bei 1000 und haben eine Schrittweite von 5 (also: 1000, 1005, 1010, 1015, 1020 usw.).

Fehler:

Wird in einer Anweisung eine nicht vorhandene Zeilennummer referiert, erfolgt die Meldung:

ERROR # 6 AT ⟨Zlnr⟩

Die referierte nicht vorhandene Zeilennummer wird dann durch die Zeilennummer der folgenden Anweisung ersetzt. Folgt keine Anweisung mehr, wird die Zeilennummer auf 0 gesetzt. RENUMBER wird fortgeführt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

4.17 Das Kommando RUN

Mit RUN kann ein beliebiger Abschnitt des im Hauptspeicher befindlichen BASIC-Programms ausgeführt werden.

Syntax:

[⟨Zlnr1⟩] RUN [⟨Zlnr2⟩ [,⟨Zlnr2⟩][4]]

⟨Zlnr1⟩ : Zeilennummer der Anweisung, ab der die Ausführung gestartet werden soll.

Diese Angabe ist nur zulässig, wenn das Programm bis ⟨Zlnr1⟩ im DEBUG-Modus ausgeführt oder bereits ein "Initialisierendes" RUN-Kommando durchgeführt und das Programm anschließend noch nicht geändert wurde.
Fehlt diese Angabe, wird das Programm mit der ersten Anweisung gestartet ("Initialisierendes" RUN).

⟨Zlnr2⟩ : Zeilennummern, bei deren Erreichen das Programm angehalten wird (Stop-Bedingung) und in den DEBUG-Modus geht.

Ist eine der unter ⟨Zlnr2⟩ angegebenen Zeilennummern erreicht, wird die Meldung:

STOP AT ⟨Zlnr2⟩ ⟨Anweisung⟩

mit der entsprechenden ⟨Zlnr2⟩ angezeigt. Zu dieser Zeit sind alle angegebenen Stop-Bedingungen gelöscht. Diese Anweisung ist noch nicht ausgeführt. Das System steht im DEBUG-Modus.
Ist keine ⟨Zlnr2⟩ angegeben, wird das Programm bis zu seinem logischen Ende ausgeführt.

- Beispiel:

```
RUN 300,400,500
```

Starten eines Programms am Programmanfang. Das Programm wird bei Erreichen von Zeilennummer 300, 400 oder 500 (je nachdem, welche es zuerst erreicht) angehalten und kann im DEBUG-Modus weitergetestet werden.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Fehlermöglichkeiten:

Wird unter <Zlnr1> eine Zeile angegeben, die nicht vorhanden ist, erfolgt die Meldung:

ERROR # 6

und das System bleibt im BASIC-Kommandomodus.

Alle weiteren Fehlermeldungen, die während der Programmausführung möglich sind, werden im Kapitel "Fehlerbehandlung" beschrieben. Mit Ausnahme der nicht abfangbaren Fehler wird das Programm nicht abgebrochen.

4.18 Das Kommando SAVE

Dieses Kommando dient zum Sichern eines BASIC-Programms. Das Programm wird dabei aus dem Hauptspeicher auf Magnetplatte ausgelagert.

Die Angabe ,B, bzw. ,D, legt den Filetyp fest, mit dem das Programm gesichert wird. Fehlt diese Angabe, wird das Programm mit dem Filetyp gesichert, mit dem es in der Partition steht. Ein neu erstelltes oder mit LOAD in eine leere (!) Partition geholtes Programm wird als Typ B in der Partition angelegt.
Ist ein besseres Laufzeitverhalten erforderlich, sollten nur noch Programme vom Typ D verwendet werden!

Syntax:

SAVE [,D,|,B,] [#<PartG>#] [<ProgName> [!]]

,B, : Das Programm wird als B-File gesichert.

,D, : Das Programm wird als D-File gesichert.

<PartG> : Partitiongröße in KB
Fehlt diese Angabe, wird bei schon existierenden Dateien der alte Wert und bei neuen Dateien ein Standardwert übernommen.

<ProgName>: Name der Plattendatei, unter dem das Programm gesichert werden soll. Diese Angabe kann entfallen, wenn das Programm schon einmal auf Magnetplatte ausgelagert worden ist und nach Änderung unter dem selben Namen gesichert werden soll.

Die BASIC-Kommandos

! : Wenn eine bereits bestehende Programmdatei ersetzt werden soll, ist die Angabe "!" erforderlich.

Das SAVE-Kommando trägt die Prüfsumme (HASH) des BASIC-Programms bis Anweisungsnummer 49999 im Header ein. Diese Prüfsumme ist identisch mit der, dir der LIBR-Prozessor über die zugehörige Textdatei nach Aufruf ermittelt und anzeigt.

Die Optimizer Funktion des Kommandos kann für ein oder mehrere Programme auch über den TAMOS-Selektor angewählt werden (siehe Kap. "Programmwartung").

4.19 Das Kommando SIZE

Dieses Kommando dient dazu, die aktuelle Größe des Programms festzustellen.

Syntax:

```
SIZE
```

Die Programmgröße wird in folgender Form ausgegeben:

```
SIZE: xxxx WORDS, TOTAL: yyyy WORDS (zz KB) OUT OF
                                    uuuu WORDS (vv KB)
```

xxxx = Anzahl Worte, die durch die Anweisungen + Datenbereiche (nur bereits dimensionierte Variablen) belegt sind.

yyyy = Gesamtanzahl Worte des Programms einschließlich der Hilfsbereiche (Stacks und Variablenliste).

zz = Minimale Partitiongröße, in der das Programm ablaufen kann.
Diese Angabe ist jedoch nur realistisch, nachdem alle Dimensionierungen des Programms durchlaufen wurden.
Der hier angezeigte Wert kann beim Aufruf des SAVE-Processors angegeben werden.

uuuu, vv = Im SAVE-Kommando angegebene Partitiongröße

Die BASIC-Kommandos

4

4.20 Das Kommando TRACE

Das Kommando TRACE wird zum Testen von BASIC-Programmen benötigt, ergänzend zu den Möglichkeiten, die DEBUG bietet. Die STOP-Funktionen sind nur in Programmen des Typs B wirksam.

Syntax:

$$\left\{\begin{array}{l}[\langle Zlnr1\rangle]\ \text{TRACE}\ [\langle Zlnr2\rangle]\left\{\begin{array}{l},\langle VarName\rangle\end{array}\right\}^{5}\left\{\begin{array}{l},\langle Bedingung\rangle\end{array}\right\}^{2} \\ \text{TRACE}\ [\langle Datei\rangle]\end{array}\right\}$$

wobei:

$$\langle Bedingung\rangle ::= \left\{\begin{array}{l}\langle NVarName1\rangle\langle VglOper\rangle\left\{\begin{array}{l}\langle NVarName2\rangle \\ \langle Dezzahl\rangle\end{array}\right\} \\ \langle NVarName\rangle\{=|\langle\rangle\}\langle Dezzahl1\rangle:\langle Dezzahl2\rangle \\ \langle SVarName1\rangle\langle VglOper\rangle\left\{\begin{array}{l}\langle SVarName2\rangle \\ \langle SLit\rangle\end{array}\right\}\end{array}\right\}$$

Die Länge von ⟨SLit⟩ ist auf 16 Zeichen begrenzt.

Das TRACE-Kommando ermöglicht 3 verschiedene Funktionen:

- Die erste Funktion besteht darin, ein Programm bei Zuweisung an bestimmte Variablen zu unterbrechen. Ihr entspricht die syntaktische Alternative

  $$[\langle Zlnr1\rangle]\ \text{TRACE}\ [\langle Zlnr2\rangle]\{,\langle VarName\rangle\}^{5}$$

  Wird das Programm anschließend mit RUN gestartet (bzw. durch ⟨Zlnr⟩RUN fortgesetzt), so wird es gestoppt, wenn einer der unter ⟨VarName⟩ genannten Variablen zwischen dem ersten Erreichen von ⟨Zlnr1⟩ und dem ersten Erreichen von ⟨Zlnr2⟩ (jeweils einschließlich) ein Wert zugewiesen wird. (Dies geschieht bei den Anweisungen LET, READ, INPUT, CALL, DIM, DEALLO und den entsprechenden MAT...-Varianten), sowie bei den Datenfernübertragungs-Anweisungen INIT, CNTRL, PUT, LON, LOFF bezüglich der Header-Variablen und GET. Beim CALL wirkt der TRACE schon bei der Referenz der angegebenen Variablen.

© Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten.

Die BASIC-Kommandos

Es wird in den DEBUG-Modus geschaltet, sowie die verursachende Anweisung und die betroffenen Variablennamen mit ihren Werten ausgegeben. Diese Anweisung ist dann bereits ausgeführt! Das Programm läßt sich mit ⟨folgende Zlnr⟩ RUN fortsetzen. Das TRACE-Kommando bleibt wirksam, auch über CHAIN und LINK hinaus. Sind ⟨Zlnr1⟩ bzw. ⟨Zlnr2⟩ nicht angegeben, ist der Programmabschnitt, in dem das TRACE-Kommando wirkt, durch die erste bzw. letzte Anweisung des Programms begrenzt.

Ein bestehendes TRACE-Kommando wird durch ein neues TRACE-Kommando oder [CTL Y und] ESC ungültig, außerdem nach Programmende oder Veränderung des Programmtextes.

Bei Zuweisung an indizierte Variablen wird nur die in der Zuweisung angesprochene Komponente (bzw. der angesprochene Teilstring) ausgegeben. Beim Stop nach MAT...-Anweisungen geschieht keine Ausgabe. Durch PRINT oder MAT PRINT lassen sich aber beliebige Komponenten der Matrix anzeigen. Falls eine Variable durch DEALLO aufgelöst wird, wird dies durch ⟨VarName⟩="deallocated" angezeigt.

- Die zweite Funktion des TRACE-Kommandos ist ein Stop bei vorgegebenen Werten bzw. Wertebereichen. Die Funktion wird durch folgende syntaktische Variante ausgelöst:

  [⟨Zlnr1⟩] TRACE [⟨Zlnr2⟩]{,⟨Bedingung⟩}^{2}

  wobei ⟨Bedingung⟩ wie oben definiert ist. Die Programmausführung wird gestoppt, wenn eine der angesprochenen Variablen im angesprochenen Programmteil einen Wert erhält, so daß ⟨Bedingung⟩ erfüllt ist. Die Stop-Bedingung ⟨NVarName⟩=⟨Dezzahl1⟩:⟨Dezzahl2⟩ ist erfüllt, wenn der Wert von ⟨NVarName⟩ bzw. der entsprechenden Komponenten zwischen ⟨Dezzahl1⟩ und ⟨Dezzahl2⟩ (einschließlich) liegt, bzw. ...⟨⟩..., wenn dies nicht der Fall ist.
  Ansonsten gelten die für die erste Funktion genannten Regeln.

4

- Die dritte Funktion dient zur Ausgabe aller im Programm dimensionierten Variablen mit ihren Werten nach dem Programmlauf auf dem Bildschirm, den Drucker oder in eine Textdatei und wird durch

  TRACE [<Datei>]

  ausgelöst. <Datei> muß der Name einer Textdatei oder eines Druckers sein, worauf die Ausgabe erfolgt. Ist <Datei> nicht angegeben, erfolgt die Ausgabe auf dem Bildschirm.

  Die Ausgabe erfolgt 3-stellig für den Variablennamen und 4-, 6-, 10-, oder 14-stellig für numerische Variablen in Exponentdarstellung und beliebig lang bei Stringvariablen. Erst werden alle numerischen Variablen ausgegeben, bevor die Stringvariablen folgen. Ein nicht druckbares Zeichen in einem String wird durch ein Blank ersetzt.

  Das TRACE-Kommando löscht den DEBUG-Modus nicht.

4.21 Das Kommando UNCOMMENT

Mit dem Kommando UNCOMMENT werden die Kommentare aus einem Programm entfernt und in eine Textdatei auslagert. Sie können mit dem Kommando COMMENT wieder eingefügt werden.

Syntax:

```
[<Zlnr1>] UNCOMMENT [<ProgName> [!]]
                                    [<Zlnr2>] [REM [<LU2>]]
```

<Zlnr1> : Zeilennummer der ersten Anweisung, von der mit der Auslagerung von Kommentaren begonnen werden soll. Fehlt diese Angabe, geht die Auslagerung vom Programmanfang aus.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Kommandos

<ProgName>: Mit dem Namen <ProgName>, der auch die Nummer der logischen Platteneinheit beinhaltet, wird das in der Partition stehende Programm um die Kommentare reduziert und gesichert. Der eigentliche Name darf höchstens 12 Zeichen lang sein, da die Kommentare in eine Textdatei des gleichen Namens, aber mit den Anfangsbuchstaben Y. geschrieben werden. Ist <ProgName> nicht angegeben, wird der Name des in der Partition liegenden Programmes genommen.

! : Ein "!" ist erforderlich, falls ein Programm mit dem Namen <ProgName> schon existiert und überschrieben werden soll.

<Zlnr2> : Zeilennummer der letzten Anweisung, bis zu der die Auslagerung von Kommentaren ausgeführt werden soll. Fehlt diese Angabe, geschieht die Auslagerung bis zum Programmende.

REM[<LU2>]: Nummer der logischen Platteneinheit, auf die die Textdatei mit dem Kommentaren geschrieben werden soll. Ist <LU2>=0, kann diese Angabe entfallen.

Es werden sowohl die Kommentare aus REM-Anweisungen, als auch die mit /* an andere Anweisungen angehängten Kommentare ausgelagert.
Ist die Sprungadresse eines GOTO oder GOSUB eine REM-Anweisung, so wird diese Sprungadresse auf die Zeilennummer der nächsten nicht auszulagernden Anweisung geändert.
Eine REM01-Anweisung wird nicht ausgelagert.
Die Funktion des Kommandos kann für ein oder mehrere Programme auch über den TAMOS-Selektor angewählt werden (s. Kap. "Programmwartung").

Datenformate und Datendarstellung

5 Datenformate und Datendarstellung

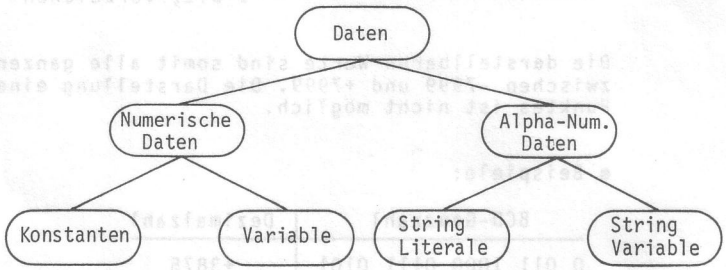
Die Business-BASIC-Sprache unterscheidet numerische Daten und alphanumerische Daten. Alphanumerische Daten werden auch Strings oder Zeichenketten genannt. Diese Unterscheidung ist sinnvoll, da numerische Daten allgemein arithmetischen Verarbeitungen unterliegen, während alphanumerische Daten vor allem zu Anzeige- und Druckzwecken und als Such- und Sortierkriterien verwendet werden.

Die numerischen und die alphanumerischen Daten werden jeweils wieder unterschieden in variable Daten und konstante Daten. Unter dem Begriff Variable werden einzelne oder mehrere Speicherworte im Datenbereich der Anwenderpartition im Hauptspeicher verstanden, in die Daten eingesetzt oder in denen sie verändert werden. Die Variable ist also nicht gleichzusetzen mit ihrem Wert.

Als Konstante werden Daten bezeichnet, die bei der Programmerstellung definiert und während der Programmausführung nicht verändert werden. Eine Konstante hat keinen Namen, sondern wird lediglich durch ihren Wert repräsentiert.

Bezeichnungen:

- (Einfache) Variable = Numerische Variable
- Konstante = Numerische Konstante
- String-Variable = Alphanumerische Variable
- String-Literal = Alphanumerische Konstante



© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

5.1 Darstellung numerischer Daten

Die Darstellung numerischer Daten ist vom Aufbau des Hauptspeichers beeinflußt. Der Speicher des Systems 8870 hat Wort-Struktur.

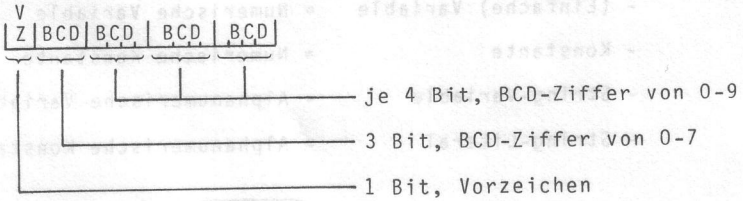
1 Wort = 2 Byte = 16 Bit

5.1.1 Einfache numerische Variablen

Eine numerische Variable belegt im Speicher 1, 2, 3 oder 4 Worte (Format der Variablen).
Die Werte von 1-Wort-Variablen werden als "BCD-Ganzzahl mit Vorzeichen" abgestellt, während die Werte von 2- bis 4-Wort-Variablen als "Gleitkommazahl" dargestellt werden.

"BCD" ist die Abkürzung für Binärdezimalcode und bedeutet, daß jede Dezimalziffer einzeln binär verschlüsselt wird.

5.1.1.1 Aufbau einer BCD-Ganzzahl



Die darstellbaren Werte sind somit alle ganzen Zahlen zwischen -7999 und +7999. Die Darstellung eines Dezimal-Punktes ist nicht möglich.

- Beispiele:

BCD-Ganzzahl	Dezimalzahl
0 011 1000 0111 0101	+3875
1 000 0010 1001 0110	-296

5

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

5.1.1.2 Aufbau von Gleitkommazahlen

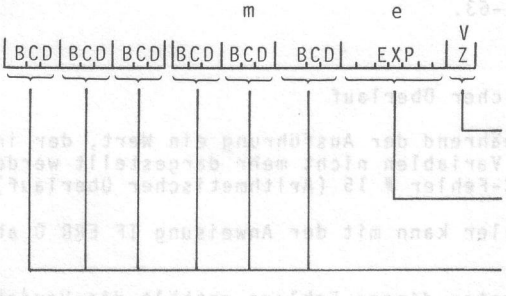
Gleitkomma (floating point) ist eine halblogarithmische Zahlendarstellung. Die Zahlen werden in Form:

$m * 10 \uparrow e$ mit $0.1 \leq ABS(m) < 1$

m = Mantisse
e = Exponent (oder Charakteristik)

dargestellt.

Aufbau einer 2-Wort-Gleitkommazahl:



Der Exponent ist auf folgende Art als Dualzahl verschlüsselt:

```
0000001   =  -63
   :
0111111   =   -1
1000000   =    0
   :
1111111   =   63
```

Der Dezimalpunkt der Mantisse wird immer vor der ersten BCD-Ziffer angenommen.

- Beispiele:

Gleitkommazahl	Dezimalzahl
0111 0010 0101 0100 0110 0001 1101011 0	+0.725461E+43
0110 0101 0000 0000 0000 0000 0011101 1	-0.65E-35

Datenformate und Datendarstellung

Bei 3- und 4-Wort-Variablen werden diesen beiden Worten noch ein bzw. zwei Worte mit je 4 BCD-Ziffern vorangestellt. Damit erhöht sich die Genauigkeit der Mantisse auf 10 bzw. 14 signifikante Ziffern.

Der größte und kleinste darstellbare Wert ist:

bei 2-Wort-Variablen: +/-0.999999E+63
bei 3-Wort-Variablen: +/-0.9999999999E+63
bei 4-Wort-Variablen: +/-0.99999999999999E+63

Der betragsmäßig kleinste von 0 verschiedene darstellbare Wert ist bei 2-, 3- und 4-Wort-Variablen:

+/-0.1E-63.

5.1.1.3 Arithmetischer Überlauf

Entsteht während der Ausführung ein Wert, der in der aufnehmenden Variablen nicht mehr dargestellt werden kann, wird BASIC-Fehler # 15 (Arithmetischer Überlauf) gemeldet.
Dieser Fehler kann mit der Anweisung IF ERR 0 abgefangen werden.

Nach Auftreten dieses Fehlers enthält die Variable den maximal/minimal darstellbaren Wert (siehe oben).

Entsteht in einer 2- bis 4-Wort-Variablen ein Wert, der aus mehr signifikanten Ziffern (Ziffern der Mantisse) besteht, als die Variable aufnehmen kann, werden die überlaufenden Ziffern ohne Fehlermeldung rechts abgeschnitten.

Datenformate und Datendarstellung

5.1.2 Vektoren/Matrizen

Vektoren:

Vektoren sind lineare Datenfelder, die aus 2 bis n Elementen bestehen. Ein Element eines Vektors entspricht im Aufbau einer numerischen Variablen und kann 1 bis 4 Worte groß sein. Alle Elemente eines Vektors haben das gleiche Format.

Matrizen:

Matrizen sind zweidimensionale Datenfelder bestehend aus:

2 bis n Reihen und
2 bis m Spalten.

Ein Element einer Matrix entspricht im Aufbau einer numerischen Variablen und kann 1 bis 4 Worte groß sein. Alle Elemente einer Matrix haben das gleiche Format.

Anmerkung:

Soweit in der Syntax der Begriff ⟨NVar⟩ benutzt wird, bezeichnet er:

- eine einfache numerische Variable oder
- ein Vektorelement oder
- ein Matrixelement.

Matrix-Variable ⟨MVar⟩ dagegen bezeichnen Matrizen in ihrer Gesamtheit.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

5.1.3 Numerische Konstanten

Eine numerische Konstante ist ein beliebiger Wert im Bereich von

0 bis +/-0.99999999999999E+63 in Gleitkommadarstellung
oder
maximal 63 Ziffern in dezimaler Darstellung.

Ist kein Vorzeichen angegeben, ist die Konstante positiv.

Eine numerische Konstante belegt grundsätzlich 4 Speicherworte, d.h. sie wird mit größtmöglicher Genauigkeit dargestellt.

- Beispiele:

  - Vergleich, ob der Inhalt der numerischen Variablen X im Bereich von 1 bis 10 liegt.

    ```
    90 IF X>=1 IF X<=10 GOTO 210
    ```

  - Durchführen einer arithmetischen Operation mit einer numerischen Konstanten.

    ```
    50 LET U=D*3.14159
    ```

5

5.2 Darstellung alphanumerischer Daten

Die Darstellung von alphanumerischen Daten erfolgt in einem modifizierten ASCII-Code. Jedes Zeichen belegt

1 Halbwort = 1 Byte = 8 Bit.

Da der ASCII-Code nur aus 7 Bit besteht, wird bei der Darstellung eines Zeichens das linke Bit grundsätzlich auf 1 gesetzt.

- Beispiel:

  Alphanumerische Darstellung der Zeichenkette "AB".

A	B
11000001	11000010

Eine Liste sämtlicher ASCII-Zeichen ist in "Tastatur-Bildschirm-Ein-/Ausgabe" zu finden.

5.2.1 String-Variablen

Im Gegensatz zu numerischen Variablen bestehen für eine String-Variable keine Vorschriften über deren Größe. Sie kann vom Programmierer frei definiert werden und kann 1 bis n Zeichen betragen. Die maximale Länge eines Strings ist nur durch die Größe der Partition eingeschränkt.

In String-Variablen werden Daten immer linksbündig abgestellt.

Andere Formen der Darstellung von Zeichen in Strings, wie sie durch verschiedene CALL-Anweisungen ermöglicht werden, sind im Kapitel "CALL-Anweisungen" behandelt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

5.2.2 String-Literale

Eine alphanumerische Konstante (String-Literal) ist eine Folge beliebiger Zeichen, die durch "" begrenzt sind. Neben der Angabe des Zeichens selbst ist es auch möglich, den Oktal-Code von Zeichen in "←" eingeschlossen anzugeben (siehe Codetabelle).

- Beispiele:

  - Eingabe mit Prüfung, ob Y oder N eingegeben wurde.

```
:
20 INPUT TAB(X,Y),"DRUCKEN Y/N ",A$
30 IF A$="Y" GOTO 100
40 IF A$="N" GOTO 660
```

  - Abstellen der alphanumerischen Konstanten "ABCD" und dem Funktionscode für den Signalgeber in einem String.

```
:
50 LET X$="ABCD←207←"
```

    Nach der Durchführung der LET-Anweisung steht ein Grenzzeichen in X$ auf Position 6.

Anmerkung:

Soll eines der beiden Zeichen " oder← in einem String-Literal angegeben werden, muß dies in Oktal-Code geschehen. Zusätzlich besteht die Möglichkeit, das Zeichen " durch zwei Hochkommata ' zu erzeugen.

Datenformate und Datendarstellung

5

5.3 Deklaration (Dimensionierung) von Variablen

Die Reservierung von Speicherplatz (Dimensionierung) für eine Variable erfolgt bei B-Files zur Programmausführungszeit, bei D-Files auch mit Programmsicherung (SAVE). Dabei wird unterschieden zwischen

- expliziter Dimensionierung mit einer DIM- oder GLOBAL-Anweisung
- impliziter Dimensionierung beim ersten Ansprechen der Variablen

Explizit dimensioniert werden müssen:

- der Name und die Länge von String-Variablen

Eine implizite oder explizite Dimensionierung ist möglich bei:

- den Namen numerischer Variablen
- der Genauigkeit einfacher numerischer Variablen
- der Genauigkeit von Vektorelementen
- der Genauigkeit von Matrixelementen
- der Anzahl Elemente eines Vektors
- der Anzahl Reihen und Spalten einer Matrix

Grundsätzlich sollte jedoch jede verwendete Variable (aus Optimierungs- und Dokumentationsgründen) explizit dimensioniert werden, und dabei zwischen lokalen Variablen (DIM) und globalen Variablen (GLOBAL) unterschieden werden.

Es können maximal 340 Variablen dimensioniert werden.

> Ein gültiger Variablenname besteht aus einem Buchstaben oder einem Buchstaben und einer Ziffer.
> Beispiele: A, F9, XO, Z.
> Ein String-Variablenname ist ein gültiger Variablenname, an den ein "$" angehängt ist.
> Beispile: A$, B5$, F7$, X$.

Nach durchgeführter Dimensionierung enthalten numerische Variablen und die Elemente von Vektoren und Matrizen den Wert 0. Strings sind mit Grenzzeichen gefüllt (Code: alle acht Bits enthalten 0).

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

GLOBAL-Anweisungen werden verwendet, um die bei einem Programmaufruf mit LINK zu übergebenden Variablen zu deklarieren. Enthält ein Programm eine GLOBAL-Anweisung, werden nur die in GLOBAL-Anweisungen genannten Variablen übergeben.

Strings und einfache numerische Variable dürfen in einem Programm nur redimensioniert werden, wenn sie vorher in einer DEALLO-Anweisung aufgetreten sind!
Variablen, die mit einer GLOBAL-Anweisung deklariert wurden, dürfen allerdings nicht in einer DEALLO-Anweisung auftreten.

Dagegen dürfen lokale Matrizen und Vektoren ohne DEALLO-Anweisung redimensioniert werden, wenn sich die Anzahl ihrer Elemente nicht vergrößert. Das Format der Elemente kann allerdings nicht verändert werden (eine Formatangabe wird ignoriert); die Elemente werden auch nicht mit 0 initialisiert. Eine Vergrößerung einer Matrix auf die alte Elementezahl ist nach einer Verkleinerung durch Redimensionierung ebenfalls nicht möglich.

Die Verwendung der DEALLO-Anweisung sowie andere Redimensionierungen verhindern eine Erzeugung von Variablenadressen. Sie sollten nur bei laufzeitunkritischen Programmen verwendet werden.

5.3.1 Aufbau der DIM- und der GLOBAL-Anweisung

DIM <Feldbeschr> [,<Feldbeschr>]^{n}

wobei:

<Feldbeschr>::=
- <NVarName>[(<NAusdr>[,<NAusdr>])]
- <SVarName> (<NAusdr>)
- <Format>

GLOBAL <Feldbeschr> [,<Feldbeschr>]^{n}

wobei:

<Feldbeschr>::=
- <NVarName>[(<Ganzzahl>[,<Ganzzahl>])]
- <SVarName> (<Ganzzahl>)
- <Format>

Datenformate und Datendarstellung

5.3.2 Dimensionierung einfacher numerischer Variablen

Zur Dimensionierung von numerischen Variablen stehen zwei Möglichkeiten zur Verfügung:

- Explizite Dimensionierung:
  Dimensionierung von Variablen mit der DIM- oder GLOBAL-Anweisung.

- Implizite Dimensionierung:
  Dimensionierung beim ersten Auftreten von Variablennamen in Anweisungen wie LET, FOR, IF, INPUT usw.

Die Größe der numerischen Variablen wird gesteuert durch einen Format-Schalter mit den folgenden vier Positionen:

1% = 1 Wort-Integer
2% = 2 Wort-Gleitkommazahl
3% = 3 Wort-Gleitkommazahl
4% = 4 Wort-Gleitkommazahl

Entsprechend dem Zustand dieses Schalters zum Zeitpunkt der Dimensionierung wird für eine numerische Variable der Platz im Speicher reserviert.

Der Standardzustand dieses Schalters ist "2%". Eine Veränderung ist nur mit einer DIM- oder GLOBAL-Anweisung möglich. Wird in einer solchen Anweisung eine Formatangabe (1% bis 4%) gemacht, nimmt der Schalter den entsprechenden Zustand an. Dieser bleibt solange für folgende Dimensionierungen gültig, bis er durch eine neue Formatangabe geändert wird.

Anmerkung:

Die Schalterstellung kann durch dynamischen Programmablauf beeinflußt werden!

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

- Beispiel für explizite Dimensionierung:

```
10 GLOBAL A,1%,B
20 DIM    C,2%,D
30 DIM    3%,E,F
```

Es werden die Variablen A und D als 2-Wort-, B und C als 1-Wort-, E und F als 3-Wort-Variablen dimensioniert.

Die letzte Formatangabe (3%) bleibt solange gültig, bis sie durch eine weitere DIM-Anweisung geändert wird.

- Beispiele für implizite Dimensionierung:

```
- :
  20 INPUT A
```

A wird als 2-Wort-Variable dimensioniert, da noch keine DIM- oder GLOBAL-Anweisung aufgetreten ist und der Formatschalter auf dem Standardzustand 2% steht.

```
- 10 DIM 1%,A,B,C,3%
  :
  50 LET X = A*3.14159
  :
  70 DIM 2%
  :
  90 INPUT D
```

Die DIM-Anweisung reserviert Platz für die 1-Wort-Variablen A, B und C. Die Variable X, die nicht in der DIM-Anweisung dimensioniert wurde, wird implizit als 3-Wort-Variable dimensioniert, da die Formatangabe 3% aktuell ist. Vor der Dimensionierung von D wird der Formatschalter auf 2% gesetzt, so daß D als 2-Wort-Variable dimensioniert wird.

5

5.3.3 Dimensionierung von Vektoren/Matrizen

Bei der Dimensionierung von Vektoren/Matrizen ist zusätzlich zur Formatangabe die Anzahl der Elemente bzw. der Reihen und Spalten festzulegen. Beides kann sowohl explizit wie implizit dimensioniert werden. Für das Format der Elemente von Vektoren und Matrizen gelten die gleichen Regeln wie für einfache numerische Variablen.

Explizite Dimensierung bei Vektoren:

Die Anzahl der Elemente, aus denen der Vektor bestehen soll, wird durch die Angabe der Nummer des letzten Vektorelementes in Klammern hinter dem Variablennamen festgelegt.

- Beispiel: 10 DIM A(24)

Die Elemente eines Vektors werden von 0 bis zu dieser Angabe durchnumeriert. Das bedeutet, daß die Anzahl der Elemente eines Vektors der Nummer des letzten Elements + 1 entspricht (im Beispiel: 25 Elemente).

Die Genauigkeit der Elemente wird durch die aktuelle Format-Schalterstellung festgelegt, die mit der gleichen Anweisung geändert werden kann.

- Beispiel: 10 DIM 4%,A(24),2%

Explizite Dimensionierung von Matrizen:

Die Anzahl der Reihen und der Spalten einer Matrix wird durch Anhängen der Nummern der letzten Reihe und der letzten Spalte an den Variablennamen erreicht.

- Beispiel: 10 DIM B(24,79)

Die Numerierung der Reihen und Spalten beginnt ebenfalls bei 0. Das bedeutet, daß die Anzahl Spalten/Reihen der angegebenen Nummer + 1 entsprechen.
Die Anzahl der Elemente einer Matrix errechnet sich aus Anzahl Reihen x Anzahl Spalten (im Beispiel: 2000 Elemente).

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

Achtung:

Sämtliche Matrix-Anweisungen und -Funktionen behandeln Matrizen nur ab Spalte 1 und Zeile 1. Die Elemente in Reihe 0 und Spalte 0 müssen, falls erforderlich, direkt angesprochen werden und können z.B. für Spaltensummen u.a.benutzt werden.

Implizite Dimensionierung von Vektoren/Matrizen:

Vektoren und Matrizen, die in einer beliebigen Anweisung angesprochen werden und noch nicht mit einer DIM-Anweisung dimensioniert wurden, werden von RUN bei ihrem ersten Auftreten dimensioniert:

- ein implizit dimensionierter Vektor besteht aus 11 Elementen (0 bis 10).

- Eine implizit dimensionierte Matrix besteht aus 11 Reihen und 11 Spalten (jeweils 0 bis 10).

Die Genauigkeit der Elemente wird durch die aktuelle Format-Schalterstellung festgelegt.

Es ist klar, daß bei impliziter Dimensionierung kein Element bzw. keine Reihe und Spalte > 10 angesprochen werden darf.

Achtung:

Wird bei impliziter Dimensionierung bei Vektoren das Element 0 oder bei Matrizen das Element in Reihe 0 und Spalte 0 zuerst angesprochen, wird nur eine einfache Variable dimensioniert.

5

- Beispiele für Vektor-/Matrix-Dimensionierung:

  - Dimensionierung eines Vektors mit 10 Elementen (0 bis 9) mit dem Namen V1. Jedes Element soll 2 Worte groß sein. Der Format-Schalter wurde noch nicht verändert.

    ```
    10 DIM V1(9)
    ```

  - Dimensionierung einer Matrix mit 6 Reihen und 8 Spalten mit dem Namen M1. Jedes Elememt soll 4 Worte groß sein.

    ```
    10 DIM 4%,M1(5,7)
    ```

    Die Formatangabe 4% gilt für alle Elemente der Matrix M1 und für alle folgenden Dimensionierungen bis zum Auftreten einer neuen Formatangabe in einer DIM-Anweisung.

  - Die Anweisungsfolge

    ```
    10 DIM 1%,A,B,C,3%,X
    :
    50 LET X1(8) = X
    ```

    dimensioniert den Vektor X1 mit 11 Elementen (0 bis 10). Alle Elemente haben 3-Wort-Format.

  - Die Anweisungsfolge

    ```
    10 DIM A,B,1%,C
    :
    50 LET M1(3,8)=C
    ```

    dimensioniert die Matrix M1 mit 11 Reihen und 11 Spalten. Alle Elemente haben 1-Wort-Format.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

5.3.4 Dimensionierung von String-Variablen

Die Dimensionierung von String-Variablen ist nur mit einer DIM- oder GLOBAL-Anweisung möglich.

Durch die Dimensionierung wird der Name des Strings und die Anzahl Zeichen (Byte) festgelegt, die der String maximal aufnehmen kann.

Diese Größe wird in der dimensionierenden Anweisung in Klammern an den Stringvariablen-Namen angehängt.

- Beispiel:

```
10 DIM A$(100),B$(30)
```

Dimensionierung der Stringvaraiblen

- A$ mit einer Länge von 100 Byte und
- B$ mit einer Länge von 30 Byte

Durch String-Dimensionierungen wird der für die Dimensionierung von numerischen Variablen vorhandene Format-Schalter nicht verändert.

5

5.4 Adressierung von Variablen

5.4.1 Einfache numerische Variablen

Die Adressierung von einfachen numerischen Variablen erfolgt durch die Angabe des Variablennamens.

- Beispiel: 50 LET A=B+C

5.4.2 Vektoren/Matrizen

Zur Adressierung der einzelnen Elemente eines Vektors ist der Name des Vektors und die Nummer des auszuwählenden Elements (0 bis n) anzugeben.
Zur Adressierung der einzelnen Elemente einer Matrix ist die Nummer der Reihe (0 bis n) und die Nummer der Spalte (0 bis n) des auszuwählenden Elements anzugeben.

Ist ein Vektor oder eine Matrix ohne Angabe von Indizes adressiert, wird

- bei Vektoren das Element 0 und
- bei Matrizen das Element 0,0 angesprochen.
  Ist eine Matrix mit nur einem Index adressiert, wird als 2. Index 0 angenommen.

Von Matrix-Anweisungen (MAT) werden Matrizen immer als ganze Einheit, allerdings ohne Reihe 0 und Spalte 0, angesprochen. In diesem Fall wird der Name ohne Indizes angegeben (<MVar>).

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

- Beispiel:

  Adressierung des Elements Reihe 2 und Spalte 4 einer Matrix mit 4 Reihen und 5 Spalten mit einer LET-Anweisung:

  50 LET X(2,4)= ...

Reihe 0 bis 3	Spalte 0 bis 4				
	0	1	2	3	4
0	0,0	0,1	0,2	0,3	0,4
1	1,0	1,1	1,2	1,3	1,4
2	2,0	2,1	2,2	2,3	(2,4)
3	3,0	3,1	3,2	3,3	3,4

5.4.3 String-Variablen

Bei der Adressierung von Strings sind folgende Regeln zu beachten:

- Daten in Strings werden immer linksbündig abgestellt und von links nach rechts bearbeitet.
- Die Bytes eines Strings werden beginnend mit 1 von links nach rechts durchnumeriert.
- Jede Information, die in einen String geschrieben wird, schließt mit einem Grenzzeichen ab.

Grenzzeichenverarbeitung

Für einen String wird bei der Dimensionierung festgelegt, wieviele Zeichen er maximal aufnehmen kann. Intern wird noch ein zusätzliches Byte für das Grenzzeichen reserviert.
Da ein String nicht in seiner ganzen dimensionierten Länge mit Daten belegt sein muß, schreibt RUN folgendermaßen Grenzzeichen in den String:

- bei der Dimensionierung in jedes Byte
- nach einer INPUT-Anweisung hinter das letzte eingegebene Zeichen.
- nach einer LET-Anweisung hinter das letzte übertragene Zeichen (Ausnahme: siehe unten).

Datenformate und Datendarstellung

Ist die String-Variable in der gesamten dimensionierten Länge mit Daten belegt, steht das Grenzzeichen außerhalb des vom Programm adressierbaren Bereichs.

Adressieren von Strings

Für die Adressierung von Strings bestehen die folgenden Möglichkeiten:

- Adressieren eines Strings von Byte 1 bis zum ersten Grenzzeichen durch Angabe des Stringnamens.
  Beispiel: 90 PRINT A$

- Adressierung eines Strings ab einer in Klammern anzugebenden Byte-Adresse bis zum nächsten Grenzzeichen.
  Beispiel: 50 IF A$(19)="END" GOTO 690

- Adressierung eines Teilstrings innerhalb anzugebender Byte-Adressen.
  Beispiel: 60 LET A$(3,7) = X$

Der Index 0 bei Strings wird wie folgt interpretiert: A$(0) als A$, A$(0,n) als A$(1,n), A$(n,0) als A$(n). Außerdem ist A$(1) identisch mit A$.

Die Bearbeitung von Strings wird beendet durch:

- Erreichen eines Grenzzeichens im Quell-String
- String-Ende im Ziel-String
- Erreichen der angegebenen Byte-Endeadresse im Quell- oder Ziel-String

Nach einer Übertragung wird grundsätzlich hinter dem letzten übertragenen Zeichen im Ziel-String ein Grenzzeichen abgestellt.

Ausnahme:

Bei Arbeiten mit Anfang- und Ende-Indizes im Ziel-String wird, falls kein vorhandenes Grenzzeichen überschrieben wird, kein Grenzzeichen abgestellt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Datenformate und Datendarstellung

Wenn bei einer LET- oder INPUT-Anweisung

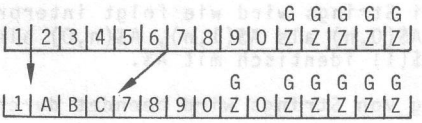
- der Ziel-String mit Anfang- und Ende-Indizes adressiert wird,
- die Anzahl zu übertragender Zeichen ist kleiner als der aufnehmende Bereich im Ziel-String und
- kein Grenzzeichen überschrieben wird,

werden die Zeichen rechts vom adressierten Ende-Byte bis zum ersten folgenden Grenzzeichen bündig an die übertragenen Zeichen herangeschoben.

- Beispiel:

```
20 DIM A$(15)
30 LET A$="1234567890"
 :
90 LET A$(2,6)="ABC"
```

A$ vor und nach Ausführung von Anweisung 90:



5.5 Hexadezimale Konstanten

Diese Konstante ist in einer BASIC-Anweisung überall dort erlaubt, wo ein String-Literal zugelassen ist. (Z.B. in der LET- oder INPUT-Anweisung)

Sie hat die Form

'H {xx}^{n}'

x steht stellvertretend für eine Ziffer (0-9) oder einen Buchstaben (A-F).

- Beispiel: LET A$='H7B3F'

Anmerkung:

Der Wert 00 (hex.) stellt ein Grenzzeichen dar und wird vom System nicht als hexadezimale Konstante akzeptiert. Mit Hilfe der CLEAR-Anweisung läßt sich eine Stringvariable mit Grenzzeichen füllen.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6 Die BASIC-Anweisungen

Die Anweisungen bilden grundlegende Einheiten einer Programmiersprache, vergleichbar mit den Sätzen einer natürlichen Sprache.
Die Anweisungen der Programmiersprache BASIC wurden in ihrem syntaktischen Aufbau in "Kurzliste der Anweisungen" vorgestellt, eine Erläuterung des Aufbaus erfolgt in diesem Kapitel.

Ein Programm besteht aus einer Folge von Anweisungen, die gemeinsam ausgeführt werden sollen. Jede Anweisung, die in das Programm eingebunden werden soll, benötigt eine Zeilennummer. Die Eingabe aus dem BASIC-Kommandomodus erfolgt also in der Form:

⟨Zlnr⟩ ⟨Anweisung⟩

wobei die Zeilennummer zwischen 1 und 65535 liegen muß.

Jede Eingabe einer einzelnen Anweisung wird durch die Betätigung der "CR"-Taste abgeschlossen.

Die Anweisung wird zunächst einer syntaktischen Prüfung unterzogen. Im Fehlerfall erscheint am Bildschirm die Meldung

ERROR # xx ,

die mit HELP entschlüsselt werden kann. Der Cursor steht in der fehlerhaften Zeile an der Stelle, an der der Fehler erkannt wurde.

Falls die Anweisung syntaktisch richtig ist, wird sie gemäß ihrer Zeilennummer in die bisher vorhandenen Anweisungen eingeordnet, und die Eingabe weiterer Anweisungen ist möglich. Es ist also nicht nötig, die Anweisungen in der Reihenfolge einzugeben, in der sie endgültig angeordnet sein sollen. Sie werden automatisch aufsteigend nach Zeilennummern sortiert. Empfehlenswert ist es, einen gewissen Abstand (etwa 10) bei der Vergabe der Zeilennummern einzuhalten, um später mühelos Anweisungen einfügen zu können. Beispiele sind im Kapitel "Programmierbeispiele" zu finden.

Soll eine Anweisung korrigiert werden, die bereits übernommen ist, muß sie einschließlich derselben Zeilennummer neu eingegeben oder mit dem EDIT-Kommando bearbeitet werden.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Soll eine übernommene Anweisung wieder aus dem Programm gelöscht werden, so ist nur die Zeilennummer der Anweisung einzugeben und danach auszulösen (vgl. DELETE-Kommando).

Zur Unterstützung des Programmtests per Dialog können einzelne BASIC-Anweisungen auch direkt nach der Eingabe ausgeführt werden. Die Direktausführung wird dadurch erzeugt, daß die Anweisung ohne Zeilennummer eingegeben wird und somit die Anweisung auch nicht in das Programm eingeordnet werden kann. Durch Direktausführung einer Anweisung wird das zur Zeit im Hauptspeicher stehende Programm nicht verändert.

Zur Beschreibung sind die Anweisungen in 5 Gruppen gegliedert.

- Deklaration von Datenfeldern und Funktionen
- Datenmanipulation und Arithmetik
- Steuerung des Programmablaufes
- Ein-/Ausgabeanweisungen
- Spezielle Anweisungen

6.1 Deklaration von Datenfeldern und Funktionen

Mit den Anweisungen DIM und GLOBAL wird Speicherplatz für Variablen (einschl. Vektoren, Matrizen und Stringvariablen) reserviert, mit DEALLO wieder freigegeben. Mit der DEF-Anweisung werden Anwenderfunktionen definiert. Anwenderfunktionen gelten lokal für das Programm, in dem sie definiert sind.

Besondere Beachtung verdient die Unterscheidung von lokalen und globalen Variablen.

Lokale Variablen eines Programms sind solche, die nur für dieses Programm Bedeutung haben und nicht direkt an andere Programme (über LINK) übergeben oder von anderen Programmen übernommen werden. Damit sind insbesondere alle Variablen in Programmen, die weder aufrufen noch aufgerufen werden, lokal.

Die BASIC-Anweisungen

Lokale Variablen werden mit der DIM-, globale Variablen mit der GLOBAL-Anweisung deklariert. Die GLOBAL-Anweisungen müssen als erste Anweisungen (Ausnahme REM-Anweisungen) und die DIM-Anweisungen hinter den GLOBAL-Anweisungen geschrieben werden. Dadurch ist gewährleistet, daß möglichst tief compiliert werden kann und ein gutes Laufzeitverhalten erreicht wird.

Die Begriffe "lokale" und "globale Variablen" beziehen sich hier nur auf den Fall des Programmaufrufs und der Variablenübergabe durch die LINK-Anweisung (nicht auf die Übergabe mit Hilfe einer Common Area oder von Dateien).

Globale Variablen sind solche, die direkt mit Hilfe der LINK-Anweisung von einem aufrufenden Programm an ein aufgerufenes übergeben werden. (Mit der CHAIN-Anweisung können keine Variablen übergeben werden.)

Es gibt 2 Möglichkeiten, globale Variable für Programme zu definieren:

1 Alle explizit (mit DIM) oder implizit dimensionierten Variablen, die nicht mit DEALLO freigegeben werden, sind global, falls keine GLOBAL-Anweisung vorhanden ist.

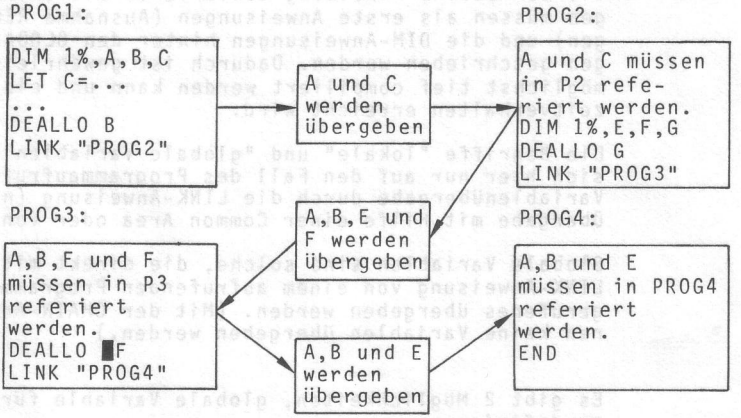
2 Alle in den GLOBAL-Anweisungen enthaltenen Variablen sind global.

Beide Verfahren sind miteinander unverträglich! Der Anwender muß innerhalb seines Programmpaketes (Programme, die mit LINK verkettet sind) entweder nach Möglichkeit 1 oder nach Möglichkeit 2 programmieren.
Die Möglichkeit 2 ist dabei zu bevorzugen, da hierbei tiefer compiliert werden kann und die Programme eine bessere Ablaufzeit haben.
Beide Verfahren arbeiten für B- oder D-Files!

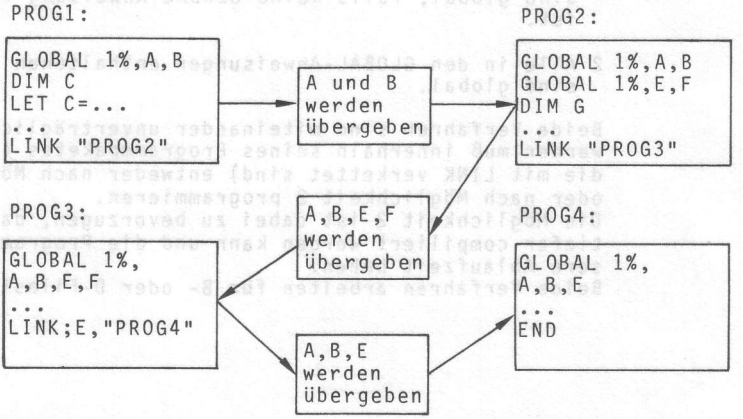
© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

- Beispiel für Möglichkeit 1:



- Beispiel für Möglichkeit 2:



Die Anweisung LINK;E,"..." in PROG3 bewirkt, daß die Werte der globalen Variablen A bis einschließlich E dem Folgeprogramm übergeben werden.

Die BASIC-Anweisungen

6.1.1 Die DIM-Anweisung

Die Anweisung reserviert Speicherbereich für:

- Numerische Variablen
- Vektoren/Matrizen
- Strings

Mit der DIM-Anweisung wird für jede Variable ein Name festgelegt, unter dem sie während des Programmablaufes angesprochen werden kann.

Zu beachten sind die im vorigen Kapitel beschriebenen Definitionen bezüglich lokaler und globaler Variablen.

Syntax:

DIM ⟨Feldbeschr⟩ [,⟨Feldbeschr⟩]^n

wobei:

$$\langle\text{Feldbeschr}\rangle ::= \begin{cases} \langle\text{NVarName}\rangle\ [(\langle\text{NAusdr}\rangle\ [,\langle\text{NAusdr}\rangle])] \\ \langle\text{SVarName}\rangle\ (\langle\text{NAusdr}\rangle) \\ \langle\text{Format}\rangle \end{cases}$$

6.1.2 Die DEALLO-Anweisung

Mit der DEALLO-Anweisung können beliebige Variablen, die implizit oder mit einer DIM-Anweisung deklariert wurden, freigegeben werden. (Name und Speicherplatz können neu belegt werden.) Die Variablen können anschließend erneut dimensioniert werden. In GLOBAL-Anweisungen deklarierte Variablen dürfen nicht in DEALLO-Anweisungen auftreten.

Die zweite Funktion der DEALLO-Anweisung ist die Freigabe lokaler Variablen vor Programmaufrufen mit LINK. Globale Variablen sind nicht freizugeben. Diese Programme enthalten keine GLOBAL-Anweisungen. Übergebene Variablen müssen im aufgerufenen Programm wenigstens einmal referiert werden. Dieses Verfahren wird nicht mehr empfohlen, da es laufzeitungünstig ist. Besser ist die Deklaration von globalen Variablen mit der GLOBAL-Anweisung, so daß die DEALLO-Anweisung entfallen kann. (Vergl. das vorherige Kapitel "Deklaration von Datenfeldern und Funktionen".)

Syntax:

DEALLO ⟨VarName⟩ [,⟨VarName⟩]^n

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Die Ausführungszeit der Anweisung ist vom Standort der betreffenden Variablen in der Variablen-Liste abhängig, da durch Freigeben entstehende Lücken wieder geschlossen werden müssen. Um die Ausführungszeit dieser Anweisung zu verringern, sollten die Variablennamen in der DEALLO-Anweisung in umgekehrter Reihenfolge zur Dimensionierung angegeben werden.

- Beispiel:

```
100 DIM A$(100),1%,A,B,B$(200)
:
190 LET X=A*B+3
:
500 DEALLO X,B$,B,A,A$
```

Bemerkung:

Sofern Fehler bei der Ausführung der Anweisung auftreten, wird das Freigeben bei der verursachenden Variablen abgebrochen.

6.1.3 Die GLOBAL-Anweisung

Mit der GLOBAL-Anweisung werden die globalen Variablen eines Programms deklariert und dimensioniert. Ein Programm, das GLOBAL-Anweisungen enthält, übergibt bei einem Programmaufruf mit LINK nur die in den GLOBAL-Anweisungen genannten Variablen, in der Reihenfolge der Aufschreibung. Die Menge der übergebenen Variablen kann durch die LINK-Anweisung auf einen Anfangsbereich beschränkt werden.

Das aufgerufene Programm muß die übergebenen Variablen in der vom aufrufenden Programm vorgegebenen Reihenfolge und Größe in GLOBAL-Anweisungen deklarieren. Um die Menge der globalen Variablen zu erweitern, muß das aufgerufene Programm diese in weiteren GLOBAL-Anweisungen deklarieren.

Die BASIC-Anweisungen

In GLOBAL-Anweisungen deklarierte Variablen können nicht mit DEALLO-Anweisungen freigegeben werden.
Vor und zwischen GLOBAL-Anweisungen dürfen höchstens REM-Anweisungen stehen.
Mit GLOBAL-Anweisungen sind keine variablen Dimensionierungen möglich.

Syntax:

GLOBAL ⟨Feldbeschr⟩ [,⟨Feldbeschr⟩]ⁿ

wobei:

⟨Feldbeschr⟩::={⟨NVarName⟩ [(⟨Ganzzahl⟩ [,⟨Ganzzahl⟩])]
⟨SVarName⟩ (⟨Ganzzahl⟩)
⟨Format⟩}

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.1.4 Die DEF-Anweisung

Mit der DEF-Anweisung werden Anwenderfunktionen definiert. Mit Anwenderfunktionen kann Schreibarbeit und Speicherplatz gespart werden, wenn gleiche numerische Ausdrücke (Formeln) mehrmals im Programm auftreten sollen.

Syntax:

DEF FN ⟨Bu1⟩(⟨Bu2⟩)=⟨NAusdr⟩

Auf das Schlüsselwort DEF müssen FN und ein Buchstabe folgen, danach ein in Klammern eingeschlossener Buchstabe, ein Gleichheitszeichen und ein numerischer Ausdruck. Der Buchstabe, der auf FN folgt, kennzeichnet die Funktion.
Der in Klammern eingeschlossene Buchstabe ⟨Bu2⟩ ist das formale Argument der Funktion, das dort, wo die Funktion aufgerufen wird, durch das aktuelle Argument ersetzt wird. Es ist nur ein Argument zugelassen.

- Beispiel:

```
:
30 DEF FNS(X)=2*X+1
:
180 LET A=FNS(5)
```

Hier ist X das formale und 5 das aktuelle Argument. A erhält den Wert 11.

Die BASIC-Anweisungen

Das formale Argument darf nur aus einem Buchstaben bestehen, dieser Buchstabe darf als Variablenname bereits vergeben sein.

Grundsätzlich muß die DEF-Anweisung vor dem ersten Aufruf der Funktion einmal durchlaufen worden sein.

Anwenderfunktionen können geschachtelt werden, d.h. im numerischen Ausdruck der DEF-Anweisung dürfen andere Anwenderfunktionen aufgerufen werden. Die Schachtelungstiefe ist auf 8 begrenzt.

- Beispiel: 30 DEF FNX(X)=FNY(X+3)*FNZ(8.5)

Das Umdefinieren einer Anwenderfunktion ist möglich.

Die BASIC-Anweisungen

6.2 Datenmanipulation und Arithmetik

Es folgt die Beschreibung der Anweisungen, die zur Durchführung von Datenmanipulationen und arithmetischen Operationen dienen.

Im einzelnen sind dies:

- LET
- DATA
- READ
- MAT
- RANDOM
- RESTOR
- MAT READ

6.2.1 Die LET-Anweisung

Die LET-Anweisung weist der links vom Zeichen "=" stehenden Variablen den Wert des rechts stehenden Ausdrucks zu.

Das Zeichen "=" ist deshalb nicht als Gleichheitszeichen im mathematischen Sinne zu interpretiert, sondern als:

....."erhält den Wert von"...

Wenn Typ und Format der aufnehmenden Variablen und des zu berechnenden Ausdrucks nicht übereinstimmen, werden die Daten in das Format der aufnehmenden Variablen umgewandelt.

Syntax:

```
[LET] { <NVar>= { <NAusdr> }                                  }
      { <SVar>= { <String> }                                  }
              { <SAusdr>                                      }
              { <NAusdr> [USING<String>]                      }
```

Das Schlüsselwort LET kann weggelassen werden, außer wenn die LET-Anweisung in einer IF-Anweisung auftritt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Numerische Variable als Zielvariable

Mit numerischer Variablen ist eine einfache numerische Variable, ein Vektor-Element oder ein Matrix-Element gemeint. Wird der Zielvariablen der Wert eines numerischen Ausdrucks zugewiesen, wird dieser berechnet und direkt übertragen.
Soll ihr eine String-Variable oder ein String-Literal zugewiesen werden, wird der String soweit übertragen, bis

- das Ende des Strings erreicht ist.
- ein nichtnumerisches Zeichen auftritt

Ausnahmen: "+" oder "-" an erster Stelle,
"E" als Zeichen für Exponent,
Dezimaltrennzeichen

Ein numerischer Wert in Exponentdarstellung wird vor der Übertragung - falls möglich - in eine Zahl mit Nicht-Exponentdarstellung umgerechnet. Wird kein numerisches Zeichen übertragen, erhält die Zielvariable den Wert 0 .

- Beispiele:

```
10 LET A=5
20 B=B+1
:
70 LET C(J)=(X-A)/2+J
:
100 X="3.75E+02WERT"  (Ergebnis: X=375)
```

Anhand der Anweisung mit Zeilennummer 20 soll hier zum besseren Verständnis eine LET-Anweisung in einzelne Arbeitsschritte zerlegt werden:

1. Lade den Inhalt von B in ein Rechenregister.
2. Addiere 1 zum Inhalt des Rechenregisters.
3. Stelle den Inhalt des Rechenregisters wieder in der Variablen B ab.

Die BASIC-Anweisungen

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

String-Variable als Zielvariable

Unter String-Variable wird eine vollständige String-Variable oder ein zusammenhängender Teil einer String-Variablen, d.h. eine mit ein oder zwei Indizes versehene String-Variable verstanden.
Einer String-Variablen kann der Wert eines String-Ausdrucks zugewiesen werden. Die Wertzuweisung von String-Ausdrücken ist nur bei String-Variablen als Ziel möglich.

Soll einer String-Variablen ein numerischer Ausdruck zugewiesen werden, wird zunächst der Wert des numerischen Ausdrucks berechnet. Anschließend wird der Wert über eine Maske Zeichen für Zeichen in ASCII-Code umgewandelt und in die String-Variable übertragen. Die Maske wird über die USING-Angabe vorgegeben oder es wird eine Standardmaske eingesetzt. Sie hat folgenden Aufbau:

- Vorzeichen (1 Byte), für "+" wird ein Leerzeichen und für "-" ein Minus-Zeichen im Ziel-String abgestellt.
- Im Anschluß an das Vorzeichen wird der Wert des Ausdrucks einschließlich Dezimalpunkt und Exponent (bei Gleitkommadarstellung) übertragen.
- Im Anschluß an die zuletzt übertragene Ziffer wird ein Leerzeichen übertragen.

Ohne Maskenangabe (USING) erfolgt die Übertragung im Gleitkommaformat, wenn ein Wert entsteht:

- größer als 99999999999999
- kleiner als 0.1

Anmerkung:

Die Übertragung in einen String erfolgt von links nach rechts. Sie wird beendet durch:

- Erreichen der dimensionierten Anzahl Bytes im Ziel-String.
- Erreichen des Grenzzeichens im Quell-String oder des Ende eines String-Literals.
- Erreichen des Endekriteriums im letzten Element, wenn als Quelle ein String-Ausdruck dient.
- Nach der Übertragung des abschließenden Leerzeichens bzw. der letzten Ziffer (bei USING) des Wertes eines numerischen Ausdrucks.

Die BASIC-Anweisungen

Grenzzeichenverarbeitung

Für einen String wird bei der Dimensionierung festgelegt, wieviele Zeichen er maximal aufnehmen kann. Intern wird noch ein zusätzliches Byte für das Grenzzeichen reserviert.
Da ein String nicht in seiner ganzen dimensionierten Länge mit Daten belegt sein muß, schreibt RUN nach einer LET- und INPUT-Anweisung hinter das letzte übertragene Zeichen ein Grenzzeichen.

Ist die String-Variable in der gesamten dimensionierten Länge mit Daten belegt, steht das Grenzzeichen außerhalb des vom Programm adressierbaren Bereichs.

Ausnahme:

Bei Arbeiten mit Anfang- und Ende-Indizes im Ziel-String wird, falls kein Grenzzeichen überschrieben wird, kein Grenzzeichen abgestellt.

Wenn bei einer LET- oder INPUT-Anweisung der Ziel-String mit Anfang- und Ende-Indizes adressiert wird und die Anzahl zu übertragender Zeichen ist kleiner als der aufnehmende Bereich im Ziel-String, werden die Zeichen rechts vom adressierten Ende-Byte bis zum ersten Grenzzeichen bündig an die übertragenen Zeichen herangeschoben.

- Beispiele:

  - Übertragung des Literals "STATUS: " und des Inhalts von M$ in die Variable A$.

```
  :
770 LET M$="DATEI NICHT GEFUNDEN"
  :
1000 LET A$="STATUS: ",M$
```

A$ nach der Übertragung:

```
STATUS:  DATEI NICHT GEFUNDEN
```

Die BASIC-Anweisungen

- Füllen der Variablen M$ mit: ....5....0....5...

```
:
50 DIM M$(50)
:
570 M$="....5....0",M$
```

M$ nach der Übertragung:
....5....0....5....0....5....0....5....0....5....0

Da die Übertragung von links nach rechts läuft, wird das Literal auf Byte 1 bis 10 in M$ abgestellt. Anschließend wird M$ als Ziel- und Quellstring interpretiert.
In diesem Beispiel wird:
- Byte 1 nach Byte 11,
- Byte 2 nach Byte 12 usw.

übertragen, bis das dimensionierte Ende von M$ (Ziel) erreicht ist.

- Alle Bytes von M$ mit Leerzeichen füllen.

```
1000 LET M$=" ",M$
```

- In die Zeichen 3 bis 5 von A$ und B$ das String-Literal "$$$" ohne bzw. mit Erzeugung eines Grenzzeichens abstellen.

```
490 LET A$(3,5)="$$$"      570 LET B$(3)="$$$"
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.2.1.1 Numerische Ausdrücke

Syntax:

⟨NAusdr⟩::= [+|-] ⟨Operand⟩ [⟨NOper⟩⟨Operand⟩]^{n}

wobei:

$$\langle\text{Operand}\rangle ::= \left\{\begin{array}{l}\langle\text{NVar}\rangle \\ (\langle\text{NAusdr}\rangle) \\ \langle\text{Dezzahl}\rangle \\ \text{FN}\langle\text{Bu}\rangle(\langle\text{NAusdr}\rangle) \\ \langle\text{NFkt}\rangle\end{array}\right\}$$

Ein numerischer Ausdruck gibt die Regel zur Berechnung eines numerischen Wertes an und wird gebildet durch Verknüpfung von Operanden mittels numerischer Operatoren. Der erste Operand kann ohne Verwendung von Klammern mit einem Vorzeichen versehen werden. Hat der zweite oder ein folgender Operand ein Vorzeichen, sind zusätzliche Klammern erforderlich!

Bei der Berechnung eines Ausdrucks gilt die übliche mathematische Rechenhierarchie:

1. Berechnung von Ausdrücken in Klammern
2. Berechnung von Exponentiationen
3. Berechnung von Multiplikationen und Divisionen
4. Berechnung von Additionen und Subtraktionen

Die Berechnung der Klammerausdrücke geht nach der gleichen Hierarchie vor. Sind also mehrere Klammern ineinander verschachtelt, werden die Klammern "von innen nach außen" bearbeitet. Überflüssige Klammerpaare werden ignoriert.

- Beispiel:

```
Numerischer Ausdruck:  (2*(8-4))/(2↑2*2)
nach dem 1. Schritt :  (2*  4  )/( 4 *2)
nach dem 2. Schritt :    8       /    8
nach dem 3. Schritt :            1
```

- Beispiele für zulässige numerische Ausdrücke:

```
-(C+1)        Z*(-4)          +99          A(3,3)*A(3,1)↑2

B1*2/(-3.14)+INT(X+.5)        -(Y1/2.)*FNS(X+5)*FNS(A3)
```

Die BASIC-Anweisungen

6.2.1.2 Verknüpfung von Werten mit verschiedenen Genauigkeiten

Werte mit unterschiedlichen Genauigkeiten können beliebig miteinander verknüpft werden.

- Beispiel:

```
10 DIM 1%,A,2%,B,3%,C,4%,D
:
50 LET E=A*B+C/D
```

Zwischenergebnisse werden grundsätzlich mit größtmöglicher Genauigkeit gebildet.

Die Darstellung des Ergebnisses richtet sich allerdings nach der Genauigkeit der Zielvariablen.

6.2.1.3 Aufruf von Anwenderfunktionen

Syntax:

FN⟨Bu⟩(⟨NAusdr⟩)

Die mit FN⟨Bu⟩ angesprochene Anwenderfunktion muß vor dem Aufruf in einer DEF FN-Anweisung definiert worden sein.

Beim Aufrufen einer Anwenderfunktion wird der Wert von ⟨NAusdr⟩ für das formale Argument auf der rechten Seite der Definition eingesetzt und der Wert des so entstehenden numerischen Ausdrucks berechnet.

- Beispiel:

```
10 DEF FNY(Z)=5*Z+17*(3-Z)
:
40 LET X=20
50 LET A=20+FNY(X-18)
```

Für das formale Argument Z wird beim Funktionsaufruf X-18 eingesetzt, der so entstehende numerische Ausdruck lautet:

```
5*(X-18)+17*(3-(X-18))
```

Der Wert dieses Ausdrucks berechnet sich zu 27, so daß der Variablen A der Wert 47 zugewiesen wird.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.2.1.4 Aufruf von numerischen BASIC-Funktionen

Business-BASIC stellt dem Programmierer eine Reihe von Funktionen (<NFkt>) zur Verfügung, die einen numerischen Funktionswert liefern und die wie folgt aufgerufen werden:

<NFkt>::= { <FktName>(<NAusdr>) | LEN(<SVar>) | DET(<MVar>) }

Die Angabe der Klammern kann entfallen, wenn <NAusdr> ein nicht zusammengesetzter numerischer Ausdruck ist. Ebenso können bei der LEN-Funktion die Klammern entfallen.

Die vom BASIC-System zur Verfügung gestellten Funktionen sind:

SIN	COS	TAN	ATN	SQR	LOG	EXP	ABS	SGN
INT	FRA	RND	NOT	MAN	CHR	IXR	SPC	DET
CHF	LEN	KEY	LKY					

Eine eingehende Beschreibung der BASIC-Funktionen erfolgt im Kapitel "BASIC-Funktionen".

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.2.1.5 Masken

Eine Maske ist ein String-Literal oder eine String-Variable. Sie dient zur Aufbereitung des Wertes von numerischen Ausdrücken als Zeichenkette.

Die Aufbereitung mittels Masken ist in den Anweisungen

- LET
- PRINT
- PRINT #

möglich. Während bei den Anweisungen PRINT und PRINT # die Aufbereitung direkt in den Ausgabepuffer erfolgt, erfolgt bei LET die Aufbereitung in einer Ziel-Variablen (String-Variable). Zur Aufbereitung der Werte dienen die Maskenzeichen:

```
#   +   -   $   *   ,   .   ↑   &
```

die jeweils eine bestimmte Aufbereitungsfunktion auslösen. Alle Zeichen ungleich dieser Steuerzeichen werden identisch übernommen, sie dürfen allerdings nur am Anfang oder am Ende der Maske angegeben werden.

Masken werden von links nach rechts abgearbeitet. Die Anzahl der übertragenen Zeichen ist abhängig von der Anzahl Zeichen der Maske. Ist die Anzahl Ziffern des numerischen Ausdrucks größer als die Anzahl Zeichen der Maske, die Ziffern aufnehmen können, werden nur Sterne (*) in den Zielstring bzw. den Ausgabepuffer übertragen.

Sind Masken in String-Variablen abgestellt, werden diese nur bis zum Auftreten eines Grenzzeichens bearbeitet.

● Beispiel:

```
10 DIM A$(10)              /* MASKENFELD DIM.
:
50 LET A$="+##,###.##"
:
90 LET X$=A USING A$(2)    /* AUFBEREITUNG OHNE VORZ.
```

Die BASIC-Anweisungen

Masken-Zeichen	Funktion
#	Übertragen von Ziffern. Führende Nullen werden durch Leerzeichen ersetzt.
*	Übertragen von Ziffern. Führende Nullen werden durch Sterne (*) ersetzt.
&	Übertragen von Ziffern. Führende Nullen werden ausgegeben.
+	Übertragen von Ziffern mit Vorzeichen. "+"-Zeichen müssen, wenn sie auftreten, die ersten Zeichen der Maske bilden. Das Vorzeichen des Wertes wird explizit als "+" oder "-" (auf einem der "+"-Felder) den Ziffern vorangestellt.
-	Übertragen von Ziffern mit Vorzeichen " " bei positivem und "-" bei negativem Wert. "-"-Zeichen müssen, wenn sie auftreten, die ersten Zeichen der Maske bilden.
$	Übertragen von Ziffern mit Voranstellen eines $-Zeichens. Achtung: Bei einem Wert mit gleicher Ziffernanzahl wie Maskenzeichenanzahl wird a) das führende $-Zeichen unterdrückt, wenn ein Vorzeichen als erstes Maskenzeichen auftritt. b) ein $-Zeichen und sonst *-Zeichen übertragen, wenn kein Vorzeichen als Maskenzeichen auftritt.
↑	Übertragen des Exponenten in der Form E±xx. Soll ein Exponent ausgegeben werden, müssen genau vier ↑ am Ende der Maske und mindestens ein Ziffernzeichen (#, *, &, $, + oder -) vorher auftreten (bei mehr Ziffern als Ziffernzeichen wird die Mantisse abgerundet). Achtung: Die Angabe von vier ↑ ohne vorherige Ziffernzeichen ist unzulässig!

Die BASIC-Anweisungen

Masken-Zeichen	Funktion
.	Einfügen des Dezimalpunkts. Ein Dezimalpunkt darf nur einmal in einer Maske auftreten. Hat die zu übertragende Zahl mehr Nachkommastellen als die Maske Ziffern nach dem Dezimalpunkt, werden die letzten Stellen abgeschnitten. Enthält eine Maske keinen Dezimalpunkt, wird nur der Vorkommateil eines Wertes übertragen.
,	Einfügen des Trennsymbols ",". Ein Komma darf in einer Maske mehrfach auftreten. Ist, wenn ein Trennungssymbol auftritt, noch keine Ziffer übertragen worden, wird anstelle des Trennungssymbols das aktuelle Ersetzungszeichen übertragen. Bei der Kombination der Ersetzungszeichen "*" und "#" ist das "," als Trennsymbol nicht zulässig.

Anmerkungen:

Ist die Anlage mit einem Dezimalkomma statt einem Dezimalpunkt konfiguriert, vertauschen sich die Bedeutungen von "." und "," nicht, es werden jedoch ein Dezimalkomma und als Trennsymbol "." übertragen.

Folgende Einschränkungen sind beim Setzen der Steuerzeichen zu beachten:

- "+" oder "-"-Zeichen müssen zu Beginn der Maske stehen und schließen sich gegenseitig aus.
- $-Zeichen dürfen nicht nach #- ,*- oder &-Zeichen stehen.
- *-Zeichen dürfen nicht nach #-Zeichen stehen.
- Bei der Kombination von *- und #-Zeichen ist nur der Dezimalpunkt als Trennsymbol erlaubt.
- Mit dem ","-Zeichen kann nur der Vorkommateil unterteilt werden.
- &- und *-Zeichen schließen sich gegenseitig aus.
- &- und #-Zeichen schließen sich gegenseitig aus.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

- Beispiele:

  Anweisung: 50 LET A$=A USING B$

Maske (B$)	Variable A	A$ nach LET
Zeit: ##.##	17.15	Zeit: 17.15
	1715	Zeit: **.**
	1.10	Zeit: 1.10
++++.++	0.20	+.20
+++#.##	0.20	+0.20
-$####	50.50	$ 50
	-1000	-$1000
	10000	10000
-&&&.&&	-0.20	-000.20

6.2.1.6 String-Ausdrücke

Syntax:

⟨SAusdr⟩::= ⟨String⟩ [,⟨String⟩]^{n}

String-Ausdrücke dürfen nur in LET-Anweisungen auftreten, wenn auf der linken Seite des "="-Zeichens eine String-Variable steht.

Ein String-Ausdruck erzeugt die Verkettung der Inhalte mehrerer String-Variablen und -Literale. Die String-Variablen werden dabei entsprechend ihrer aktuellen Länge in die Zielvariable übertragen. Zwischen die einzeln genannten String-Variablen und -Literale wird kein Grenzzeichen abgestellt.

- Beispiele:

```
10 DIM A$(10),B$(5),C$(12)
20 LET A$="ABCDEFG"
30 LET C$=A$,"123456"
40 PRINT C$                --> Anzeige: ABCDEFG12345

50 LET B$="$",B$
60 PRINT B$                --> Anzeige: $$$$$

70 LET B$="←261←←262←←200←←263←",A$
80 PRINT B$                --> Anzeige: 123A   ("←200←"
                               erzeugt keine Anzeige)
```

6.2.1.7 Die CLEAR-Anweisung

Syntax:

CLEAR ⟨VarName⟩ [,⟨VarName⟩]^{n}

Die CLEAR-Anweisung löscht typabhängig Variablenwerte.

Einfache numerische Variablen erhalten den Wert 0, ebenso alle Elemente von Vektoren und Matrizen, Stringvariable werden in der ganzen dimensionierten Länge mit Grenzzeichen belegt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.2.2 Die DATA-Anweisung

Syntax:

DATA [<Format>,] <Dezzahl> [,<Dezzahl>]^{n}

Mit der DATA-Anweisung werden numerische Konstanten im Datenbereich des BASIC-Programms abgestellt.

Gleichzeitig kann für diese Konstanten die Genauigkeit durch eine Formatangabe festlegt werden. Diese Angabe kann in einer DATA-Anweisung nur einmal gemacht werden und gilt nur für die Konstanten dieser einen Anweisung. Wird keine Formatangabe gemacht, wird 2% angenommen. Es können maximal 63 Konstanten angegeben werden. Die Interpretation der DATA-Anweisungen und die Übernahme der Konstanten erfolgen zur Ladezeit. Ein mehrmaliges Durchlaufen dieser Anweisungen zur Programm-Laufzeit bleibt daher ohne Wirkung. Die Konstanten aus den Anweisungen werden von links nach rechts übernommen. Die Anweisungen werden in der Reihenfolge ihres Auftretens bearbeitet.

Anmerkung:

Die Formatangabe für Konstanten in DATA-Anweisungen ist unabhängig von dem Format-Schalter (in DIM-Anweisungen) für numerische Variablen.

- Beispiel:

  Abstellen der 1-Wort-Konstanten 10, 20 und 30 sowie der 2-Wort-Konstanten 10000 und 20000.

```
:
980 DATA 1%,10,20,30
990 DATA 10000,20000
```

6.2.3 Die READ-Anweisung

Syntax:

READ ⟨NVar⟩ [,⟨NVar⟩]^{n}

Mit dieser Anweisung werden Konstante, die mit DATA-Anweisungen angelegt wurden, in beliebige numerische Variablen übernommen. Die Konstanten sind im Datenbereich der Partition abgestellt.

Die Bearbeitung erfolgt sequentiell. Es wird intern ein Zeiger geführt, der vor der ersten READ-Anweisung auf die erste Konstante verweist. Für jede Konstante, die übernommen wird, wird der Zeiger erhöht und verweist auf die folgende Konstante.
Dieser Zeiger kann per RESTOR-Anweisung zurückgesetzt werden.

- Beispiel:

Übertragen der unter Zeilennummer 30 und 40 definierten Konstanten in die Vektoren A und B.

```
:
30 DATA 1%,10,20,30,40,50
40 DATA 1%,3,6,9,12,15
:
1000 FOR I=0 TO 4        /*5 DURCHLÄUFE
1010 READ A(I)           /*KONSTANTE ÜBERNEHMEN
1020 NEXT I              /*SCHLEIFENENDE
1030 FOR I=0 TO 4        /*5 DURCHLÄUFE
1040 READ B(I)           /*KONSTANTE ÜBERNEHMEN
1050 NEXT I              /*SCHLEIFENENDE
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.2.4 Die RESTOR-Anweisung

Syntax:

```
RESTOR [<Zlnr>]
```

Mit der RESTOR-Anweisung wird der intern geführte Zeiger für die mit DATA angelegten Konstanten auf die erste Konstante zurückgesetzt.

Wird RESTOR mit einer Zeilennummer angegeben, steht nach der Ausführung der Zeiger auf dem ersten Datenelement der ersten DATA-Anweisung, die nach dieser Zeilennummer auftritt. Die auf die RESTOR-Anweisung folgende READ-Anweisung beginnt die Konstantenübertragung ab der aktuellen Zeigerposition.

- Beispiel:

  Übertragen der unter Zeilennummer 30 und 40 definierten Konstanten in Variablen. Im Anschluß daran wird der interne Zeiger auf die erste Konstante von Anweisung 40 zurückgesetzt und diese in die Variable X übertragen.

```
:
30 DATA 1,2,5
40 DATA 125,135
:
60 READ A0,A1,A2,B,C
70 RESTOR 40
80 READ X
```

6.2.5 Die RANDOM-Anweisung

Mit der RANDOM-Anweisung können durch die Funktion RND kontrolliert Folgen von Zufallszahlen generiert werden. Beim Arbeiten mit Zufallszahlen entsteht das Problem, wie sich das Programm bei verschiedenen den Programmablauf beeinflussenden Folgen verhalten wird. Aus diesem Grunde besteht die Möglichkeit, die Folge der Zufallszahlen mit der RANDOM-Anweisung zu steuern.

Syntax:

RANDOM ⟨NAusdr⟩

⟨NAusdr⟩ : Funktionscode, der die Zufallszahlenfolge steuert.

⟨NAusdr⟩	Wirkung
0	Die erzeugte Folge von Zufallszahlen kann nicht vorausgesehen werden. Bei jedem Programmlauf wird eine andere Zahlenfolge generiert.
⟨⟩0	Generierung einer bestimmten Folge von Zufallszahlen.

Verschiedene Werte führen zu verschiedenen Folgen von Zufallszahlen. Gleiche Werte bewirken gleiche Folgen.

Die Funktion RND ohne RANDOM ist in ihrer Wirkung wiederum anders als beim Aufruf mit RANDOM:

- Beim Laden und Ausführen des Programms wird immer die gleiche Folge von Zufallszahlen erzeugt.
- Wird das Programm mehrmals ausgeführt, ohne daß es neu geladen wird, kann die Zufallszahlenfolge nicht vorausgesehen werden.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.2.6 Die MAT-Anweisung

Die MAT-Anweisung ist die Wertzuweisung für Matrizen in ihrer Gesamtheit.

Matrizen sind zweidimensionale Datenfelder bestehend aus 2 bis n Reihen und 2 bis n Spalten. Ein Element einer Matrix entspricht im Aufbau einer numerischen Variablen und kann 1 bis 4 Worte groß sein. Alle Elemente einer Matrix haben das gleiche Format.

Achtung:

Die Numerierung der Reihen und Spalten einer Matrix beginnt bei 0, die Übertragung bei den MAT-Anweisungen aber erst bei Reihe 1 und Spalte 1.

Syntax:

```
MAT <MVar>={ <MVar>
           { <MVar> {+} <MVar>
           {        {-}
           {        {*}
           { (<NAusdr>)*<MVar>
           { INV(<MVar>)
           { MIV(<MVar>)
           { TRN(<MVar>)
           { ZER[(<NAusdr>[,<NAusdr>])]
           { CON[(<NAusdr>[,<NAusdr>])]
           { IDN[(<Nausdr> ,<NAusdr> )]
```

Die BASIC-Anweisungen

6.2.6.1 Zuweisung einer einfachen Matrix-Variablen

Syntax:

MAT ⟨MVar1⟩=⟨MVar2⟩

Die Elemente der Ziel-Matrix erhalten den Wert der entsprechenden Elemente der Matrix ⟨MVar2⟩.

Die Matrix ⟨MVar1⟩ muß nicht dimensioniert sein. Eine dimensionierte Matrix kann eine andere Dimension erhalten, die Anzahl der Elemente darf jedoch nicht vergrößert werden. Nach Ausführung der MAT-Anweisung hat ⟨MVar1⟩ die gleiche Dimension wie ⟨MVar2⟩.

- Beispiel:

Übertragen des Inhaltes der Matrix A in die Matrix B. Matrix A besteht aus 3 Reihen und 3 Spalten. Matrix B besteht aus 4 Reihen und 3 Spalten.

```
:
50 MAT B=A
```

Matrix A

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

Matrix B vorher

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Matrix B nachher

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

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.2.6.2 Matrizen-Addition und -Subtraktion

Syntax:

MAT ⟨MVar1⟩=⟨MVar2⟩ [+|-] ⟨MVar3⟩

Die Elemente der aufnehmenden Matrix erhalten den Wert der Summe/Differenz der entsprechenden Elemente von ⟨MVar2⟩ und ⟨MVar3⟩. ⟨MVar2⟩ und ⟨MVar3⟩ müssen die gleiche Dimension haben.
Ist ⟨MVar1⟩ nicht dimensioniert, erhält sie die gleiche Dimension wie ⟨MVar2⟩ und ⟨MVar3⟩. Ist ⟨MVar1⟩ bereits dimensioniert, muß die Anzahl der Elemente gleich oder größer als die Anzahl von ⟨MVar2⟩ und ⟨MVar3⟩ sein.

Mit ⟨MVar1⟩ und ⟨MVar2⟩ und/oder ⟨MVar3⟩ darf die gleiche Matrix angesprochen werden.

- Beispiel:

  Addition der Matrix A mit sich selbst und Zuweisung in sich selbst.

```
:
50 MAT A=A+A
```

Matrix A vorher

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

Matrix A nachher

$$\begin{pmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}$$

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.2.6.3 Matrizen-Multiplikation

Syntax:

MAT ⟨MVar1⟩=⟨MVar2⟩*⟨MVar3⟩

Die Multiplikation von Matrizen ist keine elementweise Multiplikation der entsprechenden Matrixelemente. Dem (I,J)-Element der Ziel-Matrix wird das Skalarprodukt der I-ten Reihe von ⟨MVar2⟩ mit der J-ten Spalte von ⟨MVar3⟩ zugewiesen (siehe auch das folgende Beispiel).

Die beiden Matrizen ⟨MVar2⟩ und ⟨MVar3⟩ müssen nicht quadratisch sein, jedoch muß die Anzahl der Spalten von ⟨MVar2⟩ gleich der Anzahl der Reihen von ⟨MVar3⟩ sein. ⟨MVar1⟩ darf mit ⟨MVar2⟩ oder ⟨MVar3⟩ nicht übereinstimmen.

Die Matrix ⟨MVar1⟩ erhält durch die Multiplikation folgende Dimension:

- Anzahl Reihen wie ⟨MVar2⟩
- Anzahl Spalten wie ⟨MVar3⟩

Ist ⟨MVar1⟩ bereits dimensioniert, darf sich dadurch die Anzahl ihrer Elemente nicht vergrößern.

- Beispiel:

  Die Matrizen B und C werden miteinander multipliziert. Das Ergebnis der Matrizen-Multiplikation wird in der Matrix A abgestellt.

```
10 DIM B(2,3),C(3,2)
:
50 MAT A=B*C
```

Matrix B Matrix C Matrix A

$$\begin{pmatrix}1 & 2 & 3\\3 & 2 & 1\end{pmatrix} \qquad \begin{pmatrix}1 & 2\\2 & 1\\3 & 3\end{pmatrix} \qquad \begin{pmatrix}14 & 13\\10 & 11\end{pmatrix}$$

Die Berechnung der Elemente von A sei am Beispiel A(1,2) erklärt. A(1,2) ergibt sich als Skalarprodukt des 1-ten Reihenvektors von B und des 2-ten Spaltenvektors von C:

$$A(1,2)=\sum_{I=1}^{3} B(1,I)*C(I,2)=1*2+2*1+3*3=13$$

6.2.6.4 Skalarmultiplikation mit einer Matrix

Syntax:

MAT ⟨MVar1⟩=(⟨NAusdr⟩)*⟨MVar2⟩

Den Elementen der Matrix ⟨MVar1⟩ wird das Produkt von ⟨NAusdr⟩ mit den entsprechenden Elementen von ⟨MVar2⟩ zugewiesen.

⟨MVar1⟩ muß nicht dimensioniert sein. Ist ⟨MVar1⟩ bereits dimensioniert, darf die Anzahl ihrer Elemente die Anzahl Elemente von ⟨MVar2⟩ nicht unterschreiten.
Nach der Ausführung hat ⟨MVar1⟩ die gleiche Dimension wie ⟨MVar2⟩.

⟨MVar1⟩ und ⟨MVar2⟩ können identisch sein.

- Beispiel:

  Multiplikation der Matrix A mit dem Wert der Variablen B.

```
10 DIM A(3,3)
:
50 LET B=3
:
90 MAT A=(B)*A
```

Matrix A vorher

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

Matrix A nachher

$$\begin{pmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{pmatrix}$$

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.2.6.5 Die Funktion INV

Syntax:

MAT ⟨MVar1⟩=INV(⟨MVar2⟩)

Die Funktion INV invertiert eine vorgegebene quadratische Matrix (das Produkt einer invertierten Matrix mit der Matrix selbst ergibt die Einheitsmatrix, siehe auch Funktion IDN). Der dabei benutzte Algorithmus ist eine Variante des Gaußschen Eliminierungsverfahrens mit Zeilenvertauschungen.

Die Matrix ⟨MVar1⟩ muß nicht dimensioniert sein. Ist ⟨MVar1⟩ dimensioniert, muß deren Anzahl Elemente mindestens gleich der Anzahl Elemente von ⟨MVar2⟩ sein. Nach der Zuweisung hat ⟨MVar1⟩ die gleiche Dimension wie ⟨MVar2⟩ (⟨MVar2⟩ muß quadratisch sein).

Anmerkung:

Zu einer quadratischen Matrix gibt es nur dann eine Inverse, wenn ihre Determinante nicht Null ist.

- Beispiel:

  Das Beispiel berechnet von einer 2x2-Matrix die zugehörige Inverse und das Produkt der beiden Matrizen.

```
20 DIM A(2,2),B(2,2),C(2,2)
:
50 MAT B=INV(A)
60 MAT C=B*A
```

Matrix A $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$ Matrix B $\begin{pmatrix}-2 & 1\\1.5 & -.5\end{pmatrix}$ Matrix C $\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$

6.2.6.6 Die Funktion MIV

Syntax:

MAT <MVar1>=MIV(<MVar2>)

Die Funktion MIV invertiert mit einem aufwendigeren Algorithmus auch solche quadratischen Matrizen, die von der Funktion INV nicht mehr invertiert werden können, obwohl ihre Determinanten von 0 verschieden sind.

- Beispiel:

```
20 DIM 2%, A(2,2),B(2,2)
:
50 MAT B=INV(A)
```

Matrix A

$$\begin{pmatrix} 1E\text{-}30 & 1E+34 \\ -1E\text{-}30 & 0 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} 9E+29 & -9.99999E+28 \\ 1E\text{-}34 & 1E\text{-}34 \end{pmatrix}$$

Bei der Berechnung eines Matrixelements geschieht ein Überlauf.(ERROR #15)
Ersetzt man die Funktion INV durch MIV, wird ein korrektes Ergebnis geliefert.

```
50 MAT B=MIV(A)
:
```

Matrix A

$$\begin{pmatrix} 1E\text{-}30 & 1E+34 \\ -1E\text{-}30 & 0 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} 0 & -1E+30 \\ 1E\text{-}34 & 1E\text{-}34 \end{pmatrix}$$

Achtung:

Die Funktion MIV überschreibt die Reihe 0 von <MVar1> mit Ausnahme des ersten Elements. Ist <MVar1> als 1%-Variable dimensioniert, wird auch der Inhalt dieses ersten Elements überschrieben.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.2.6.7 Die Funktion TRN

Syntax:

MAT ⟨MVar1⟩=TRN(⟨MVar2⟩)

Die Funktion TRN weist der Matrix ⟨MVar1⟩ die zu ⟨MVar2⟩ transportierte Matrix zu. Die transponierte Matrix wird durch Vertauschung von Spalten und Zeilen gebildet. (Reihen- und Spaltenangaben jedes Elements werden vertauscht).

⟨MVar1⟩ und ⟨MVar2⟩ müssen verschieden sein.

Die Matrix ⟨MVar1⟩ muß nicht dimensioniert sein. Ist ⟨MVar1⟩ dimensioniert, muß die Anzahl Elemente mindestens gleich der Anzahl Elemente von ⟨MVar2⟩ sein.

Die transponierte Matrix ⟨MVAR1⟩ hat die gleiche Dimension wie ⟨MVAR2⟩ bei vertauschter Anzahl Reihen und Spalten.

- Beispiel:

```
20 DIM A(2,3)
:
50 MAT B=TRN(A)
```

Matrix A

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

6.2.6.8 Die Funktion ZER

Syntax:

MAT <MVar>=ZER [(<NAusdr> [,<NAusdr>])]

Anlegen oder Zuweisen einer Null-Matrix.In einer Null-Matrix hat jedes Element den Wert 0.

Eine bestehende Matrix kann mit dieser Anweisung neudimensioniert werden, wenn in Klammern die Anzahl Reihen und Spalten angegeben werden. Dabei darf sich die Anzahl ihrer Elemente nicht vergrößern.

Neu anzulegende Matrizen müssen durch Angabe der Anzahl Reihen und Spalten dimensioniert werden.

Wird nur ein numerischer Ausdruck angegeben, wird eine Matrix mit 2 Spalten (einschließlich Spalte 0) gebildet.

- Beispiel:

```
10 DIM A(3,3)
:
50 MAT A=ZER(2,4)
60 MAT B=ZER(3)
```

Matrix A vorher

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}$$

Matrix A nachher

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Die BASIC-Anweisungen

6.2.6.9 Die Funktion CON

Syntax:

MAT <MVar>=CON [(<NAusdr> [,<NAusdr>])]

Alle Elemente der Matrix <MVar> werden auf 1 gesetzt.

Ist die angegebene Matrix noch nicht dimensioniert, muß sie durch Angeben der Anzahl Reihen und Spalten (in Klammern) dimensioniert werden. Eine bestehende kann auf die gleiche Weise eine andere Dimension erhalten. Dabei darf die Anzahl ihrer Elemente nicht vergrößert werden.

Wird nur ein numerischer Ausdruck angegeben, wird eine Matrix mit 2 Spalten (einschließlich Spalte 0) gebildet.

- Beispiel:

```
20 DIM A(3,3)
:
50 MAT A=CON(2,4)
```

Matrix A vorher

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}$$

Matrix A nachher

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.2.6.10 Die Funktion IDN

Syntax:

MAT ⟨MVar⟩=IDN [(⟨NAusdr⟩,⟨NAusdr⟩)]

Die Funktion IDN erstellt eine Einheitsmatrix: eine Einheitsmatrix ist immer quadratisch, die Elemente, welche die Hauptdiagonale bilden, haben den Wert 1, sämtliche anderen Elemente haben den Wert 0.

⟨MVar⟩ muß nicht dimensioniert sein. Soll ⟨MVar⟩ neudimensioniert oder erstellt werden, sind die Anzahl Reihen und Spalten in Klammern an das Funktionswort anzuhängen.

⟨MVar⟩ sollte quadratisch dimensioniert werden bzw. quadratisch dimensioniert sein, da nur in diesem Fall eine Einheitsmatrix sinnvoll ist. Wird eine bestehende Matrix neu dimensioniert, darf sich die Anzahl der Elemente nicht vergrößern.

- Beispiel:

```
20 DIM A(3,3)
:
50 MAT A=IDN
```

Matrix A vorher

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}$$

Matrix A nachher

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

6.2.7 Die Anweisung MAT READ

Syntax:

```
MAT READ <MVar> [(<NAusdr> [,<NAusdr>])]
                [,<MVar>[(<NAusdr> [,<NAusdr>])]]^n
```

Übertragen von Konstanten aus DATA-Anweisungen in Matrizen. Das Übertragen in eine Matrix erfolgt reihenweise und innerhalb der Reihen von links nach rechts.

Eine Matrix kann mit dieser Anweisung durch Angabe der Anzahl Reihen und Spalten erstmals oder redimensioniert werden. Bei einer Redimensionierung darf sich die Anzahl der Elemente nicht vergrößern.
Wird nur ein numerischer Ausdruck angegeben, wird die Matrix mit 2 Spalten (einschließlich Spalte 0) dimensioniert.

- Beispiel:

Übernahme von Konstanten aus der DATA-Anweisung auf Zeile 50 in die Matrizen A und B. Die Matrix A ist auf Zeile 30 für 4 Reihen und 4 Spalten dimensioniert. Die Matrix B wird mit der MAT READ-Anweisung dimensioniert.

```
:
30 DIM A(3,3)
:
50 DATA 2%,5,5,5,8,8,8,3,3,3,1,2,3,4,5,6,7,8
:
90 MAT READ A,B(2,4)
```

Matrix A

$$\begin{pmatrix} 5 & 5 & 5 \\ 8 & 8 & 8 \\ 3 & 3 & 3 \end{pmatrix}$$

Matrix B

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

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.3 Steuerung des Programmablaufs

Dieser Punkt enthält die Beschreibung der Anweisungen, die den Ablauf eines Programmes direkt oder in Abhängigkeit von Bedingungen steuern.
Diese Anweisungen sind:

- GOTO
- GOSUB
- RETURN
- ON
- IF
- IF ERR 0
- IF ESC
- FOR
- NEXT
- CHAIN
- STOP
- END
- LINK

6.3.1 Die GOTO-Anweisung

Syntax:

GOTO <Zlnr>

Diese Anweisung erzeugt im Programmablauf einen unbedingten Sprung zu der Anweisung mit der angegebenen Zeilennummer. Die Programmausführung wird mit dieser Anweisung fortgesetzt.

- Beispiel:

Sprunganweisung nach Zeilennummer 50.

```
:
100 GOTO 50
```

Die BASIC-Anweisungen

6.3.2 Die GOSUB-Anweisung

Syntax:

GOSUB ⟨Zlnr⟩

Die GOSUB-Anweisung erzeugt einen Sprung in eine Unterprogramm-Routine. Die Zeilennummer der ersten auszuführenden Anweisung des Unterprogramms ist anzugeben.

Da BASIC-Unterprogramme Bestandteile des Hauptprogramms sind, ist eine gesonderte Parameterübergabe überflüssig. Alle beim Unterprogrammaufruf aktuellen Variablen können benutzt und verändert werden.

Bei der Ausführung der GOSUB-Anweisung wird außerdem der Unterprogramm-Stufenzähler erhöht und die Zeilennummer der auf GOSUB folgenden Anweisung als Rückkehr-Adresse gespeichert.
Die GOSUB-Anweisung sollte nicht die letzte Anweisung des Programms sein.

Innerhalb eines Unterprogrammes können weitere Unterprogramme aufgerufen werden. Es stehen 32 Unterprogramm-Stufen zur Verfügung.

Ein Unterprogramm muß durch eine RETURN-Anweisung beendet werden.

- Beispiel:

  Aufruf eines Unterprogrammes, dessen erste auszuführende Anweisung die Zeilennummer 9000 trägt.

```
:
100 GOSUB 9000
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

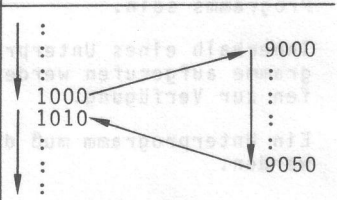
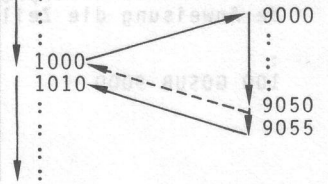
6.3.3 Die RETURN-Anweisung

Syntax:

RETURN ⟨NAusdr⟩

Rücksprung aus einem Unterprogramm. Der Rücksprung erfolgt auf die Anweisung nach der GOSUB-Anweisung, die das Unterprogramm aufgerufen hat.
Durch Angabe eines numerischen Ausdrucks hinter RETURN ist die Modifikation der Rückkehradresse möglich. Der Wert des Ausdrucks gibt die Anzahl Anweisungen an, um die die Rückkehradresse modifiziert werden soll. Als Basis für die Ermittlung der Rückkehradresse dient die bei der Ausführung der GOSUB-Anweisung gespeicherte Zeilennummer.

- Beispiele:

Programm	Ablauf
: 1000 GOSUB 9000 1010 ... : : : 9000 ... : 9050 RETURN	
1000 GOSUB 9000 1010 ... : : : 9000 ... : : 9050 IF X=0 RETURN-1 9055 RETURN	

Das erste Beispiel zeigt den Aufruf eines Unterprogrammes mit einem Rücksprung ohne Modifikation.

Im zweiten Beispiel wird, wenn X = 0 ist, auf die gespeicherte Rückkehradresse -1 Anweisung zurückgesprungen, also auf den Unterprogrammaufruf selbst.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.3.4 Die ON-Anweisung

Syntax:

ON ⟨NAusdr⟩ { GOTO / GOSUB } ⟨Zlnr⟩ [,⟨Zlnr⟩]^{n}

Sprung- oder Unterprogrammaufruf-Verteiler. Die Zeilennummer der nächsten auszuführenden Anweisung wird aus der Zeilennummern-Liste mittels einem in ⟨NAusdr⟩ abgestellten Wertes ausgesucht.

Der Sprung kann sowohl mit der GOTO- als auch mit der GOSUB-Anweisung durchgeführt werden.

Der Ganzzahlteil des Wertes von ⟨NAusdr⟩ legt fest, welches Listenelement die Zeilennummer der nächsten auszuführenden Anweisung bildet. Ist der Wert von ⟨NAusdr⟩ kleiner 1 oder größer als die Anzahl angegebener Zeilennummern, wird das Programm mit der folgenden Anweisung fortgesetzt.

Die Länge der Zeilennummernliste hat keinen Einfluß auf die Ausführungszeit der Anweisung.

- Beispiel:

```
:
50 ON X GOSUB 200,300,390
60 ...
```

Die Anweisungen 200,300,390 bilden Aufsetzpunkte eines oder verschiedener Unterprogramme. Die ON-Anweisung erzeugt einen Sprung nach Anweisung

- 200, wenn INT(X)=1
- 300, wenn INT(X)=2
- 390, wenn INT(X)=3

Beim Aufteten einer RETURN-Anweisung nach der betreffenden Zeilennummer wird mit Anweisung 60 fortgefahren. Ist INT(X) verschieden von 1, 2 und 3, wird sofort Anweisung 60 ausgeführt.

Die BASIC-Anweisungen

6.3.5 Die IF-Anweisung

Syntax:

IF ⟨logAusdr⟩ ⟨Anweisung⟩

Bedingte Ausführung einer Anweisung. Ist der Wert des logischen Ausdrucks "wahr", wird die nächste Bedingung abgefragt oder ⟨Anweisung⟩ ausgeführt. Ist der Wert "falsch", wird die Programmausführung mit der auf die IF-Anweisung folgenden Anweisung fortgesetzt.

String-Vergleich

Ist nur ein String-Ausdruck angegeben, wird geprüft, ob das erste Zeichen kein Grenzzeichen ist, d.h. ob kein Leerstring vorliegt. Ist dies der Fall, gilt die Bedingung als erfüllt.
Beim Vergleich von 2 String-Ausdrücken wird bis zum Auftreten eines Grenzzeichens verglichen. Bei Prüfung auf Identität müssen die String-Ausdrücke einschließlich des ersten Grenzzeichens identisch sein.
Anmerkung: In einer IF-Anweisung darf ein String-Ausdruck höchstens aus zwei String-Variablen oder -Literalen bestehen.

- Beispiel:

  Vergleichen des Inhalts der String-Variablen X$ mit den zusammengesetzen Inhalten von A$ und B$. Sind die Inhalte identisch, wird der Sprung nach Anweisung 1440 (GOTO 1440) ausgeführt. Bei Nichtgleichheit wird mit Anweisung 1120 fortgefahren.

```
:
1110 IF X$=A$,B$ GOTO 1440
1120 ...
```

Vergleich numerischer Werte

Ist nur ein numerischer Ausdruck angegeben, wird überprüft, ob dessen Wert ungleich 0 ist. Ist dies der Fall, gilt die Bedingung als erfüllt.

Die BASIC-Anweisungen

Bei zwei numerischen Ausdrücken wird überprüft, ob der dargestellte Vergleich wahr oder falsch ist, und je nach dem die Anweisung <Anweisung> ausgeführt oder übergangen. Beim Vergleich wird unabhängig von dem Format und der Datendarstellung der zu vergleichenden Ausdrücke nur der algebraische Wert verglichen.

Ist eine angegebene Variable eine Matrix oder ein Vektor, wird nur Element (0,0) bei Matrizen bzw. (0) bei Vektoren verglichen.

Verknüpfen mehrerer logischer Ausdrücke

Indem das Schlüsselwort IF jeweils mit einem logischen Ausdruck mehrmals angegeben wird, besteht die Möglichkeit, in einer Anweisung mehrere Bedingungen durch ein logisches "UND" miteinander zu verknüpfen. Es müssen also alle logischen Ausdrücke erfüllt sein, damit die Anweisung ausgeführt wird.

Eine "ODER"-Verknüpfung ist nur über zwei separate IF-Anweisungen möglich.

- Beispiele:

  - Vergleich zweier String-Variablen auf Gleichheit. Ist die Bedingung erfüllt, wird ein Programmsprung zur Zeilennummer 9000 durchgeführt.

    ```
    1110 IF A$=B$ GOTO 9000
    ```

  - Überprüfen einer String-Variablen, ob ihr Inhalt leer ist. Ist dies der Fall, wird der Text "FALSCHE EINGABE" am Bildschirm angezeigt.

    ```
    1120 IF A$="" PRINT "FALSCHE EINGABE"
    ```

  - Prüfen eines Strings auf den Inhalt "END". Ist dies der Fall, wird ein "CHAIN" zu dem TAMOS-Programm "TA.END" durchgeführt.

    ```
    1130 IF A$="END" CHAIN "TA.END"
    ```

  - Prüfen, ob A eine ganze Zahl zwischen 1 und 9 ist. Ist dies der Fall wird zur Zeilennummer 9000 gesprungen.

    ```
    1140 IF A>0 IF A<10 IF FRA A=0 GOTO 9000
    ```


© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.3.6 Die IF ERR 0-Anweisung

Syntax:

IF ERR 0 [<Anweisung>]

Abfangen der standardmäßigen BASIC-Fehleranzeige

ERROR # xx AT <Zlnr> IN <Programmname>

durch eine gezielte Reaktion innerhalb der Programmausführung. <Anweisung> ist die Anweisung, die beim Auftreten eines BASIC-Fehlers auszuführen ist.

In der Regel wird dies ein Sprung in ein Fehlerbehandlungs-Unterprogramm sein, in dem mit SPC 8 und SPC 10 sowohl die Fehlernummer als auch die Zeilennummer der verursachenden Anweisung festgestellt werden kann. Die RETURN-Anweisung erzeugt in diesem Fall einen Sprung auf die Anweisung nach der fehlerverursachenden Anweisung.

Mit der Anweisung IF ERR 0 REM bzw. IF ERR 0 /* kann die BASIC-Fehleranzeige ersatzlos unwirksam gemacht werden.

Während der Programmausführung kann nur eine IF ERR 0-Anweisung gespeichert werden. Das bedeutet, daß jede IF ERR 0-Anweisung die vorhergehende aufhebt.

Sollen BASIC-Fehler wieder in der Standardform angezeigt werden, kann die gezielte Fehlerreaktion durch die Anweisung IF ERR 0 (ohne <Anweisung>) aufgehoben werden.

Anmerkung 1:

Zum Zeitpunkt der Fehlererkennung kann die verursachende Anweisung bereits teilweise abgearbeitet sein. Das kann besonders bei den Anweisungen zu Problemen führen, die mehrere Operanden enthalten (z.B. OPEN #, CLOSE #).

Anmerkung 2:

Die Fehlernummern:

3,6,16,18,19,20,27,31,59,76,77,81,86

können mit IF ERR 0 nicht abgefangen werden.

Alle Kanäle werden geschlossen, und es wird in den BASIC-Processor verzweigt.

Ist das Programm aus einem TAMOS-Selektor gestartet worden, wird bei den oben genannten Fehlernummern die Fehlermeldung im Common-Bereich abgestellt und das Programm TA.ABO aufgerufen.

6.3.7 Die IF ESC-Anweisung

Syntax:

IF ESC [⟨Anweisung⟩]

Die ESCAPE-Tasten-Behandlung kann außer über IF ERR 0 mit Abfrage von Fehler 99 auch direkt mit dieser Anweisung durchgeführt werden. Bei Wirksamkeit von IF ESC ⟨Anweisung⟩ und Betätigung der ESC- oder der CTL-C-Taste wird die Anweisung ⟨Anweisung⟩ durchgeführt.

Ist außerdem IF ERR 0 ⟨Anweisung⟩ wirksam, ist eine ESC-Behandlung in der Fehlerroutine nicht mehr möglich. Mit IF ESC allein wird diese ESC-Tasten-Behandlung wieder unwirksam.

In der Regel wird es sich bei ⟨Anweisung⟩ um ein GOSUB handeln. Wird mehrmals die ESC-Taste gedrückt, wird ⟨Anweisung⟩ trotzdem nur einmal ausgeführt, im Gegensatz zu IF ERR 0, wo der GOSUB-Stack überlaufen kann!

Nach Aufruf eines Folgeprogramms durch CHAIN oder LINK muß im Folgeprogramm wieder die Anweisung IF ESC ⟨Anweisung⟩ stehen, wenn die ESCAPE-Tasten-Behandlung wieder genutzt werden soll.

GOTO als ⟨Anweisung⟩ ist nicht erlaubt.

REM als ⟨Anweisung⟩ unterdrückt die Wirkung der ESC-Taste.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Achtung:

Bei der Verwendung von IF ESC und IF ERR 0 sollte IF ESC vor IF ERR 0 stehen, sonst kann es vorkommen, das Escape bei IF ERR 0 abgefangen wird, wenn IF ESC noch nicht durchlaufen wurde.

6.3.8 Die FOR-Anweisung

Syntax:

FOR ⟨NVar⟩=⟨NAusdr1⟩ TO ⟨NAusdr2⟩ [STEP ⟨NAusdr3⟩]

Startanweisung einer Programmschleife.

⟨NVar⟩ : Laufvariable.

⟨NAusdr1⟩ : Anfangswert der Laufvariablen. Der Wert von ⟨NAusdr1⟩ wird bei der Ausführung der Anweisung FOR in ⟨NVar⟩ abgestellt.

⟨NAusdr2⟩ : Endwert, bei dessen Über-/Unterschreitung die Programmschleife verlassen und mit der auf die NEXT-Anweisung folgenden Anweisung fortgefahren wird. Bereits der Anfangswert der Laufvariablen wird auf Über-/Unterschreitung des Endwertes überprüft.

⟨NAusdr3⟩ : Schrittweite. Der Wert von ⟨NAusdr3⟩ wird bei jeder Wiederholung der Programmschleife zur Laufvariablen addiert. Ist STEP ⟨NAusdr3⟩ nicht angegeben, wird als Schrittweite +1 eingesetzt.

Anmerkung:

Programmschleifen dürfen sich nicht überschneiden, können aber bis zu achtmal ineinandergeschachtelt werden. ⟨NAusdr1⟩ und ⟨NAusdr2⟩ dürfen nur Werte mit maximal 6 signifikanten Stellen annehmen, da diese in 2%-Format gespeichert werden.

6.3.9 Die NEXT-Anweisung

Syntax:

NEXT ⟨NVar⟩

Letzte Anweisung einer Programmschleife. Es ist der Name der in der zugehörigen FOR-Anweisung am Schleifenanfang definierten Laufvariablen anzugeben.

In der NEXT-Anweisung wird die Schrittweite, die in der einleitenden FOR-Anweisung definiert wurde, auf die Laufvariable addiert. Ist der Endwert über-/unterschritten, wird die Programmschleife beendet und die auf NEXT folgende Anweisung ausgeführt.
Ist der Endwert noch nicht über-/unterschritten, wird zu der auf die FOR-Anweisung folgenden Anweisung gesprungen.

- Beispiele zu FOR und NEXT:

  - Zehnmaliges Durchlaufen einer Programmschleife.

```
  :
┌1000 FOR I=1 TO 10
│1005 ...
│ :
└1100 NEXT I
 1105 ...
```

  - Geschachtelte Programmschleife. Die äußere Schleife wird 5-mal, die innere je 10-mal, insgesamt also 50-mal durchlaufen.

```
┌1000 FOR I=1 TO 5
│1005 ...
│ :
│┌1100 FOR M=1 TO 10
│1105 ...
│ :
└1200 NEXT M
 :
└1300 NEXT I
 :
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.3.10 Die CHAIN-Anweisung

Syntax:

CHAIN <String>

Aufruf eines beliebigen BASIC-Programms oder Processors. Der als Variable oder Literal übergebene String darf eine Länge von 127 Zeichen nicht überschreiten.

Die String-Variable bzw. das String-Literal enthält den Namen des aufzurufenden Programms/Processors. Das aufrufende Programm wird beendet, und zusätzlich werden bei Aufruf eines Processors sämtliche Datenkanäle geschlossen. Bei Aufruf eines BASIC-Programms bleiben eröffnete Kanäle eröffnet.

Die direkte Datenübergabe von einem zum anderen Programmsegment innerhalb der Partition ist mit der CHAIN-Anweisung nicht möglich. Zur Datenübergabe an ein nachfolgendes BASIC-Programm bestehen die folgenden Möglichkeiten:

- Benutzen der Anweisung GLOBAL/LINK oder DEALLO/LINK statt CHAIN.

- Übergabe der Daten im Common-Bereich. Für jeden Teilnehmer besteht ein Common-Bereich in der Größe von 512 Byte. Dieser Bereich kann mittels CALL (siehe "CALL-Anweisungen") angesprochen werden.

- Datenübergabe in Plattendateien. Diese Möglichkeit sollte für Segmentwechsel nur in Ausnahmefällen angewendet werden, da sie relativ zeitaufwendig ist.

6.3.11 Die LINK-Anweisung

Syntax:

LINK [;<VarName>,] <String>

Die LINK-Anweisung ruft ein BASIC-Programm (-Programmsegment) auf und übergibt globale Variablen vom aufrufenden Programm an das aufgerufene.

Es bleiben die vom aufrufenden Programm eröffneten Kanäle, sowie die übergebenen Variablen mit ihren Namen, Werten, Dimensionen und Formaten für das aufgerufene Programm erhalten.

Es gibt 2 Varianten:

- Enthält das aufrufende Programm keine GLOBAL-Anweisungen, werden alle (mit DIM-Anweisungen oder implizit) dimensionierten Variablen des Programms übergeben, die nicht vor Erreichen der LINK-Anweisung durch eine DEALLO-Anweisung freigegeben wurden. Eine Angabe von <VarName> ist in diesem Fall wirkungslos.

  Im Folgesegment müssen alle übergebenen Variablen mindestens einmal genannt werden.

- Enthält das aufrufende Programm GLOBAL-Anweisungen, werden alle in diesen Anweisungen referierten Variablen übergeben. Im Folgeprogramm müssen diese Variablen in der gleichen Reihenfolge und Genauigkeit in GLOBAL-Statements vorkommen.

  Ist <VarName> angegeben, werden alle GLOBAL-Variablen nur bis einschließlich <VarName> übergeben.

  Vergleiche das Kapitel "Deklaration von Datenfeldern und Funktionen".

In der String-Variablen bzw. dem String-Literal wird der Name des aufzurufenden Programmes abgestellt. Es darf nicht mehr als ein Programmname angegeben werden.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Fehler:

ERROR	Bedeutung
44	Eine unzulässige Dateiart ist aufgetreten.
75	- Die angegebene Datei ist kein BASIC-Programm. - Das Programm ist nicht auf der angegebenen LU. Dieser Fehler kann über IF ERR 0 abgefangen werden.
76	Die Kapazität der Partition ist zu gering, um das Programm (-Segment) aufzunehmen. In diesem Fall bleibt die Steuerung beim aufgerufenen Programm.
77	Eine oder mehrere Variablen, die verfügbar bleiben sollen, sind im aufgerufenen Programm nicht definiert. Im Anschluß an die Fehlermeldung wird eine Liste der nicht definierten Variablen angezeigt. Die Meldung wird in die LOG-Datei eingetragen.
81	Das Programm ist größer als das Active-File.
85	Unterschiedliche Reihenfolge der GLOBAL-Variablen.

6.3.12 Die STOP-Anweisung

Syntax:

```
STOP
```

Die Anweisung unterbricht die Ausführung des Programms mit der Meldung:

```
STOP AT <Zlnr> STOP
```

und ruft den BASIC-Processor und den DEBUG-Modus auf. Alle Anweisungen nach der STOP-Anweisung sind noch nicht ausgeführt, und es werden auch keine Datenkanäle geschlossen.

Die STOP-Anweisung ist besonders für den Testlauf von Programmen geeignet: nach der Programmunterbrechung per STOP-Anweisung kann man im DEBUG-Modus weitertesten oder sich im Direktausführungsmodus Variableninhalte anzeigen lassen.

Nach der Testphase sollte die STOP-Anweisung aus dem Programm entfernt werden.

6.3.13 Die END-Anweisung

Syntax:

```
END
```

Mit der END-Anweisung wird die Ausführung eines BASIC-Programms beendet und sämtliche eröffneten Datenkanäle werden geschlossen. Anschließend befindet sich der Anwender im BASIC-Kommandomodus.

Soll nach Beendigung des Programmes ein anderes Programm oder ein Processor aufgerufen werden, so ist die END-Anweisung durch eine CHAIN- oder LINK-Anweisung zu ersetzen.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.4 Die Ein-/Ausgabeanweisungen

Die Anweisungen, die Ein-/Ausgaben auf beliebige angeschlossene Peripheriegeräte ermöglichen, sind:

- INPUT
- PRINT
- MAT INPUT
- MAT PRINT
- OPEN #
- ENV #
- CLOSE
- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #
- SEARCH #
- BUILD #
- KILL
- CALL

Sämtliche Anweisungen bis auf CALL werden im folgenden behandelt. Die CALL-Anweisungen werden im Kapitel "CALL-Anweisungen" vorgestellt.

Die BASIC-Anweisungen

6.4.1 Die Anweisungen zur Tastatur-Bildschirm-Ein-/Ausgabe

Zur Programmierung der Arbeitsplatz-Ein-/Ausgabe dienen die Anweisungen:

- INPUT : Eingabe über Tastatur
- MAT INPUT : Matrixeingabe über Tastatur
- PRINT : Bildschirm-Ausgabe
- MAT PRINT : Matrixausgabe am Bildschirm

sowie die Bildschirmfunktionen:

```
<DisFkt>::= TAB    (Tabulation, Cursor-Positionierung)
            'BS'   (Backspace)
            'MP'   (Mark Cursorposition)
            'BP'   (Back to marked Cursorposition)
            'CR'   (Carriage Return)
            'LD'   (Line Deletion)
            'LI'   (Line Insertion)
            'SB'   (Start Background)
            'SF'   (Start Foreground)
            'CS'   (Clear Screen)
            'CF'   (Clear Foreground)
            'BEL'  (Ring Keyboard Bell)
            'TB'   (Tab to next Foregroundfield)
            'CFF'  (Clear Foregroundfield)
            'CH'   (Cursor home)
            'DRK'  (Dark)
            'X'    (Hexadezimalausgabe)
            'DF'   (Duplicate Field)
            'DLC'  (Duplicate Characters Limited)
            'DC'   (Duplicate Characters)
            'DM'   (Display Memory)
            'CP'   (Cursor Position)
            'DW'   (Define Window)
            'NTY'  (No Typewriter Keyboard)
            'TY'   (Typewriter Keyboard)
"←376←←236←←210←" (Einschalten 8-Bit-Übertragung)
"←376←←236←←211←" (Ausschalten 8-Bit-Übertragung)
```

Die syntaktischen Gerätefunktionen (<DevFkt>) zur Programmierung von Peripheriegeräten finden Sie im Kapitel "Tastatur-Bildschirm-Ein-/Ausgabe".

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Jeder Bildschirm verfügt über einen intern geführten Zeiger (Cursor), der nur bei Eingaben am Bildschirm sichtbar wird. Ein- und Ausgaben werden grundsätzlich ab der aktuellen Cursorposition ausgeführt.

Außer durch Ein- und Ausgaben wird die Cursorposition durch die TAB-Funktion sowie einige andere Bildschirmfunktionen verändert.

6.4.1.1 Die INPUT-Anweisung

Syntax:

```
                  [ <Slit>,                          ]n {<NVar>}
                  | <DisFkt>,                          | {<SVar>}
INPUT             | TAB(<NAusdr>[,<NAusdr>]),          |
                  | <NVar>,                            |
                  [ <SVar>,                            ]
```

Eingabe von Daten über die Tastatur in beliebige numerische und String-Variablen. Zusätzlich können Führungstexte und Bildschirmfunktionen ausgegeben werden.

<Slit> : String-Literal, das als Führungstext vor der Eingabe am Bildschirm angezeigt wird.

<DisFkt> : Beliebige Bildschirmfunktion (siehe Beschreibung der Bildschirmfunktionen).

TAB : Tabulation des Cursors (Lichtzeiger am Bildschirm) auf eine bestimmte Position. Es kann
- eine Spalte in der aktuellen Zeile durch:
  TAB (<NAusdr>)
- eine Spalte und Zeile durch:
  TAB (<NAusdr>,<NAusdr>)

angegeben werden. Die Spaltenangabe muß in dem Bereich von 0 bis 79, die Zeilenangabe im Bereich von 0 bis 24 liegen.
Wird nur die Position innerhalb der Zeile angegeben (TAB(<NAusdr>)), muß diese größer als die aktuelle Position sein.

<NVar>
<SVar> : Numerische Variable, in die eingegeben werden soll. In einer INPUT-Anweisung können mehrere Eingabevariable angegeben werden.

Die BASIC-Anweisungen

Alle Operanden sind grundsätzlich durch Kommata zu trennen. Als letzter Operand muß eine Eingabevariable angegeben sein!

Beim Ausführen einer INPUT-Anweisung wird das Programm in Wartezustand versetzt, der Cursor erscheint am Bildschirm, und bis zum Ende der Eingabe erhält das Programm keine Rechnerzeit mehr ("INPUT-mode").
Sind mehrere Eingabevariablen einzugeben, muß die Eingabe jeder Variablen mit der Taste "CR" ausgelöst werden.

Jeder Bildschirmarbeitsplatz hat einen Ein-/Ausgabepuffer in der Größe von 256 Byte, von denen 40 (beim DAP4 80) Byte als Voreingabepuffer reserviert sind. Damit ist schon vor dem Auftreten des "INPUT-mode" die Eingabe von bis zu 40 (80 beim DAP4) Zeichen und Auslösetasten möglich. Werden mehr Zeichen eingegeben, ertönt der akustische Signalgeber, und die zuviel eingegebenen Zeichen werden nicht übernommen.

Eingegebene Zeichen werden, sobald sich das Programm im "INPUT-mode" befindet, direkt ab der aktuellen Cursorposition auf dem Bildschirm angezeigt. Voreingegebene Zeichen werden erst zum Zeitpunkt des Umschaltens auf "INPUT-mode" angezeigt.
Werden zwei Tasten gleichzeitig gedrückt, ertönt der akustische Signalgeber, und keines der Zeichen wird übernommen.

Steht der Cursor in einem Hintergrundfeld, können keine Zeichen eingegeben werden. Ist ein Eingabefeld auf dem Bildschirm durch ein Hintergrundzeichen begrenzt, ist eine Eingabe ab dieser Position nicht mehr möglich.

Die Anweisung INPUT bietet zusätzlich die Möglichkeit, Führungstexte und Bildschirmfunktionen vor der eigentlichen Eingabe auszugeben. Damit übernimmt die INPUT-Anweisung in gewissem Sinn auch Ausgabefunktionen. Führungstexte dürfen allerdings nicht in Variablen stehen, sondern müssen als String-Literale angegeben werden.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

- Beispiel:
  - Ausgabe des Führungstextes "RABATT" ab Spalte 0 in Zeile 10.
  - Ausgabe von 2 Punkten zur Anzeige der maximal einzugebenden Anzahl Zeichen ab Position 30.
  - Ausgabe eines Hintergrund-Blanks zur Begrenzung des Eingabefeldes.
  - Umschalten auf Vordergrund.
  - Tabulation auf den ersten der ausgegebenen Punkte.
  - Eingabe in die Variable R.

```
50 INPUT TAB(0,10),"RABATT",TAB(30,10),'MP',"..",
                                        'SB',"%",'SF','BP',R
```

6.4.1.2 Die Anweisung MAT INPUT

Syntax:

```
MAT INPUT <MVar> [(<NAusdr> [,<NAusdr>])]
                    [,<MVar> [(<NAusdr> [,<NAusdr>])]]^n
```

Mit dieser Anweisung wird die Eingabe aller Elemente einer Matrix (ohne Reihe und Spalte 0) über die Tastatur ermöglicht. Jede Matrixreihe muß einzeln eingegeben werden (d.h. auch mit "CR" abgeschlossen werden), die Werte der einzelnen Elemente einer Reihe werden durch Kommata getrennt.
Als Kennzeichnung einer Matrix-Eingabe erscheint ein "?" am Bildschirm.

Die einzugebenden Matrizen können mit der MAT INPUT-Anweisung durch Angabe der Anzahl Reihen und Spalten erstmals oder redimensioniert werden, bei einer Redimensionierung darf sich die Anzahl ihrer Elemente nicht vergrößern.
Bei der Angabe nur eines numerischen Ausdrucks in Klammern hinter der Matrix-Variablen wird eine Matrix mit 2 Spalten (einschließlich Spalte 0) dimensioniert.

- Beispiel:

  Eingabe von Werten in die Matrix A. Die Matrix wird mit 2 Reihen und 4 Spalten neudimensioniert.

```
:
40 MAT INPUT A(2,4)
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.4.1.3 Die PRINT-Anweisung

Syntax:

$$\left\{\begin{matrix}\text{PRINT}\\;\end{matrix}\right\}[\text{USING}\langle\text{String}\rangle;]\left[\begin{matrix}\langle\text{NAusdr}\rangle\\\langle\text{String}\rangle\\\langle\text{DisFkt}\rangle\end{matrix}\right]\left[\left\{\begin{matrix};\\,\end{matrix}\right\}\left\{\begin{matrix}\langle\text{NAusdr}\rangle\\\langle\text{String}\rangle\\\langle\text{DisFkt}\rangle\end{matrix}\right\}\right]^{n}\left[\begin{matrix};\\,\end{matrix}\right]$$

PRINT ermöglicht die Ausgabe

- des Inhalts von String-Variablen
- des Wertes von numerischen Ausdrücken
- von numerischen und alphanumerischen Konstanten
- von Bildschirm-Funktionen

in beliebiger Reihenfolge am Bildschirm.

PRINT : Schlüsselwort.
; Bei der Eingabe der Anweisung kann statt PRINT auch ein Semikolon (;) angegeben werden.

USING⟨String⟩: Mit dieser Angabe wird eine Aufbereitungsmaske vorgegeben, die die Aufbereitung aller mit der Anweisung auszugebenden numerischen Variablen steuert. Die Maske wird in ⟨String⟩ vorgegeben.

USING darf in einer PRINT-Anweisung nur einmal direkt nach dem Schlüsselwort angegeben werden und muß durch ein Semikolon von den Folgeoperanden getrennt werden.

Der Aufbau von Masken und die Funktionen der zur Verfügung stehenden Maskenzeichen sind im Pkt. "Masken" beschrieben.

Die BASIC-Anweisungen

Im Anschluß daran kann eine beliebige Folge von:

⟨NAusdr⟩ = numerischen Ausdrücken
⟨String⟩ = String-Variablen oder String-Literalen
⟨DisFkt⟩ = Bildschirmfunktionen

angegeben werden, die durch Kommata oder Semikolon zu trennen sind.

Ausgabe numerischer Werte

Wird die Ausgabe von Gleitkomma- oder Festkommazahlen nicht über Masken gesteuert, erfolgt sie ab der aktuellen Cursorposition in folgendem Standardformat:

- Vorzeichen (1 Zeichen), für "+" wird Blank " " und für "-" wird Minus "-" ausgegeben.
- Ziffern des Wertes einschließlich Dezimalpunkt und Exponent (bei Gleitkommazahlen). Die Anzahl dieser Zeichen wird bestimmt durch die Genauigkeit des auszugebenden Wertes, führende Nullen werden unterdrückt.
- An die zuletzt ausgegebene Ziffer wird ein Blank " " angefügt.

Numerische Werte werden (ohne USING-Angabe) im Gleitkommaformat ausgegeben, wenn der Wert

- größer als 99999999999999
- kleiner als 0.1 ist.

• Beispiele: Der Wert 0.01 wird als 1E-02 ausgegeben.

Der Wert 999999999999999 wird als 9.9999999999999E+14 ausgegeben.

Ausgabe von Strings

Strings werden grundsätzlich ab der aktuellen Cursorposition bündig ohne führendes oder abschließendes Blank angezeigt.

Als Trennzeichen zwischen und nach den Operanden dienen "," (Komma) und ";" (Semikolon).

Die BASIC-Anweisungen

Trennzeichen "," (Komma)

Nach der Ausgabe des vorhergehenden Operanden wird bis zur nächsthöheren durch 15 teilbaren Spaltenposition Blanks ausgegeben und der Cursor auf diese Position gesetzt. Ist dieser Wert größer als 79, wird ein Zeilenvorschub durchgeführt.

Trennzeichen ";" (Semikolon)

Nach Bearbeitung des vorhergehenden Operanden bleibt der Cursor auf der aktuellen Position stehen.
Nach Ausgabe des letzten Operanden in der PRINT-Anweisung erfolgt die Cursorpositionierung nach denselben Regeln.
Wird nach dem letzten Operanden weder Komma noch Semikolon codiert, wird der Cursor auf den Beginn der nächsten Zeile positioniert.
Jedem Bildschirmarbeitsplatz ist im System ein Ein-/Ausgabepuffer in Größe von 256 Byte zugeordnet. Ausgaben am Bildschirm erfolgen erst, wenn:

- ein gleichzeitig laufendes Programm seine Zeitscheibe erhält
- der Ausgabepuffer voll ist
- eine PRINT #-Anweisung,
- eine SIGNAL 3-Anweisung,
- eine INPUT-Anweisung,
- eine CHAIN-Anweisung,
- eine LINK-Anweisung ausgeführt wird
- das Programmende erreicht ist.

Erfolgt die Ausgabe eines Zeichens auf Spalte 79 in Zeile 24, wird der gesamte Bildschirminhalt um eine Zeile nach oben geschoben. Die Informationen in Zeile 0 gehen verloren, und Zeile 24 wird frei.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.4.1.4 Die Anweisung MAT PRINT

Syntax:

MAT PRINT ⟨MVar⟩ [,|;]

Die Anweisung MAT PRINT erzeugt das Anzeigen von gesamten Matrizen (ohne Reihe und Spalte 0) am Bildschirm. Die Ausgabe erfolgt reihenweise und innerhalb der Reihen von links nach rechts. Mit einer Anweisung kann nur eine Matrix ausgegeben werden. Die Anweisung arbeitet grundsätzlich mit Dezimalpunkt.

Das abschließende Semikolon oder Komma legt das Format der Trennung zwischen den Elementen der Matrix fest. Ist keines der beiden Zeichen angegeben, wird standardmäßig ein Komma angenommen.

, Die Ausgabe des nächsten Elementes einer Reihe wird bei der nächsten durch 15 teilbaren Position begonnen. Wird dabei 75 überschritten, wird eine neue Bildschirmzeile begonnen.

; Die Ausgabe des nächsten Elementes schließt bündig an das letzte Element an (ein Vorzeichen vor dem Wert und ein Blank nach dem Wert werden dabei berücksichtigt).

6.4.2 Die Anweisungen des logischen IOCS

6.4.2.1 Die OPEN #-Anweisung

Syntax:

OPEN # ⟨NAusdr⟩ [;⟨String1⟩],⟨String2⟩
[[,#⟨NAusdr⟩ [;⟨String1⟩]], ⟨String2⟩]ⁿ

Eröffnen von Magnetplattendateien oder Geräte-Kanalprogrammen (Drivern) auf einem Datenkanal.

OPEN # : Schlüsselwort

⟨NAusdr⟩ : Nummer des Datenkanals, auf dem die Datei eröffnet werden soll. Die Kanäle sind numeriert beginnend bei 0.
Beim Arbeiten unter TAMOS sind die Kanäle 0 und 1 belegt.

⟨String2⟩: Name der Datei oder des Drivers, die eröffnet werden soll. Innerhalb von ⟨String2⟩ muß ⟨DatName⟩ oder ⟨DriverName⟩ stehen. Die Angabe der Kanalnummer ist nur für die erste mit einer "OPEN #"-Anweisung zu eröffnenden Datei erforderlich. Sollen mehrere Dateien mit einer OPEN #-Anweisung auf aufeinanderfolgenden Kanälen eröffnet werden, ist nur die Kanalnummer für die erste Datei anzugeben.

- Beispiel:

```
10 OPEN #2,"$LPT",A$,B$,C$
```

Die Dateien (Driver) ohne führende Kanalnummer-Angabe werden in aufeinanderfolgenden Kanälen eröffnet (im Beispiel $LPT auf Kanal 2, A$ auf Kanal 3, B$ auf Kanal 4 und C$ auf Kanal 5).

⟨String1⟩: Zum Eröffnen von Magnetplattendateien sowie von Druckern und anderen Peripheriegeräten können hier Parameter zur Bearbeitung der jeweils folgenden Datei (des Drivers) festgelegt werden.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

- Plattendateien:

"L" als erstes Zeichen im String bewirkt, daß auf die betreffende Datei nur zugegriffen werden kann, nachdem sie vom derzeitigen Benutzer geschlossen wurde. Eine OPEN #-Anweisung auf eine gesperrte Datei erzeugt:

ERROR # 50 AT ⟨Zlnr⟩ IN ⟨Programmname⟩

"R" als erstes Zeichen im String bewirkt, daß man bei Zugriff auf einen gesperrten Satz in der betreffenden Datei folgende Fehlermeldung erhält:

ERROR # 123 AT ⟨Zlnr⟩ IN ⟨Programmname⟩

Ohne "R" wird das Programm geparkt, bis der gesperrte Satz freigegeben wird.

"C" als erstes Zeichen im String bewirkt, daß eine relative Datei bis zum höchsten geschriebenen Datensatz gelöscht und der EOF-Zeiger auf -1 gesetzt wird. Bei anderen Dateien wird der Parameter "C" ignoriert. Ist die betreffende Datei bereits eröffnet, erscheint die Fehlermeldung

ERROR # 50 AT ⟨Zlnr⟩ IN ⟨Programmname⟩

Die Parameter "L", "R", und "C" können in beliebiger Reihenfolge kombiniert werden.

Die BASIC-Anweisungen

- Lochkartenleserdateien:

  In den ersten beiden Zeichen des Strings kann die Anzahl der Spalten (1-80) der zu verarbeitenden Lochkarten angegeben werden.

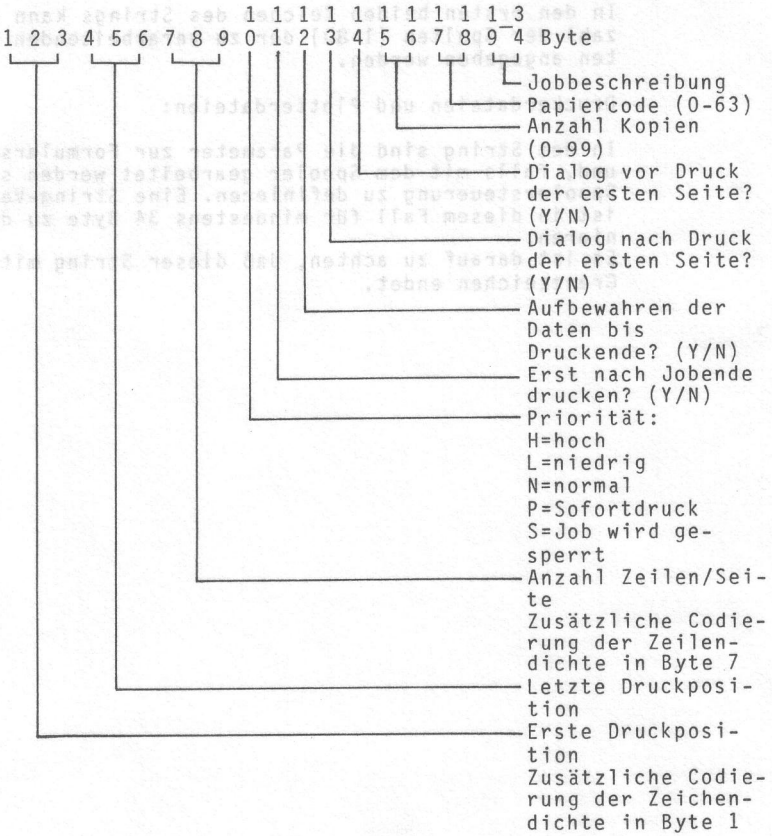
- Druckerdateien und Plotterdateien:

  In dem String sind die Parameter zur Formularsteuerung und, falls mit dem Spooler gearbeitet werden soll, zur Spoolersteuerung zu definieren. Eine String-Variable ist in diesem Fall für mindestens 34 Byte zu dimensionieren.
  Es ist darauf zu achten, daß dieser String mit einem Grenzzeichen endet.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Aufbau:



Bei Nadeldruckern mit zwei Papiervorschüben dürfen sich die Druckbereiche nicht überlappen.

Die Angabe von "Kleinbuchstaben" für die Priorität (h, l, n, p oder s) bewirkt einen Schutz gegen Löschen des Druckauftrages.

Die BASIC-Anweisungen

Werden keine Angaben gemacht, gelten folgende Standardwerte:

Kanalprogramm (Driver)	Erste Druckposition	Letzte Druckposition	Formularhöhe
$LPT	000	176	048
$LPTR	177	177	048
$LPT1	000	135	048
$LPT1S	000	135	048
$LPTS	000	176	048
$LPTRS	177	177	048
$RPLO-15	000	131	048
$RPRO-15	145	177	048
$RPFO-15	000	131	062
$LCP	000	131	048
$LCPS	000	131	048
$ORDP	000	135/*131	048
$ORDPS	000	135/*131	048
$HDD	000	131/*157	048
$HDDS	000	131/*157	048
$HDDSAS	000	131	048
$HDDSASS	000	131	048
$ORDPSR	000	131/*157	048
$ORDPSRS	000	131/*157	048

* einstellbar durch Codetabelle im Driver

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

- Drucker-Drivernamen:

$LPT	Nadeldrucker, linker Vorschub
$LPTR	Nadeldrucker, rechter Vorschub
$LPT1	Zeilendrucker (Schnelldrucker)
$LPT1S	Zweiter Zeilendrucker
$LPT2	Drucker nur für USA
$LPT2S	2. Drucker (nur für USA)
$LPTS	Zweiter Nadeldrucker, linker Vorschub
$LPTRS	Zweiter Nadeldrucker, rechter Vorschub
$RPLO-15	Arbeitsplatzdrucker 0-15, linker Vorschub
$RPRO-15	Arbeitsplatzdrucker 0-15, rechter Vorschub
$RPFO-15	Arbeitsplatzdrucker 0-15, Formulareinzug
$LCP	Kompaktdrucker
$LCPS	Zweiter Kompaktdrucker
$ORDP	Banddrucker
$ORDPS	Zweiter Banddrucker
$HDD	Systemkompaktdrucker
$HDDS	Zweiter Systemkompaktdrucker
$HDDSAS	Systemkompaktdrucker über SAS
$HDDSASS	2. Systemkompaktdrucker über SAS
$ORDPSR	Banddrucker über PSR
$ORDPSRS	2. Banddrucker über PSR

- Übrige Parameter:

Priorität	= N
Erst nach Jobende drucken	= N
Aufbewahren der Daten bis Druckende	= N
Dialog nach Druck der ersten Seite	= N
Dialog vor Druck der ersten Seite	= N
Anzahl Kopien	= 00
Papiercode	= 00

- Jobbeschreibung:

Wird keine Jobbeschreibung vorgegeben, wird bei einem OPEN aus einem BASIC-Programm der Selektorname der 3. Ebene und bei einem OPEN von einem Prozessor (z.B. LIBR) der Prozessorname übernommen.
Diese Angabe wird aus dem Common-Bereich gelesen; sie kann daher von einem Anwenderprogramm überschrieben worden sein.

Als Druckername kann auch ein beliebiger Name eingetragen werden (einzige Bedingung: Das 1. Zeichen muß ein "$"-Zeichen sein; z.B. $LPT). Die Druckerzuordnungstabelle, die mit dem TAMOS-Programm GERAETEZUORDNUNG verändert werden kann, wird auf einen Eintrag für den betreffenden Arbeitsplatz durchsucht. Existiert dort ein Eintrag für den angegebenen Druckernamen, wird der Druckername durch den betreffenden Eintrag (Drucker-Drivername) ersetzt. Die Parameter werden nach folgenden Prioritäten eingesetzt:

Erste/letzte Druckposition	Zeilen pro Seite
1. Druckerzuordnungstabelle	1. OPEN #-Anweisung
2. OPEN #-Anweisung	2. Druckerzuordnungstabelle
3. Standardwerte	3. Standardwerte

- Zeichendichte:

Zum Inhalt von Byte 1 zu addierender Wert	Bedeutung
0	Standardwert (s. Kap. "Die Drucker")
2	10 Zeichen pro Zoll
4	12 Zeichen pro Zoll
6	15 Zeichen pro Zoll
8	16,5 Zeichen pro Zoll

- Beispiel:

  Wert in Byte 1-3 : 500
  Bedeutung : 12 Zeichen pro Zoll und 100 als erste Druckposition

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

- Zeilendichte:

Zum Inhalt von Byte 7 zu addierender Wert	Bedeutung
0	Standardwert= 6 Zeilen pro Zoll
2	Reserviert
4	Reserviert
6	6 Zeilen pro Zoll
8	8 Zeilen pro Zoll

  - Beispiel:

    Wert in Byte 7-9 : 848
    Bedeutung : 8 Zeilen pro Zoll und Blatthöhe = 48

- Lochstreifendateien:

  Der Name einer Code- bzw. Decode-Tabelle für den Lochstreifenstanzer/-Leser kann als String-Variable oder -Literal an dieser Stelle angegeben werden. Näheres hierzu siehe Kapitel "Lochstreifenleser/Lochstreifenstanzer".

6.4.2.2 Die ENV #-Anweisung

Die ENV # - Anweisung benötigen Sie bei der Programmierung der

- Datenübertragung über den 1-Kanal-PLC und zur
- Magnetbandprogrammierung (Peripheriedriver-Schnittstelle).

Sie übergibt für den Betrieb einer Datenübertragungs-Leitung notwendige Angaben zusätzlich zu den

- OPEN #
- WRITE #
- READ #

Anweisungen. Anzahl und Bedeutung der Parameter werden durch die ENV #-Funktionsnummer festgelegt.

Syntax:

ENV #⟨NAusdr⟩;⟨NVar1⟩[,⟨NVar⟩ ,⟨SVar⟩]^5

⟨NAusdr⟩ : Kanalnummer, auf der die Leitung eröffnet wurde.

⟨NVar1⟩ : Funktionsnummer.

⟨NVar⟩ : Parameter, deren Anzahl, Größe und Bedeutung
⟨SVar⟩ bei den einzelnen Funktionen beschrieben wird.

Im Fehlerfall kann mit der ENV #-Anweisung ein Fehlercode übernommen werden, der den aufgetretenen Fehler differenziert auswertet.

Für Test- und Statistikzwecke sind ebenfalls Funktionen vorhanden. Die Unterscheidung der einzelnen Funktionen wird über die Funktionsnummer ⟨NVar1⟩ vorgenommen.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung Ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Übersicht über die ENV #-Funktionen (Magnetband-Schnittstelle)

Funktion	Bedeutung
0	Übergabe der Codetabellen
1	Status-Abfrage
2	Rückspulen
3	Rückspulen und Abschalten
4	Bandmarke schreiben
5	Blöcke vorsetzen
6	Blöcke zurücksetzen
7	Bandmarken vorsetzen
8	Bandmarken zurücksetzen

Eine genaue Beschreibung dieser Funktionen finden Sie im Kapitel "Das Magnetband", die ENV #-Funktionen zur Programmierung des 1-Kanal-PLC sind im Handbuch "Nixdorf 8870/U Datenfernübertragung" aufgeführt.

6.4.2.3 Die READ #-Anweisung

Syntax:

```
READ # <NAusdr1>[,<NAusdr2>[,<NAusdr3>]];
       {<NVar>} [,<NVar>]^n [;]
       {<SVar>} [,<SVar>]
```

Lesen von Daten aus Magnetplattendateien oder Gerätedrivern und Übertragen der gelesenen Daten in beliebigen Variablen.

<NAusdr1> : Nummer des Datenkanals, auf dem die Datei (der Driver) eröffnet wurde.

<NAusdr2> : Satznummer des zu lesenden Satzes relativ zum Dateianfang (Satznummer 0).

Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben.

Ist <NAusdr2> nicht angegeben, wird die Datei sequentiell bearbeitet, ebenso wenn der Wert vom <NAusdr2> = -1 ist.
Ist der Wert von <NAusdr2> = -2, wird der Satz gelesen, der auf diesem Kanal zuletzt gelesen oder geschrieben wurde. Textdateien sind davon ausgenommen.

Bei der Bearbeitung von Peripheriegeräten ist die Angabe von <NAusdr2> bedeutungslos.

<NAusdr3> : Anfangsadresse innerhalb eines Datensatzes. Diese Angabe ist nur dann erlaubt, wenn <NAusdr2> codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld # 0) angegeben.

Bei allen anderen Magnetplattendateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0) anzugeben.

Bei der Bearbeitung von Peripheriegeräten ist die Angabe von <NAusdr3> ohne Bedeutung.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

<NVar> : Zielvariable.
<SVar> In diese Variablen wird der Inhalt des adressierten Datensatzes übertragen.

Eine Überprüfung des Formats der Daten, die gelesen werden sollen, erfolgt nur bei formatierten Dateien. Bei relativen Dateien erfolgt die Übertragung ohne Rücksicht auf Typ und Format der Ziel-Variablen. Aus Textdateien darf nicht in numerische Variablen gelesen werden.

Die Anzahl Zeichen, die in die jeweiligen Ziel-Variablen zu übertragen sind, ist abhängig von:

- dem dimensionierten Format (1-4 Worte) bei numerischen Variablen.
- dem Auftreten eines Grenzzeichens (in Textdateien auch 'CR'-Code) in alphanumerischen Daten.
- dem Erreichen der dimensionierten bzw. durch Indizes angegebenen Länge bei String-Variablen.

Achtung:

Die Übertragung von Zeichen in eine Stringvariable wird immer mit einem Grenzzeichen abgeschlossen, auch wenn es sich um einen Teilstring handelt.

; : Kennzeichen "keine Satzsperre".
Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung adressierte Satz vor dem Zugriff anderer Teilnehmer geschützt (nur bei Plattendateien).
Ist dagegen ein Semikolon codiert, erfolgt keine Satzsperre.

Die Angaben zur Adressierung der zu lesenden Daten (<NAusdr1>, <NAusdr2> und <NAusdr3>) müssen durch Semikolon ";" von den Namen der Ziel-Variablen getrennt sein.

Für Lochkartenleser-Dateien ist nur eine Ziel-Variable, und zwar eine String-Variable, zugelassen, da nur alphanumerische Daten angeliefert werden.

- Beispiele:

  - 450 READ #C;X,Y,A$;

    Anweisung zum sequentiellen Lesen. Die Kanalnummer wird in der Variablen C vorgegeben. Die Daten werden in die Variablen X, Y und A$ übertragen.

  - 450 READ #C;X,Y,A$

    Gleiche Anweisung mit Sperren des gelesenen Datensatzes.

  - 480 READ #C,R,50;A$;

    Direktes Lesen in einer relativen Datei. Die Kanalnummer wird in der Variablen C und die Satznummer in der Variablen R vorgegeben. Die Übertragung in die Ziel-Variable A$ erfolgt ab Byte 50 des adressierten Satzes. Der Satz wird nicht gesperrt.

6.4.2.4 Die WRITE #-Anweisung

Syntax:

```
WRITE # <NAusdr1>[,<NAusdr2>[,<NAusdr3>]];
        {<NAusdr4>} [,<NAusdr4>]^n [;]
        {<String> }  [,<String> ]
```

Übertragen von Daten in Magnetplattendateien und auf Peripheriegeräte über Gerätedriver (bspw. über Drucker- und Leitungsdriver).

<NAusdr1> : Nummer des Datenkanals, auf dem die Datei (der Driver) eröffnet ist.

<NAusdr2> : Satznummer des zu schreibenden Satzes relativ zum Dateianfang (Satz 0).
Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Ist ⟨NAusdr2⟩ nicht angegeben, wird die Datei sequentiell bearbeitet, ebenso wenn der Wert von ⟨NAusdr2⟩ = -1 ist.
Ist der Wert von ⟨NAusdr2⟩ = -2, wird der Satz geschrieben, der auf diesem Kanal zuletzt gelesen oder geschrieben wurde (außer bei Textdateien). Damit wird das Zurückschreiben eines gelesenen und geänderten Datensatzes ermöglicht.
Bei der Bearbeitung von Peripheriegeräten ist die Angabe ⟨NAusdr2⟩ ohne Bedeutung.

⟨NAusdr3⟩ : Anfangsadresse innerhalb des adressierten Datensatzes. Diese Angabe ist nur dann erlaubt, wenn ⟨NAusdr2⟩ angegeben ist.

Bei formatierten Dateien ist ⟨NAusdr3⟩ die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld 0), ab dem die Daten in den adressierten Satz übertragen werden sollen.

Bei allen anderen Plattendateien ist die Adresse des ersten zu beschreibenden Bytes relativ zum Anfang des Datensatzes (Byte 0) anzugeben.

Bei der Bearbeitung von Peripheriegeräten ist ⟨NAusdr3⟩ ohne Bedeutung.

⟨NAusdr4⟩ :
⟨String⟩ - Numerische Ausdrücke,
- String-Variablen und
- String-Literale,
die ausgegeben werden sollen, in beliebiger Anzahl und Reihenfolge. Bei Drucker- und Textdateien ist die Angabe von numerischen Ausdrücken nicht zugelassen.

; : Kennzeichen "keine Satzsperre". Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung adressierte Satz vor dem Zugriff anderer Teilnehmer geschützt, andernfalls erfolgt keine Satzsperre.

Bei der Bearbeitung von Peripheriegeräten ist diese Angabe ohne Bedeutung.
Mit WRITE # ⟨NAusdr1⟩ ;; kann die Satzsperre wieder aufgehoben werden.

Die BASIC-Anweisungen

Eine Überprüfung des Formats der Daten, die übertragen werden sollen, erfolgt nur bei formatierten Dateien. (Nicht, wenn ⟨NAusdr3⟩ = -1)
In jedes Feld ab dem adressierten Feld wird genau ein Ausgabeoperand übertragen.
In allen anderen Fällen werden die Daten identisch übertragen.

Nach String-Literalen und -Variablen ohne Angabe von Indizes wird ein Grenzzeichen abgestellt, bei der Ausgabe von Teilstrings nicht, wenn die Länge des Teilstrings echt kleiner als die dimensionierte Länge des Strings ist. Wird jedoch vor dem Ende des Teilstrings ein Grenzzeichen erreicht, wird auch ein Grenzzeichen abgestellt und die Übertragung beendet.
Wird bei Ausgabe in eine formatierte Datei ein alphanumerisches Feld nicht vollständig beschrieben, wird grundsätzlich nach der Übertragung ein Grenzzeichen abgestellt.

Die Anzahl Zeichen, die in den Datensatz einer relativen oder Indexdatei übertragen wird, wird durch folgende Kriterien bestimmt:

- das dimensionierte Format (1-4 Worte) bei numerischen Variablen, 4 Worte bei numerischen Ausdrücken
- das Auftreten eines Grenzzeichens in Strings
- die dimensionierte bzw. durch Indizes angegebene Länge bei String-Variablen
- die Länge von String-Literalen

Bei der Ausgabe mehrerer Variablen in relative oder Indexdateien ist folgendes zu beachten:

- Bei der Ausgabe von String-Variablen ohne Indizes wird zwar nur bis zum ersten Grenzzeichen übertragen, die Ausgabe des nächsten Operanden beginnt aber erst nach

  dimensionierte Länge der String-Variablen + 1

  Bytes (relativ zum ersten Zeichen der Variablen).
- Numerische Werte werden grundsätzlich ab Beginn eines Wortes übertragen.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Die Angaben zur Adressierung der zu schreibenden Daten (<NAusdr1>, <NAusdr2> und <NAusdr3>) müssen durch Semikolon ";" von den Namen der zu schreibenden Variablen getrennt sein.
Bei Ausgabe auf Drucker bewirkt die Ausgabe eines Grenzzeichens als erstes Zeichen, daß der Druckpuffer erst ganz geleert wird, bevor der Kanal geschlossen werden kann. (WRITE # C;"";)

- Beispiele:

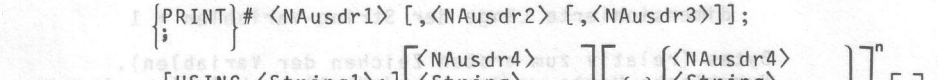
- 450 WRITE #C,R+S,5;A;

  Direktes Schreiben des Inhalts der Variablen A in Feld 5 einer formatierten Datei. Die Kanalnummer wird in der Variablen C vorgegeben. Die Satznummer berechnet sich aus R+S. Der Satz wird nicht gesperrt.

- :
```
310 READ #C;X,Y,A$
320 GOSUB 2000              /* DATEN VERÄNDERN
330 WRITE #C,-2;X,Y,A$;
340 GOTO 310
```
  Sequentielles Lesen, Verändern und Zurückschreiben in einer relativen Datei. Die Kanalnummer wird in der Variablen C vorgegeben.
  Die Variablen X, Y und A$ werden bearbeitet. Nach dem Lesen wird der betreffende Satz gesperrt und erst nach dem Schreiben wieder freigegeben.

6.4.2.5 Die PRINT #-Anweisung

Syntax



Übertragen von Daten in Magnetplattendateien oder Peripheriegeräte über Gerätedriver (bspw. Drucker). Numerische Werte werden im Gegensatz zur (schnelleren) WRITE #-Anweisung mit dieser Anweisung in alphanumerische Daten (ASCII-Code) umgewandelt.

Die BASIC-Anweisungen

Das Schlüsselwort PRINT kann bei der Eingabe durch ; ersetzt werden. (Ausnahme: bedingte Anweisung)

⟨NAusdr1⟩ : Nummer des Datenkanals, auf dem die Datei eröffnet ist.
Ist der angegebene Kanal nicht eröffnet, erfolgen sämtliche Ausgaben ohne Fehlermeldung auf dem Bildschirm.

⟨NAusdr2⟩ : Satznummer des Datensatzes relativ zum Dateianfang (Satz 0), in den geschrieben werden soll.
Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben.
Ist ⟨NAusdr2⟩ nicht angegeben, wird die Datei sequentiell bearbeitet, ebenso wenn der Wert von ⟨NAusdr2⟩ = -1 ist.
Ist der Wert von ⟨NAusdr2⟩ = -2, wird in den Satz geschrieben, der in dieser Datei zuletzt gelesen oder geschrieben wurde (ausgenommen Textdateien).
Bei der Bearbeitung von Peripheriegeräten ist ⟨NAusdr2⟩ ohne Bedeutung.

⟨NAusdr3⟩ : Anfangsadresse innerhalb des adressierten Datensatzes. Diese Angabe ist nur dann erlaubt, wenn ⟨NAusdr2⟩ codiert ist.

Bei formatierten Dateien wird mit ⟨NAusdr3⟩ die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld 0) angegeben, ab dem die Übertragung der Daten beginnen soll.
In formatierten Dateien können nur alphanumerische Felder bearbeitet werden.
Bei allen anderen Dateien ist ⟨NAusdr3⟩ die Adresse des Bytes relativ zum Anfang des Datensatzes (Byte 0), ab dem die Daten geschrieben werden.
Es ist zu beachten, daß mit PRINT # grundsätzlich ASCII-Code ausgegeben wird, numerische Werte werden vor der Ausgabe automatisch in ASCII-Code umgewandelt.
Beim direkten Schreiben mit PRINT # dürfen nur Strings mit einer Stringlänge bis zu 250 Zeichen angegeben werden. Für längere Strings ist eine entsprechende WRITE #-Anweisung zu verwenden.
Bei der Bearbeitung von Peripheriegeräten ist ⟨NAusdr3⟩ ohne Bedeutung.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

USING ⟨String1⟩ : Mit dieser Angabe wird eine Aufbereitungsmaske vorgegeben, die die Umwandlung aller mit dieser Anweisung auszugebenden numerischen Werte steuert. Die Maske wird in ⟨String⟩ angegeben.

Die USING-Angabe darf in einer PRINT #-Anweisung nur einmal vorkommen und muß vom Folgeoperanden durch ein Semikolon ";" getrennt sein.
Der Aufbau von Masken und die Funktionen der zur Verfügung stehenden Maskenzeichen sind im Pkt. "Masken" beschrieben.

⟨NAusdr4⟩ : - Numerische Ausdrücke,
⟨String⟩ - String-Literale und String-Variable, deren Wert bzw. Inhalt ausgegeben werden soll, in beliebiger Reihenfolge.

TAB(⟨NAusdr⟩) : Das System führt (intern) pro Datenkanal einen Zeiger, der die Anzahl ausgegebener Zeichen seit dem letzten 'CR'-Code (215 oktal) bzw. seit der Eröffnung der Ausgabedatei beinhaltet. Der Zeiger weist also auf das zuletzt beschriebene Byte + 1. Die Angabe TAB(⟨NAusdr⟩) erzeugt die Ausgabe von Blanks ab dieser Position bis zur Position ⟨NAusdr⟩ - 1 und stellt im Zeiger den Wert von ⟨NAusdr⟩ ab. Der Wert von ⟨NAusdr⟩ darf nicht kleiner als die aktuelle Zeigerposition sein!
Der TAB-Zeiger wird auf 0 gesetzt, wenn:

- ein 'CR'-Code auftritt,
- eine Anweisung komplett abgearbeitet und hinter dem letzten Operanden kein Semikolon codiert ist (dann erzeugt PRINT # den 'CR'-Code).

Die Angaben zur Adressierung der zu schreibenden Daten (⟨NAusdr1⟩, ⟨NAusdr2⟩ und ⟨NAusdr3⟩) müssen durch Semikolon ";" von dem Folgeoperanden getrennt sein.

Die Operanden, deren Inhalte ausgegeben werden sollen, müssen durch Kommata (,) oder Semikolon (;) voneinander getrennt sein.

Die BASIC-Anweisungen

<DevFkt> : An dieser Stelle können Gerätefunktionen zur Steuerung der Peripheriegeräte angegeben werden, z.B. 'ELO' für ein verbreitertes Schriftbild. Weitere Gerätefunktionen sind unter dem Kapitel "Formularsteuerung" aufgeführt.

Trennzeichen ",": Nach Ausgabe des Inhalts der vorhergehenden Variablen werden soviele Blanks ausgegeben, bis die aktuelle TAB-Position ohne Rest durch 15 teilbar ist.

Trennzeichen ";": Die Ausgabe des Inhalts der auf ";" folgenden Variablen erfolgt bündig zur letzten Ausgabe.

Die Anweisung PRINT # setzt für Magnetplattendateien immer eine Satzsperre für den geschriebenen Datensatz und ein Grenzzeichen hinter das letzte übertragene Zeichen.
Bei Druckern bewirkt die Ausgabe eines Grenzzeichens als erstes Zeichen, daß der Druckpuffer erst ganz geleert wird, bevor der Kanal geschlossen werden kann. (PRINT # C;"";)

letztes { ; , } : PRINT # stellt grundsätzlich hinter dem letzten übertragenen Ausgabeoperanden ein Grenzzeichen ab. Ist nach den letzten Operanden kein Zeichen codiert, wird zusätzlich vor dem Grenzzeichen ein Zeilenvorschub-Code (215 oktal) ausgegeben. Ist ein Semikolon codiert, entfällt diese Ausgabe. Wird ein Komma angegeben, werden vor dem Grenzzeichen Blanks bis vor die nächste durch 15 teilbare Position ausgegeben.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

Es ist zu beachten, daß mit PRINT #

- grundsätzlich ASCII-Code
- nach dem letzten Ausgabeoperanden grundsätzlich ein Grenzzeichen

ausgegeben wird. Numerische Werte werden vor der Ausgabe in ASCII-Code umgewandelt.
Ohne USING-Angabe werden Gleitkomma- und Festkommandozahlen über folgende Standardmaske aufbereitet:

- Vorzeichen (1 Zeichen), für "+" wird Blank " " und für "-" wird Minus "-" ausgegeben.
- Im Anschluß an das Vorzeichen werden die Ziffern, Dezimalpunkt und Exponent (bei Gleitkommazahlen) ausgegeben. Führende Nullen werden unterdrückt.
- Im Anschluß an die zuletzt ausgegebene Ziffer wird ein Blank ausgegeben.

Ohne USING-Angabe wird eine Zahl im Gleitkommaformat ausgegeben, wenn ihr Wert

- größer als 99999999999999 oder
- kleiner als 0.1 ist.

● Beispiel:

```
490 PRINT #C;USING "####";TAB(10);K$;TAB(70);"SEITE:
";A
```

Ausgabe des Inhalts der Variablen K$ auf Position 10. Ausgabe des Textes "SEITE:" ab Position 70 und anschließend Übertragen des Inhalts der Variablen A in die Maske ####. Abschließend wird ein 'CR'-Code (215 oktal) ausgegeben. Die Ausgabe erfolgt in die Datei, die auf dem in der Variablen C angegebenen Kanal eröffnet ist.

Für die Druckausgabe sollten PRINT #-Anweisungen möglichst ohne TAB-Angaben programmiert werden, da innerhalb dieser Anweisung auftretende Fehler nicht über SPC 8 und RETURN-1 abgefangen werden, ohne Druckanfangspositionen zu verschieben.

6.4.2.6 Die MAT READ #-Anweisung

Syntax:

```
MAT READ # <NAusdr1> [,<NAusdr2> [,<NAusdr3>]];
                                        <VarName> [;]
```

Lesen von Daten aus formatierten, relativen und Indexdateien und Übertragen der gelesenen Daten in

- eine Matrix
- einen Vektor
- eine numerische Variable
- eine String-Variable

<NAusdr1> : Nummer des Datenkanals, auf dem die Datei eröffnet wurde.

<NAusdr2> : Satznummer des zu lesenden Satzes relativ zum Dateianfang (Satznummer 0).

Ist <NAusdr2> nicht angegeben, wird die Datei sequentiell bearbeitet, ebenso wenn der Wert von <NAusdr2> = - 1 ist.
Ist der Wert von <NAusdr2> = -2, wird der Satz gelesen, der in dieser Datei zuletzt gelesen oder geschrieben wurde.

<NAusdr3> : Anfangsadresse innerhalb eines Datensatzes. Diese Angabe ist nur dann erlaubt, wenn <NAusdr2> codiert ist.
Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld 0) angegeben.Die Angabe von -1 als Feldnummer bei formatierten Dateien ermöglicht die Bearbeitung ohne Prüfung der Feldformate. Die Übertragung beginnt dann mit dem ersten Byte des adressierten Satzes und endet mit dem letzten oder bei Erreichen der dimensionierten Länge der Zielvariablen.

Bei allen anderen Magnetplattendateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte 0) anzugeben. Bei einer ungeraden Byte-Adresse wird wegen der wortorientierten Übertragung (1 Wort = 2 Byte) intern zur nächsthöheren geradzahligen Adresse erhöht.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

〈VarName〉 : Numerische Variable, Matrix, Vektor oder Stringvariable, in die der adressierte Datensatz übertragen werden soll. Es darf nur ein Variablenname ohne Indizes vorgegeben werden. Die Varaible muß schon dimensioniert sein. Die Übertragung der Daten erfolgt bis:

- das Ende der aufnehmenden Variablen oder
- das Ende der Datei erreicht ist.

Strings werden also einschließlich eventueller Grenzzeichen übertragen.
Bei der Bearbeitung eines Vektors oder einer Matrix wird die Variable in ihrer Gesamtheit einschließlich Reihe und Spalte 0 übertragen.

; : Kennzeichen "keine Satzsperre". Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung adressierte Satz vor dem Zugriff anderer Teilnehmer geschützt. Ist dagegen ein Semikolon codiert, erfolgt keine Satzsperre. Die Satzsperre wird nur für den adressierten Satz wirksam, auch dann, wenn mit einer Anweisung mehrere Datensätze gelesen werden.

Die Angaben zur Adressierung der zu lesenden Daten (〈NAusdr1〉, 〈NAusdr2〉 und 〈NAusdr3〉) müssen durch Semikolon von dem Namen der Ziel-Variablen getrennt sein.

Die BASIC-Anweisungen

- Beispiele:

  - 20 DIM A(3,3)
    :
    50 MAT READ #C,R;A;

    Lesen der Matrix A in ihrer Gesamtheit aus dem Satz, dessen Nummer in der Variablen R abgestellt ist. Die Nummer des Kanals, auf dem die Datei eröffnet ist, steht in der Variablen C.

  - 20 DIM A$(512)
    :
    50 MAT READ #C;A$;

    Übertragen von 512 Byte in die Variable A$. Die Daten werden aus dem Satz gelesen, der dem zuletzt bearbeiteten Satz dieser Datei folgt. Die Nummer des Kanals, auf dem die Datei eröffnet wurde, steht in der Variablen C.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.4.2.7 Die MAT WRITE #-Anweisung

Syntax:

MAT WRITE # ⟨NAusdr1⟩[,⟨NAusdr2⟩[,⟨NAusdr3⟩]];⟨VarName⟩[;]

Übertragen von Daten aus einer String-Variablen, einer numerischen Variablen, einem Vektor oder einer Matrix in eine Magnetplattendatei (ausgenommen Textdatei).

⟨NAusdr1⟩ : Nummer des Datenkanals, auf dem die Datei eröffnet wurde.

⟨NAusdr2⟩ : Satznummer des zu schreibenden Datensatzes relativ zum Dateianfang (Satznummer 0).

Ist ⟨NAusdr2) nicht angegeben, wird die Datei sequentiell bearbeitet, ebenso wenn der Wert von ⟨NAusdr2⟩ = -1 ist.

Ist der Wert von ⟨NAusdr2⟩ = -2, wird der Satz geschrieben, der auf diesem Kanal zuletzt gelesen oder geschrieben wurde.

⟨NAusdr3⟩ : Anfangsadresse innerhalb eines Datensatzes. Diese Angabe ist nur dann erlaubt, wenn ⟨NAusdr2⟩ codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld 0) angegeben. Die Angabe von -1 als Feldnummer bei formatierten Dateien ermöglicht eine Bearbeitung ohne Prüfung der Feldformate. Die Übertragung beginnt dann beim ersten Byte des adressierten Satzes.

Bei allen anderen Magnetplattendateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte 0) anzugeben. Bei einer ungeraden Byte-Adresse wird wegen der wortorientierten Übertragung (1 Wort = 2 Byte) intern zur nächsthöheren geradzahligen Adresse erhöht.

<VarName> : Numerische Variable, Matrix, Vektor oder String-Variable, deren Inhalt in den adressierten Datensatz übertragen werden soll.

Es darf nur ein Variablenname ohne Indizes angegeben werden. Die Variable muß schon dimensioniert sein.

; : Kennzeichen "keine Satzsperre".
Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung adressierte Satz vor dem Zugriff anderer Teilnehmer geschützt.
Ist dagegen ein Semikolon codiert, erfolgt keine Satzsperre.
Die Satzsperre wird nur für den adressierten Satz wirksam, auch dann, wenn mit einer Anweisung mehrere Datensätze geschrieben werden.

Die Übertragung der Daten erfolgt einschließlich eventueller Grenzzeichen bis:

- die Variable in ihrer dimensionierten Länge ausgegeben ist
- das Dateiende erreicht ist (nur bei relativen Dateien)
- die Plattenkapazität erschöpft ist

Nach der Übertragung von Stringvariablen wird kein Grenzzeichen gesetzt, wenn die dimensionierte Länge der Stringvariablen geradzahlig ist. Da die Übertragung bei MAT WRITE wortorientiert ist, wird bei ungeradzahliger Dimensionierung ein Grenzzeichen ergänzt.
Bei der Bearbeitung eines Vektors oder einer Matrix wird auch das Element 0 bzw. Reihe und Spalte 0 übertragen.
Es können mehrere Sätze von der übertragenen Variablen belegt werden.

Die Angaben zur Adressierung der zu schreibenden Daten (<NAusdr1>, <NAusdr2> und <NAusdr3>) müssen durch Semikolon von dem Namen der Variablen getrennt sein.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

- Beispiele:

```
- 20 DIM A$(512)
   :
   50 MAT WRITE #C,-2;A$;
```

Zurückschreiben von 512 Byte aus der Variablen A$ ab Byte 0 des zuletzt in dieser Datei bearbeiteten Satzes. Die Nummer des Kanals, auf dem die Datei eröffnet ist, ist in der Variablen C abgestellt.

```
- 20 DIM A(3,3)
   :
   50 MAT WRITE #C,0,X;A;
```

Schreiben der Matrix A in ihrer Gesamtheit ab der in der Variablen X angegebenen Byte-Adresse in Satz 0. Die Nummer des Kanals, auf dem die Datei eröffnet wurde, ist in der Variablen C abgestellt.

6.4.2.8 Die SEARCH #-Anweisung

Bearbeiten von Schlüsselverzeichnissen in einer Indexdatei.

Syntax:

SEARCH # ⟨NAusdr1⟩,⟨NAusdr2⟩,⟨NAusdr3⟩;
⟨SVar⟩,⟨NVar1⟩,⟨NVar2⟩[:|;]

⟨NAusdr1⟩ : Nummer des Datenkanals, auf dem die Datei eröffnet ist.

⟨NAusdr2⟩ : Modus, definiert die auszuführende Funktion.

⟨NAusdr3⟩ : Nummer des zu bearbeitenden Schlüsselverzeichnisses (1-15). Die Angabe 0 ist dann zugelassen, wenn ⟨NAusdr2⟩ = 0 oder 1 ist.

⟨SVar⟩ : String-Variable, in der der Ordnungsbegriff vorgegeben bzw. vom IOCS übergeben wird. ⟨SVar⟩ muß mindestens wie die Länge des Ordnungsbegriffs dimensioniert sein.

⟨NVar1⟩ : Numerische Variable, mindestens 2%, die zur Übergabe der relativen Satznummer an das Anwenderprogramm bzw. an das IOCS dient.

⟨NVar2⟩ : Numerische Variable zur Parameterübergabe an das Betriebssystem und zur Übergabe von Statusmeldungen an das Anwenderprogramm.

: : Setzen einer Satzsperre bei Modus 1,2,3 u. 5.

; : Keine Satzsperre (ebenso ohne ";" oder ":").

Die Funktionen von SEARCH #

⟨NAusdr2⟩	Funktion
0	⟨NAusdr3⟩ ≠ 0: Definieren des Verzeichnisses ⟨NAusdr3⟩ mit der in ⟨NVar1⟩ vorgegebenen Ordnungsbegrifflänge. Die Verzeichnisnummern müssen aufsteigend mit 1 beginnend in ⟨NAusdr3⟩ angegeben werden. ⟨NAusdr3⟩ = 0: Initialisieren sämtlicher Verzeichnisse, die mit SEARCH-Modus 0, ⟨NAusdr3⟩ ≠ 0 angelegt wurden. In ⟨NVar1⟩ ist die Anzahl Datensätze zu übergeben.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

⟨NAusdr2⟩	Funktion
1	⟨NAusdr3⟩ ≠ 0: Diese Funktion liefert in ⟨NVar1⟩ die Ordnungsbegrifflänge des unter ⟨NAusdr3⟩ angegebenen Verzeichnisses. Ist das Verzeichnis nicht vorhanden, wird ⟨NVar2⟩ auf 5 gesetzt. ⟨NAusdr3⟩ = 0 und ⟨NVar2⟩ = 0: Diese Funktion übergibt in ⟨NVar1⟩ die relative Satznummer des ersten Datensatzes im Datenbereich. ⟨NAusdr3⟩ = 0 und ⟨NVar2⟩ = 1: Diese Funktion übergibt in ⟨NVar1⟩ die Anzahl freier Datensätze. ⟨NAusdr3⟩ = 0 und ⟨NVar2⟩ = 2: Diese Funktion übergibt in ⟨NVar1⟩ die relative Satznummer des ersten freien Datensatzes, löscht diesen Satz aus der Liste der freien Datensätze und setzt eine Satzsperre bei Codierung von ":". ⟨NAusdr3⟩ = 0 und ⟨NVar2⟩ = 3: Der in ⟨NVar1⟩ angesprochene Datensatz wird in die Liste der freien Datensätze eingetragen. Der Datensatz wird erst dann freigegeben, wenn kein anderer Teilnehmer mehr den Satz gesperrt hat.
2	Diese Funktion sucht in dem angegebenen Verzeichnis den unter ⟨SVar⟩ angegebenen Ordnungsbegriff. Die Anzahl Zeichen, die verglichen werden, wird von der aktuellen Länge von ⟨SVar⟩ bestimmt (diese darf also die Ordnungsbegrifflänge nicht überschreiten). Gleichheit wird erkannt, wenn alle verglichenen Zeichen von ⟨SVar⟩ mit dem Ordnungsbegriff übereinstimmen. Gleichheit kann somit auch dann erkannt werden, wenn der in ⟨SVar⟩ vorgegebene Ordnungsbegriff kürzer ist als der im Verzeichnis stehende. Das Verfahren, nur bis zu dem ersten auftretenden Grenzzeichen zu vergleichen, bietet die Möglichkeit, Teilordnungsbegriffe vorzugeben. Bei Gleichheit wird der gesamte Ordnungsbegriff aus dem Verzeichnis in ⟨SVar⟩ übergeben. In ⟨NVar1⟩ wird die relative Satznummer des dazugehörigen Datensatzes übergeben und der Satz gegebenenfalls gesperrt. Wird keine Übereinstimmung festgestellt, bleiben ⟨SVar⟩ und ⟨NVar1⟩ unverändert, und in ⟨NVar2⟩ wird Status 1 übergeben.

⟨NAusdr2⟩	Funktion
3	Im angegebenen Verzeichnis wird der nächstgrößere Ordnungsbegriff gegenüber dem in ⟨SVar⟩ vorgegebenen gesucht. Wird ein größerer Ordnungsbegriff gefunden, wird dieser in ⟨SVar⟩ übergeben. In ⟨NVar1⟩ wird die relative Satznummer des dazugehörigen Datensatzes übergeben und der Satz wird bei Codierung von ":" gesperrt. Ist kein größerer Ordnungsbegriff vorhanden bleiben ⟨SVar⟩ und ⟨NVar1⟩ unverändert. Als Statusmeldung wird in ⟨NVar2⟩ der Wert 2 übergeben.
4	Diese Funktion fügt den in ⟨SVar⟩ angegebenen Ordnungsbegriff in das angegebene Verzeichnis ein, gegebenenfalls wird das Verzeichnis dabei reorganisiert. Der Ordnungsbegriff darf in dem angegebenen Verzeichnis noch nicht vorhanden sein. Gleichheit wird erkannt, wenn ⟨SVar⟩ einschließlich Grenzzeichen identisch mit dem Ordnungsbegriff im Verzeichnis ist. Die relative Satznummer des Datensatzes, auf den der Ordnungsbegriff verweisen soll, muß in ⟨NVar1⟩ angegeben werden. Ist der Ordnungsbegriff im angegebenen Verzeichnis bereits vorhanden, werden in ⟨NVar1⟩ die Satznummer des zugehörigen Datensatzes und in ⟨NVar2⟩ der Status 1 übergeben.
5	Der in ⟨SVar⟩ vorgegebene Ordnungsbegriff wird aus dem angegebenen Verzeichnis gelöscht. In ⟨NVar1⟩ wird die relative Satznummer des dazugehörigen Datensatzes übergeben, der mit SEARCH-Modus 1 gesondert freigegeben werden muß. Das Verfahren zum Vergleichen der Ordnungsbegriffe entspricht dem beim Suchen eines Ordnungsbegriffs. Ist der vorgegebene Ordnungsbegriff nicht im Verzeichnis vorhanden, bleiben ⟨SVar⟩ und ⟨NVar1⟩ unverändert, und in ⟨NVar2⟩ wird der Status 1 übergeben.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

⟨NAusdr2⟩	Funktion
6	Die Verzeichnisse werden gelöscht und die Datensätze in die Liste der freien Sätze eingetragen, d.h. sie werden neu verkettet. Nach Durchführung dieser Anweisung hat die Datei den gleichen Zustand wie nach der Neuanlage. Der Inhalt von ⟨NAusdr3⟩, ⟨SVar⟩, ⟨NVar1⟩ und ⟨NVar2⟩ ist bei dieser Funktion ohne Bedeutung. Die Funktion gibt keine Zeit an andere Teilnehmer ab. Ist die Datei von einem anderen Teilnehmer eröffnet, wird Status 15 gemeldet. Nach erfolgreicher Durchführung der Funktion wird in ⟨NVar1⟩ die Anzahl der freien Datensätze übergeben.
8	Modus 8 wirkt wie Modus 3. Wird beim OPEN durch Angabe von "R" definiert, daß eine Satzsperre ausgewertet werden soll, und wird ein gesperrter Satz angesprochen, so wird der BASIC-Fehler #123 gemeldet. In ⟨SVar⟩ werden der nächstgrößere Ordnungsbegriff und in ⟨NVar1⟩ die dazugehörige Satznummer ausgegeben. Die Statusvariable ⟨NVar2⟩ wird auf "0" gesetzt.

Die BASIC-Anweisungen

Statusmeldungen von SEARCH #

Nach Durchführung einer jeden SEARCH #-Anweisung ist die Statusvariable ⟨NVar2⟩ unbedingt abzuprüfen.
Folgende Statusmeldungen können nach Durchführung der verschiedenen Funktionen von SEARCH # in ⟨NVar2⟩ abgestellt sein:

⟨NVar2⟩	Bedeutung
0	Funktion konnte fehlerfrei durchgeführt werden.
1	Funktion konnte nicht erfolgreich durchgeführt werden.
2	Ende des Verzeichnisses erreicht.
3	Kein freier Datensatz vorhanden.
4	Keine Indexdatei.
5	Satznummer negativ, Modus > 8, ⟨NVar2⟩ > 3, Verzeichnisnummer zu groß, Schlüsselfeldlänge zu klein oder anderer schwerer Fehler. Die Datei ist möglicherweise zerstört.
6	Unzulässige Reihenfolge bei Angabe der Verzeichnisse (nur Modus 0).
7	Datei ist keine relative Datei (nur Modus 0).
8	Modus 0, ⟨NAusdr3⟩ = 0 wurde bereits ausgeführt.
9	Kein Verzeichnis angegeben.
10	Zu viele Verzeichnisse (max. 15 pro Datei).
12	Datei ist zu klein, um alle Verzeichnisse aufzunehmen (nur Modus 0, ⟨NAusdr3⟩ = 0).
15	Datei von anderem Teilnehmer eröffnet.

© Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten.

Die BASIC-Anweisungen

Regeln für die Indexdatei-Programmierung

- Soll ein Datensatz mit Ordnungsbegriffen in mehreren Verzeichnissen eingefügt werden, ist zuerst der Datensatz zu schreiben. Danach sind alle auf den Datensatz verweisenden Ordnungsbegriffe in die Verzeichnisse einzutragen.
- Ist ein Datensatz mit Ordnungsbegriffen in mehreren Verzeichnissen verkettet, sind zuerst alle auf den Datensatz verweisenden Ordnungsbegriffe aus den Verzeichnissen zu löschen. Anschließend kann der Datensatz selbst freigegeben werden.
- Nach Durchführung einer SEARCH #-Anweisung grundsätzlich den Status (<NVar2>) abfragen.
- Der Ordnungsbegriff sollte auch im Datensatz vorhanden sein. Er kann gegen den vorgegebenen Ordnungsbegriff verglichen werden und bei Verlust des Indexbereichs zur Rekonstruktion herangezogen werden.

Zum Bearbeiten der Datensätze werden die Anweisungen zur Programmierung relativer Dateien verwendet.

● Beispiel:

Direktes Lesen eines Datensatzes über einen Ordnungsbegriff. Der Datensatz wird gesperrt. Der gesuchte Begriff ist "AA". Es werden folgende Variablennamen verwendet:

C = Kanalnummer
V = Verzeichnisnummer
V$ = Ordnungsbegriff
N = Relative Satznummer
S = Parameter und Statusvariable

Der Modus von SEARCH # wird in dem Beispiel als numerisches Literal angegeben. Die Behandlung von Statusmeldungen ist hierbei nur angedeutet. Es ist selbstverständlich, daß in Anwenderprogrammen auf Statusmeldungen gezielt reagiert werden soll.

```
:
9000 LET V$="AA"
9010 SEARCH #C,2,V;V$,N,S:        /* BEGRIFF SUCHEN
9020 IF S GOTO 9500               /* STATUS ≠ 0
9030 READ #C,N;R$                 /* DATENSATZ LESEN
:
9500 IF S=1 PRINT "SATZ NICHT VORHANDEN";
:
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.4.2.9 Die CLOSE #-Anweisung

Syntax:

CLOSE # ⟨NAusdr⟩ [,#⟨NAusdr⟩]ⁿ

Schließen von eröffneten Dateien und Freigeben des Datenkanals.

Mit ⟨NAusdr⟩ wird die Nummer des Datenkanals angegeben, auf dem die zu schließende Datei eröffnet ist.
Mit einer CLOSE #-Anweisung können beliebig viele Dateien durch die Angabe ihrer Datenkanalnummern geschlossen werden.

Ist unter der angegebenen Kanalnummer keine Datei eröffnet, wird BASIC-Fehler 49 gemeldet.

Anmerkung:

Tritt beim Schließen mehrerer Dateien (Kanäle) mit einer CLOSE #-Anweisung ein Fehler auf, kann nicht festgestellt werden, welche Datei den Fehler verursacht hat.

- Beispiel:

```
:
490 IF ERR 0 GOTO ...    /* FEHLERAUSW. 49
500 FOR I=2 TO 13
510 CLOSE #I
520 NEXT I
530 IF ERR 0 GOSUB ...   /* STANDARDFEHLERBEHANDL.
```

Routine zum Schließen der Kanäle 2 bis 13. Die von TAMOS belegten Kanäle 0 und 1 werden nicht geschlossen.

Die BASIC-Anweisungen

6.4.2.10 Die BUILD #-Anweisung

Syntax:

BUILD # ⟨NAusdr⟩,[+] ⟨String⟩ [[,#⟨NAusdr⟩],[+] ⟨String⟩]ⁿ

Die Anweisung erstellt neue Dateien bzw. ersetzt vorhandene Dateien auf der Magnetplatte. Es wird der Datei-Headerblock angelegt, gegebenenfalls ein Bereich auf der Magnetplatte reserviert und der Dateiname ins Platteninhaltsverzeichnis (INDEX) eingetragen.
Die reservierten Plattenblöcke werden in der Platenbelegungsliste als als belegt eingetragen.
Gleichzeitig mit dem Erstellen wird die Datei eröffnet und gegen den Zugriff anderer Teilnehmer gesperrt.

Die mit BUILD # erstellte und eröffnete Datei muß nach der Bearbeitung unbedingt mit CLOSE # geschlossen werden, da andernfalls die Datei wieder gelöscht wird.

⟨NAusdr⟩ : Die Nummer des Kanals, auf dem die Datei erstellt werden soll.

\+ : "+" kennzeichnet die zu erstellende Datei als Text-Datei.

⟨String⟩ : String, der die zu erstellende Datei einschließlich Dateiparametern enthält.

Der String hat folgenden Aufbau:

"[(⟨Ganzzahl1⟩ {:|?} ⟨Ganzzahl2⟩)] ⟨Datei⟩ [*|!|#]"

⟨Ganzzahl1⟩ = Anzahl Datensätze, für die ein Plattenbereich reserviert werden soll.

⟨Ganzzahl2⟩ = Satzlänge in Worten (1 Wort = 2 Byte).

Diese beiden Angaben stehen in Klammern und sind durch ":" oder "?" voneinander getrennt. Durch diese Angabe wird die zu erstellende Datei als relative Datei gekennzeichnet. Bei Codierung eines "?" wird das Löschen des Datenbereichs beim Anlegen der relativen Datei verhindert.

⟨Datei⟩ : Der Name, unter dem die Datei auf der Platte erstellt werden soll bzw. der Name der Datei, die ersetzt werden soll.

Die BASIC-Anweisungen

! : Muß angegeben werden, wenn eine bereits bestehende Datei ersetzt werden soll.
Liegt die zu ersetzende Datei auf der ersten ILU und wurde sie mit "*" angelegt, wird die Datei unbedingt auf der ersten ILU angelegt.

* : Die Datei wird auf der ersten ILU angelegt.

: Die Datei wird auch dann auf der ersten ILU angelegt, wenn sie bereits vorhanden ist und auf der zweiten ILU liegt.

Anmerkungen:

Ist nicht genug freier Bereich auf der ersten ILU, wird der BASIC-Fehler 47 gemeldet.

Soll eine Datei ersetzt werden, wird diese erst dann gelöscht, wenn die neue Datei (die sie ersetzt) vorher angelegt wurde.

Werden mit einer BUILD #-Anweisung mehrere Dateien erstellt und sollen die Dateien auf aufeinanderfolgenden Kanälen erstellt werden, ist nur die Kanalnummer für die erste zu erstellende Datei anzugeben. Die Dateien, deren Bezeichnung keine Kanalnummer vorausgeht, werden auf aufeinanderfolgenden Kanälen angelegt.

Mit BUILD # erstellte Dateien sind gleichzeitig für die Verarbeitung eröffnet. Bis zum Schließen mit der CLOSE #-Anweisung sind sie vor dem Zugriff anderer Teilnehmer geschützt.

Eine mit BUILD # angelegte Datei wird gelöscht, wenn das Programm beendet wird und der Kanal, auf dem die Datei erstellt wurde, nicht vorher durch die Anweisung CLOSE # geschlossen wurde.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

- Beispiele:

  - 50 BUILD #2,"(100:50)9/TEST"

    Erstellen einer relativen Datei auf Kanal 2. Alle Parameter sind als ⟨SLit⟩ in der Anweisung angegeben:

    - Anzahl Sätze : 100
    - Satzlänge : 50 (Worte)
    - LU-Nummer : 9
    - Dateiname : TEST
    - Neuanlage

  - 50 BUILD #2,+A$,"(100:50)TEST!",B$,#6,C$

    Erstellen mehrerer Dateien mit einer BUILD #-Anweisung:

    - Erstellen/Ersetzen einer Text-Datei auf Kanal 2, deren Bezeichnung in der Variablen A$ abgestellt ist.
    - Ersetzen der relativen Datei TEST auf Kanal 3. Die Dateibezeichnung ist als ⟨SLit⟩ angegeben.
    - Erstellen/Ersetzen einer Datei, deren Dateibezeichnung in der Variablen B$ abgestellt ist, auf Kanal 4.
    - Erstellen/Ersetzen einer Datei, deren Dateibezeichnung in der Variablen C$ abgestellt ist, auf Kanal 6.

  - :

```
40 DIM 3%,A,B,C$(50)
50 BUILD #2,"FORDAT"
60 WRITE #2,0;A,B;
70 MAT WRITE #2,0,2;C$;
```

    Erstellen einer formatierten Datei auf Kanal 2 mit folgender Datensatzstruktur:

    Feld 0: Numerisch, 3 Worte
    Feld 1: Numerisch, 3 Worte
    Feld 2: Matrix-Feld, 25 Worte

    Auch in allen weiteren Sätzen dieser Datei können die beiden ersten Felder nur mit 3%-Variablen per WRITE # und das dritte Feld per MAT WRITE # beschrieben werden.

6.4.2.11 Die KILL-Anweisung

Syntax:

KILL ⟨String⟩ [,⟨String⟩]ⁿ

Die Anweisung dient zum Löschen von Magnetplattendateien: die ersten zwei Zeichen des Dateinamens im Platteninhaltsverzeichnis (INDEX) werden gelöscht, und die belegten Plattenblöcke werden in der Plattenbelegungsliste als frei eingetragen

Anmerkung:

Ist die Datei noch von anderen Teilnehmer eröffnet, erfolgt die Freigabe der belegten Blöcke erst bei der Durchführung eines IPL bzw. INSTALL.

⟨String⟩ : String-Variable und String-Literale, die die Namen der zu löschenden Dateien ("⟨Datei⟩") aufnehmen.

Wird mit KILL eine Datei gelöscht, die durch einen anderen Teilnehmer oder auch durch das löschende Programm eröffnet ist, wird nur ein Lösch-Kennzeichen im Datei-Kennsatz eingetragen. Eine Datei mit eingetragenem Lösch-Kennzeichen kann nicht mehr eröffnet werden. Sie wird gelöscht, sobald der letzte Teilnehmer, der sie eröffnet hat, sie schließt.

Mit einer KILL-Anweisung können mehrere Dateien gelöscht werden. Tritt dabei ein Fehler auf, kann nicht festgestellt werden, durch welche Datei er verursacht wurde.

- Beispiel:

  Löschen einer Datei, deren Name in der Variablen A$ vorgegeben wird.

```
:
60 KILL A$
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.5 Spezielle Anweisungen

Dieser Punkt enthält die Beschreibung der Anweisungen:

- REM
- SIGNAL 1
- SIGNAL 2
- SIGNAL 3
- CALL
- LGERR

Zusätzlich ist die Möglichkeit der Angabe von Kommentaren in beliebigen BASIC-Anweisungen beschrieben.

6.5.1 Kommentare in BASIC-Programmen

Bei längeren BASIC-Programmen ist es notwendig, die Übersichtlichkeit des Programms durch Kommentare zu erhöhen. Dabei ist allerdings zu berücksichtigen, daß Kommentare sowohl Speicherplatz belegen als auch die Programmausführungszeit beeinflussen.

Zur Kommentierung in Programmen gibt es zwei Möglichkeiten:

- die REM-Anweisung
- "/*" nach beliebigen Anweisungen.

Die BASIC-Anweisungen

6.5.1.1 Die REM-Anweisung

Syntax:

REM [⟨Zeich⟩]^{n}

Nach dem Schlüsselwort kann eine beliebige Anzahl von Zeichen folgen, die als Kommentar dienen sollen.

- Beispiel:

```
100 REM ***** ALLGEMEINE FEHLERMELDUNG *****
```

Anmerkung:

Die Beschreibung der REM01 Kommentarzeile, die jedes BASIC-Standardprogramm enthalten muß, finden Sie im Kapitel "Programmwartung" unter "Programmvergleich".

6.5.1.2 Kommentare in beliebigen Anweisungen

Syntax:

⟨Anweisung⟩ /* [⟨Zeich⟩]^{n}

Der Kommentar wird durch die Zeichenfolge /* von der Anweisung getrennt.
Bei Ausgabe des BASIC-Programmes mit LIST oder DUMP werden zwischen der Anweisung und der Zeichenfolge /* immer zwei Leerzeichen ausgegeben.

- Beispiel: 10 IF ERR 0 GOSUB 9000 /*FEHLERVERZWEIGUNG

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Anweisungen

6.5.2 Inter-Task Kommunikation

Das System 8870/U ist ein multi-user-System, d.h. mehrere Teilnehmer können gleichzeitig mit dem System arbeiten. Zur Kommunikation zwischen den angeschlossenen Teilnehmern bietet Business-BASIC die Möglichkeit, Daten oder Nachrichten von Programmen der einzelnen Teilnehmer (Tasks) während der Verarbeitung auszutauschen.

Hierzu stehen folgende Anweisungen zur Verfügung:

SIGNAL 1 : Übergabe von numerischen Werten für einen anderen Teilnehmer in die "Signal-Liste".

SIGNAL 2 : Übernahme von numerischen Werten, die mit SIGNAL 1 der "Signal-Liste" übergeben wurden.

CALL 2 : Übergabe des Inhalts von Variablen in den gemeinsamen Bereich (Common Area) eines beliebigen Teilnehmers.

CALL 3 : Übernahme des Inhalts des gemeinsamen Bereiches (Common Area) eines beliebigen Teilnehmers in beliebige Variablen.

CALL 4 : Ausgabe einer Nachricht auf dem Bildschirm eines anderen Arbeitsplatzes.

Das System verfügt über eine Liste (Signal-Liste), in der die mit der Anweisung SIGNAL 1 übermittelten beiden Werte, die Nummer des sendenden Arbeitsplatzes und die Nummer des adressierten Arbeitsplatzes abgestellt werden. Jede Eintragung besteht also aus vier Werten.

Die BASIC-Anweisungen

6.5.2.1 Die SIGNAL 1-Anweisung

Syntax:

SIGNAL 1,⟨NAusdr1⟩,⟨NAusdr2⟩,⟨NAusdr3⟩

Überstellen von vier numerischen Werten in die Signal-Liste. Ein Wert ist die Nummer des sendenden Arbeitsplatzes, die automatisch eingesetzt wird.

Die anderen drei Werte sind in der Anweisung wie folgt anzugeben:

⟨NAusdr1⟩ : Arbeitsplatznummer des adressierten Teilnehmers.

⟨NAusdr2⟩ : In die Signal-Liste zu überstellende beliebi-
⟨NAusdr3⟩ ge numerische Werte. Ihr Wert darf 32767 nicht übersteigen.

Die Werte bleiben solange in der Signal-Liste, bis vom adressierten Teilnehmer eine SIGNAL 2-Anweisung ausgeführt wird.

Ist eine Teilnehmernummer angegeben, die nicht konfiguriert ist, oder ist die Signal-Liste voll, erscheint

ERROR # 62 (SIGNALPUFFER VOLL/PORT-NR. NICHT VORHANDEN)

Ein Eintrag in der Signal-Liste wird gestrichen, wenn er zwei Stunden in der Liste steht, ohne daß er mit einer SIGNAL 2-Anweisung abgerufen wurde.

Anmerkung:

Mit "CTL-B" kann jederzeit (d.h. daß der Arbeitsplatz sich nicht im INPUT-mode befinden muß) ein Eintrag von zwei Nullen (0) für den eigenen Arbeitsplatz in die Signal-Liste gemacht werden.

- Beispiel:

  Überstellen der Werte 0 und 10 in die Signal-Liste. Adressiert wird der Masterplatz.

```
:
50 LET P=0              /* ADR. ARBEITSPLATZ
60 LET X=0              /* ⟨NAusdr2⟩
70 LET Y=10             /* ⟨NAusdr3⟩
80 SIGNAL 1,P,X,Y
```


© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.5.2.2 Die SIGNAL 2-Anweisung

Syntax:

SIGNAL 2,⟨NVar1⟩,⟨NVar2⟩,⟨NVar3⟩ [,⟨NAusdr⟩]

Abrufen numerischer Werte aus der Signal-Liste in numerische Variable.

Mit ⟨NAusdr⟩ kann ein Zeitraum in Zehntelsekunden angegeben werden, für den auf die Übermittlung eines "Signals" gewartet wird.
Wird ein Signal empfangen, bevor die Zeit abgelaufen ist, wird die Ausführung des Programms fortgesetzt.

Der Wert von ⟨NAusdr⟩ muß zwischen 1 und 32767 liegen.

Die numerischen Variablen ⟨NVar1⟩ bis ⟨NVar3⟩ enthalten nach der Ausführung der Anweisung folgende Werte:

⟨NVar1⟩ : Arbeitsplatznummer des Teilnehmers, von dem das Signal in die Signal-Liste überstellt wurde. Liegt kein Signal vor, enthält ⟨NVar1⟩ den Wert -1.

⟨NVar2⟩ : Die mit der SIGNAL 1-Anweisung in die Signal-
⟨NVar3⟩ Liste überstellten numerischen Werte. Liegt kein Signal vor, bleiben die Werte der Variablen unverändert.

- Beispiel:

Ein Teilnehmer wartet auf ein Signal vom Masterplatz. Signale anderer Arbeitsplätze werden ignoriert. Das Programm läuft erst dann weiter, wenn ein Signal empfangen wurde.

```
:
100 SIGNAL 2,T,X,Y,100  /* 10 SEKUNDEN WARTEN
110 IF T<>0 GOTO 100    /* KEIN SIGNAL VON MASTER
```

6.5.3 Die SIGNAL 3-Anweisung

Syntax:

```
SIGNAL 3,<NAusdr>
```

Das Programm wird für den in <NAusdr> angegebenen Zeitraum in einen Wartezustand versetzt. Bis zum Ablauf der angegebenen Zeit erhält das Programm keine CPU-Zeit zugeteilt. Der Inhalt des Ein-/Ausgabepuffer des aufrufenden Arbeitsplatzes wird ausgegeben.

Der Wert von <NAusdr> muß zwischen 0 und 32767 (einschließlich) betragen und gibt die Wartezeit in Zehntelsekunden an.

- Beispiel:

  Ausgabe einer Fehlermeldung am Bildschirm. Nach 30 Sekunden wird die Fehlermeldung gelöscht und das Programm arbeitet weiter.

```
:
9000 PRINT M$;
9010 SIGNAL 3,300
9020 PRINT 'LD';
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

6.5.4 Die CALL-Anweisungen

Syntax:

CALL <NAusdr> [,<NumPar> / ,<String>]^12

Mit CALL-Anweisungen werden Bibliotheks-Unterprogramme in Maschinencode aufgerufen. Diese Unterprogramme erlauben die Durchführung von Funktionen, die im Sprachumfang von Business-BASIC nicht vorhanden sind oder nur mit Hilfe aufwendiger Routinen zu lösen wären.

Der Wert Wert von <NAusdr> gibt die Nummer und damit die Funktion des Unterprogrammes an.

Im Anschluß an die Unterprogrammnummer folgt eine Liste von Parametern, bestehend aus numerischen und String-Variablen oder Konstanten, in denen Werte vom BASIC-Programm an das Unterprogramm bzw. vom Unterprogramm an das BASIC-Programm (nur in Variablen) übergeben werden. Die Parameter-Liste hat je nach CALL-Unterprogramm einen speziellen Aufbau.

> Die Adressierung von Sub-Strings (z.B.: A$(2,5)) ist nicht möglich. Elemente von Vektoren und Matrizen können nur adressiert werden, wenn sie vorher dimensioniert wurden.

Eine eingehende Beschreibung aller verfügbaren CALL-Funktionen erfolgt in Kapitel "CALL-Anweisungen".

6.5.5 Die LGERR-Anweisung

Syntax:

```
LGERR
```

Diese Anweisung bewirkt den Eintrag von vorher aufgetretenen BASIC-Fehlern ins Systemlogbuch TF.LOGFILE in der Form

```
ERROR # <Ganzzahl> AT <Zlnr> IN <LU>/<Prog.Name>
```

<Ganzzahl> : 1- bis 3-stellige Fehlernummer wie bei SPC8

<Zlnr> : 5-stellige Zeilennummer wie bei SPC10

<Prog.name>: 14-stelliger Programmname wie bei PRN

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

7 Die BASIC-Funktionen

BASIC stellt dem Programmierer eine Reihe von Funktionen zur Verfügung, die bei ihrem Aufruf entweder einen numerischen Wert (⟨NFkt⟩) oder eine Matrix (⟨MFkt⟩) liefern.

Syntax:

⟨NFkt⟩::= { ⟨FktName⟩ (⟨NAusdr⟩) | LEN (⟨SVar⟩) | DET (⟨MVar⟩) }

wobei

⟨FktName⟩::= SIN COS TAN ATN SQR LOG EXP
ABS SGN INT FRA RND NOT MAN
CHR IXR SPC CHF KEY LKY

⟨MFkt⟩::= { { INV | MIV | TRN } (⟨MVar⟩) | { ZER | CON } [(⟨NAusdr⟩[,⟨NAusdr⟩])] | IDN [(⟨NAusdr⟩,⟨NAusdr⟩)] }

Zusätzlich gibt es noch die Funktionen CKY, FLN und PRN, die keinen numerischen Wert liefern, und die Funktion CHN.

Die Angabe der Klammern kann bei den numerischen Funktionen entfallen, wenn ⟨NAusdr⟩ ein nicht zusammengesetzter numerischer Ausdruck ist. Ebenso können bei der LEN-Funktion die Klammern entfallen.

Die Matrixfunktionen wurden ausführlich unter Beschreibung der Anweisungen erläutert, die Wirkungsweise der numerischen Funktionen wird in den folgenden Abschnitten erklärt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

7.1 Mathematische Funktionen

7.1.1 Trigonometrische Funktionen

Die zur Verfügung stehenden trigonometrischen Funktionen sind:

- SIN (Sinus)
- COS (Cosinus)
- TAN (Tangens)
- ATN (Arcustangens)

7.1.1.1 Die Funktionen SIN, COS und TAN

Das Argument <NAusdr> für die Sinus-, Cosinus- bzw. Tangensfunktion ist ein in Bogenmaß ausgedrückter Winkel. Ist der in Graden gegeben, muß er vorher mit einer der Formeln:

```
X * 3,1415926535898 / 180 oder
X / 57,295779513085
```

in Bogenmaß umgewandelt werden.

- Beispiele:

```
:

50 LET H=SIN(B)*L
60 LET C=COS(B)
70 PRINT TAN(B)*D/2
```

Die BASIC-Funktionen

7.1.1.2 Die Funktion ATN

Das Argument <NAusdr> für die Arcustangensfunktion kann jede beliebige Zahl sein.
Die Arcustangensfunktion liefert einen Winkel im Bogenmaß und ist die Umkehrfunktion der Tangensfunktion:

TAN (ATN(X)) = X
ATN (TAN(X)) = X für $-\frac{\pi}{2} < X < \frac{\pi}{2}$

- Beispiel: 50 LET B=ATN(H/D)

7.1.2 Transzendente Funktionen

Die zur Verfügung stehenden transzendenten Funktionen sind:

- SQR (Quadratwurzel)
- LOG (Natürlicher Logarithmus)
- EXP (Exponentialfunktion)

7.1.2.1 Die Funktion SQR

Die Funktion SQR liefert die Quadratwurzel von <NAusdr>.

- Beispiel:

  50 LET X=SQR(36)

  Ergebnis: X=6

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

7.1.2.2 Die Funktion EXP

Die Funktion EXP ist Exponentialfunktion zur Basis e (Eulersche Zahl):

EXP(X) = e↑X (e= 2,71828182..)

- Beispiel:

  50 LET X=EXP(0)

  Ergebnis: X=1

7.1.2.3 Die Funktion LOG

Die Funktion LOG liefert den natürlichen Logarithmus mit Basis e von ⟨NAusdr⟩ und ist die Umkehrfunktion der Exponentialfunktion:

LOG(X) = ln(X)

- Beispiel:

  50 LET X=LOG(EXP(1))
  Ergebnis: X=1

7.1.3 Arithmetische Funktionen

Die zur Verfügung stehenden arithmetischen Funktionen sind:

- ABS (Absolutwert)
- SGN (Signumfunktion)
- INT (Ganzzahlenwert)
- NOT (0-1-Funktion)
- FRA (Bruchteil)
- RND (Zufallszahl)

Die BASIC-Funktionen

7.1.3.1 Die Funktion ABS

Die Funktion ABS liefert den absoluten Wert von ⟨NAusdr⟩, d.h. den Wert von ⟨NAusdr⟩ mit positivem Vorzeichen.

- Beispiel: 50 LET A=ABS(A/B)

7.1.3.2 Die Funktion SGN

Die Funktion SGN liefert

```
 0, wenn der Wert von ⟨NAusdr⟩ = 0 ist.
+1, wenn der Wert von ⟨NAusdr⟩ positiv ist.
-1, wenn der Wert von ⟨NAusdr⟩ negativ ist.
```

- Beispiel: 50 LET X=SGN(A/B)

7.1.3.3 Die Funktion INT

Die Funktion INT liefert die größte Ganzzahl, die den Wert von ⟨NAusdr⟩ nicht überschreitet.

- Beispiele:

⟨NAusdr⟩	INT(⟨NAusdr⟩)
0,5	0
-1,999	-2
1,99	1

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

7.1.3.4 Die Funktion NOT

Diese Funktion liefert als Ergebnis:

0, wenn das Argument ungleich 0 ist
1, wenn das Argument gleich 0 ist.

- Beispiele:

  - Abstellen einer 0 oder 1 in der Variablen B, abhängig vom Inhalt der Variablen A.

```
:
50 LET B=NOT A
```

  - Aufrunden des Inhaltes der Variablen B:

```
:
50 LET B=INT B+NOT NOT FRA B
```

7.1.3.5 Die Funktion FRA

Die Funktion FRA liefert den Nachkommawert und das Vorzeichen des Wertes von ⟨NAusdr⟩.

- Beispiel:

⟨NAusdr⟩	FRA(⟨NAusdr⟩)
+1,1234	+0,1234
-1,5	-0,5

Die BASIC-Funktionen

7.1.3.6 Die Funktion RND

Die Funktion RND liefert eine Zahl, die der Rechner aus einer Folge von Pseudozufallszahlen generiert. Die Zufallszahl liegt in folgendem Bereich:

0 $\leq$ RND(⟨NAusdr⟩) < ⟨NAusdr⟩

Wird als ⟨NAusdr⟩ 0 angegeben, wird 1 eingesetzt.

- Beispiel:

  Bilden einer Ganzzahl im Bereich von +1 bis +6. Die ermittelte Zahl wird in der Variablen A abgestellt.

```
50 LET A=INT(RND(6))+1
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

7.2 Funktionen zur Zahlenmanipulation

Es stehen die Funktionen:

- MAN (Mantisse)
- CHR (Charakteristik)
- IXR (10↑Vorkommateil)

zur Verfügung.

Diese Funktionen ermöglichen die Zerlegung von Gleitkommazahlen in ihre Bestandteile.
Eine Gleitkommazahl wird in der Form:

$$X = m * 10 \uparrow e \quad \text{mit } 0.1 \leq ABS(m) < 1$$

dargestellt. Das Vorzeichen der Zahl gehört mit zur Mantisse

- m = Mantisse
- 10 = Basis
- e = Charakteristik (Exponent)

Durch die Funktionen werden folgende Werte geliefert:

- MAN (X) = m
- CHR (X) = e
- IXR (CHR(X)) = 10↑e
- IXR (X) = 10↑(SGN (X) * INT (ABS (X))

X läßt sich mit diesen Funktionen wie folgt darstellen:

```
X = MAN (X) * IXR (CHR (X))
```

Die BASIC-Funktionen

7.2.1 Die Funktion MAN

Die Funktion MAN liefert die Mantisse einer beliebigen Zahl (bei einer Ganzzahl nach Umrechnung in eine Gleitkommazahl).

MAN (X) = X/10↑CHR (X)

Das von MAN gelieferte Ergebnis ist eine Nachkommazahl.

- Beispiel:

Die Variable X enthält den Wert 57,60.
Nach Durchführung der Anweisung

```
50 LET Y=MAN(X)
```

enthält die Variable Y den Wert 0,576.

7.2.2 Die Funktion CHR

Die Funktion CHR liefert die Charakteristik (Exponent) einer Gleitkommazahl. CHR (X) ergibt immer eine Ganzzahl. Diese Funktion ermöglicht z.B. die Feststellung der Anzahl Ziffern vor dem Komma.

- Beispiel:

Überprüfung einer Eingabe auf:

- maximal 5 Vorkommastellen
- minimal 3 Vorkommastellen
- Nachkommastellen sind nicht erlaubt.

```
:
100 INPUT X
110 IF FRA(X)=0 IF CHR(X)<6 IF CHR(X)>2 GOTO 130
120 GOTO 100 /* FALSCHE EINGABE
130 ...      /* EINGABE OK
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

7.2.3 Die Funktion IXR

Die Funktion IXR liefert den Wert:

IXR (X) = 10↑ (SGN (X) * INT (ABS (X)))

- Beispiel:

```
- :                              - :
  50 LET X=4.25                    50 LET X=-2.6
  60 LET Y=IXR(X)                  60 LET Y=IXR(X)
  :                                :
  Ergebnis: Y=10000                Ergebnis: Y=0.01
```

7.3 Sonderfunktionen

Als Sonderfunktion stehen

- SPC - TAB - CHN
- KEY - CKY - FLN
- LEN - CHF - PRN
- LKY - DET

zur Verfügung.

7.3.1 Die Funktion SPC

Die Funktion SPC liefert abhängig von dem unter ⟨NAusdr⟩ angegebenen Wert folgende Informationen:

⟨NAusdr⟩	Information
0	Beanspruchte CPU-Zeit in Zehntelsekunden seit der Anmeldung.
1	Anschlußzeit in Minuten seit der Anmeldung.
2	Vergangene Zeit in Stunden seit dem 1.1.1980.
3	Stundenteil in Zehntelsekunden (Ergänzung zu SPC 2).
4	System-Erstellungsdatum, als Differenz in Stunden zum Basisdatum.
5	Benutzernummer.
6	Nummer des ausführenden Arbeitsplatzes.
8	Die Nummer des zuletzt aufgetretenen BASIC-Fehlers. Diese Funktion liefert 0, wenn bisher noch kein BASIC-Fehler aufgetreten ist.
9	Aktuelle Zeilennummer.
10	Zeilennummer der Anweisung, die den letzten BASIC-Fehler verursacht hat. Diese Funktion liefert 0, wenn bisher noch kein BASIC-Fehler aufgetreten ist.
12	Nummer der physikalischen Platteneinheit, von de zuletzt ein IPL durchgeführt wurde.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

<NAusdr>	Information
14	Programmstart aus TAMOS-Selektor : 0 = Das Programm wurde nicht unter TAMOS gestartet. 1 = Das Programm wurde unter TAMOS gestartet.
15	Typ des Arbeitsplatzes, an dem das Programm läuft: 0 = nicht ladbarer BA: DAP - Slave 1 = Phantomport 2 = ladbarer BA: RAP (DAP 3 - Master) 3 = ladbarer BA: BA13 (DAP 4 - Master)
20	Zustand des Druck Spooling : -1 : Nicht aktiv >=0 : Anzahl aktiver Jobs
22	Anzahl freier Blöcke der LU 0 (Systemplatte)

- Beispiel:

Auswertung von BASIC-Fehlern in einer Fehlerbehandlungsroutine mit gezielter Reaktion auf den Fehler 98 und die Betätigung der ESC-Taste. Die Routine ist als Unterprogramm aufgebaut.

```
40 IF ESC REM       /* ESC-TASTE WIRKUNGSLOS
50 IF ERR 0 GOSUB 9005
:
1490 INPUT A
:
9005 IF SPC 8=98 GOTO 9030
9010 PRINT TAB (15,24);"BASIC-FEHLER # ";SPC 8;
"ZEILENNUMMER : "; SPC 10;
9012 LGERR          /* EINTRAG IN LOGDATEI
9015 SIGNAL 3,20
9020 CHAIN A$
9030 PRINT TAB (15,24);"EINGABE MUSS NUMERISCH SEIN";
9035 SIGNAL 3,20
9040 PRINT 'LD';  /* MELDUNGSZEILE LOESCHEN
9045 RETURN -1
```

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

7.3.2 Die Funktion LEN

Die Funktion LEN ermittelt die Position des ersten Grenzzeichens in String-Variablen und übergibt diese Position minus 1, d.h. die aktuelle Länge des Strings. Die String-Variable ist als Argument der LEN-Funktion anzugeben. Durch diese Funktion wird z.B. die Adressierung von Strings in Abhängigkeit von ihrer aktuellen Länge ermöglicht.

- Beispiele:

```
- 10 DIM A$(20)
   :
   50 LET A$="DIENSTAG"
   60 LET X=LEN(A$)          (Ergebnis: X=8)
   70 LET Y=LEN(A$(5))       (Ergebnis: Y=4)
- Abstellen des Zeichens ":" hinter dem derzeit letzten
  Zeichen im String A$.
   :
   110 LET A$(LEN A$+1)=":"
```

7.3.3 Die Funktion CHF

Die Funktion CHF ermöglicht dem Programmierer, Informationen über eine Datei, die auf einem Kanal eröffnet worden ist, zu ermitteln (Alternative zum CALL 96).
Das Argument der Funktion definiert die Kanalnummer und die durchzuführende Funktion.
Für die verschiedenen Arten von Dateien werden durch CHF verschiedene Angaben übergeben. Es bestehen folgende Funktionsmöglichkeiten (C = Kanalnummer):

Dateiart	<NAusdr>	Funktion
Formatiert	C	Nummer des höchsten geschriebenen Datensatzes +1.
	C+100	Nummer des Datensatzes, auf den zuletzt zugegriffen wurde.
	C+200	Byteadresse des zuletzt bearbeiteten Datensatzes, relativ zum Blockanfang
	C+300	Satzlänge in Worten

Die BASIC-Funktionen

	Dateiart	NAusdr	Funktion
	Relativ	C	Anzahl Datensätze, die die Datei maximal aufnehmen kann.
		C+100	Nummer des Datensatzes, auf den zuletzt zugegriffen wurde.
		C+200	Byteadresse des zuletzt bearbeiteten Datensatzes, relativ zum Blockanfang
		C+300	Satzlänge in Worten.
*		C+400	Nummer des Datensatzes, der hinter dem höchsten geschriebenen steht, wenn der EOF-Zeiger definiert ist. Das ist er nur, wenn die Datei vorher mit OPEN "C" eröffnet worden ist. Sonst ergibt sich die Anzahl der Datensätze, die die Datei aufnehmen kann.
	Index	C	Nummer des höchsten Datensatzes.
		C+100	Nummer des Datensatzes, auf den zuletzt zugegriffen wurde.
		C+200	Byteadresse des zuletzt bearbeiteten Datensatzes, relativ zum Blockanfang
		C+300	Satzlänge in Worten
	Text	C	Nummer des höchsten geschriebenen Datenblocks +1.
		C+100	Nummer des aktuellen Datenblocks.
		C+200	Nummer des Bytes, das hinter dem zuletzt übertragenen Byte liegt, relativ zum Anfang des aktuellen Blocks.
	Drucker	C	Aktuelle Druckzeile
		C+200	Druckbetrieb über Spooler CHF(C+200) = 0 : NEIN CHF(C+200) <> 0 : JA

Für Indexdateien können weitere Informationen über den Dateistatus mit SEARCH # -Modus 1- abgefragt werden.

7.3.4 Die Funktion DET

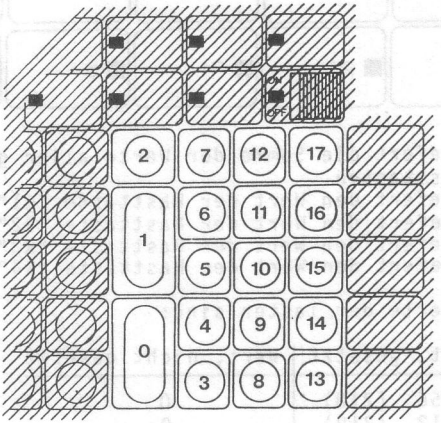
Die Funktion DET liefert den Determinantenwert der zuletzt invertierten Matrix.
Als Argument ist eine Matrix-Variable anzugeben (<MVar>) die jedoch ohne Bedeutung ist.

Die BASIC-Funktionen

7.3.5 Die Funktion KEY

Die Funktion KEY liefert den numerischen Wert der zuletzt betätigten Auslösetaste. Als Argument der Funktion ist nur 0 zugelassen.

Die Werte für die Tasten sind:



- Beispiel:

```
:
11  IF KEY (0)=4 GOTO 250
:
250 REM *** A4 BETÄTIGT ***
```

Anmerkung:

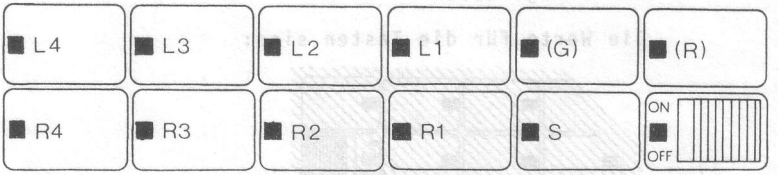
Die Tasten "Y" und "N" im numerischen Tastenfeld sind wirkungsgleich mit "Y" "CR" und "N" "CR" und erzeugen wie "CR" den Wert 0.

Beim Betätigen von Tasten, die keine Auslösetasten sind, bleibt KEY (0) ohne Auswirkung.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

7.3.6 Die Funktion LKY

Die Funktion LKY liefert den Wert der gesetzten Rastta-ste(n):



- LKY (0) liefert die Summe der Werte aller gesetzten Rasttasten.
- LKY (1) liefert den Wert der Rasttaste "R1".
- LKY (2) liefert den Wert der Rasttaste "R2".
- LKY (3) liefert den Wert der Rasttaste "R3".
- LKY (4) liefert den Wert der Rasttaste "R4".

Die einzelnen Werte der Tasten sind:

Rasttaste	Wert, gesetzt	Wert, nicht gesetzt
"R1"	256 (2↑8)	0
"R2"	512 (2↑9)	0
"R3"	1024 (2↑10)	0
"R4"	2048 (2↑11)	0

- Beispiel:

```
190 IF LKY (0)=1536 GOTO 250
 :
250 REM *** R2, R3  GESETZT ***
 :
550 IF LKY (4)=2048 GOTO 660
 :
660 REM *** R4 GESETZT ***
```

Anmerkung:

Bei Nutzung eines Programms für REMOTE-Verarbeitung sollte auf die Programmierung der Rasttasten verzichtet werden, da diese nur lokal genutzt werden können.

Die BASIC-Funktionen

7.3.7 Die Funktion CKY

Die Funktion CKY löscht alle gesetzten Rasttasten und setzt die Rasttastenwerte auf 0. Das Argument der Funktion muß 0 sein.

Die Funktion ist ausschließlich in den Anweisungen PRINT und INPUT (analog den TAB-Funktionen) erlaubt.

- Beispiel:

```
100 INPUT CKY (0), A
 :
190 PRINT CKY (0)
```

7.3.8 Die Funktion CHN

Die Funktion CHN (ohne Parameter) liefert die Nummer des Kanals, in dessen Zusammenhang zuletzt ein Fehler aufgetreten ist.
Ist kein BASIC-Fehler oder nur einer ohne Bezug zu einer Kanalnummer aufgetreten, liefert CHN den Wert -1 .
Nach CHAIN oder LINK wird CHN auf -1 gesetzt.
Diese Funktion ist bei den Anweisungen IF, LET und PRINT zugelassen. Sie ist nicht erlaubt in einem numerischen Ausdruck.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die BASIC-Funktionen

7.3.9 Die Funktion FLN

Die Funktion FLN (ohne Parameter) liefert den Namen der letzten Datei (bzw. des Drivers), bei deren Benutzung ein Fehler aufgetreten ist, und zwar maximal 17-stellig in der Form <LU>/Datei-(bzw. Driver-) Name.
Nach CHAIN oder LINK wird FLN gelöscht.
Diese Funktion ist bei den Anweisungen IF, LET und PRINT zugelassen.

7.3.10 Die Funktion PRN

Die Funktion PRN (ohne Parameter) liefert den aktuellen Programmnamen maximal 17-stellig in der Form <LU>/<Programmname> .
Diese Funktion ist nützlich bei mehrfach mit CHAIN verketteten Programmen und bei Fehlerroutinen.
Diese Funktion ist bei den Anweisungen IF, LET und PRINT zugelassen.

Die CALL-Anweisungen

8 Die CALL-Anweisungen

CALL-Unterprogramme sind Maschinencode-Unterprogramme, die mit der Anweisung CALL aufgerufen werden. Diese Unterprogramme erlauben die Durchführung von Funktionen, die im Sprachumfang von Business-BASIC nicht vorhanden sind oder nur mit Hilfe aufwendiger Routinen zu lösen wären.

Syntax:

CALL ⟨NAusdr⟩ { ,⟨NumPar⟩ / ,⟨String⟩ }[12]

⟨NAusdr⟩ : Nummer des aufzurufenden Unterprogramms (siehe Kap. "Übersicht über verfügbare Unterprogramme).

⟨NumPar⟩ ⟨String⟩ : Maximal 12 numerische Parameter (⟨NVar⟩ oder ⟨Dezzahl⟩) oder Strings (⟨SVar⟩ oder ⟨SLit⟩). In vielen Fällen sind als numerische Parameter nur Ganzzahlen sinnvoll (Nachkommateil wird dann ignoriert). Diese Größen dienen zur Vorgabe von Parametern an das Unterprogramm bzw. zum Austausch von Informationen zwischen dem BASIC-Programm und dem Unterprogramm.

Die Adressierung von Sub-Strings (z.B. A$(X,Y)) ist nicht möglich, während bei Vektoren und Matrizen einzelne Elemente adressiert werden können.

Auf Einschränkungen/Bedingungen, die nicht allgemein gültig sind, wird bei der Beschreibung der zur Verfügung stehenden Unterprogramme hingewiesen.

BASIC-Fehler	Bedeutung
36	Aufgerufenes Unterprogramm ist nicht vorhanden.
37	Statt einer Variablen wurde ein Ausdruck angegeben.
38	Falscher Variablentyp/Fehler vom aufgerufenen Unterprogramm erkannt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.1 Übersicht über die verfügbaren Unterprogramme

CALL-Nummer	Funktion
1	Bildschirmaufbereitung für eine Eingabe, Prüfung und Formatierung eingegebener Daten.
2	Transport des Inhalts von Variablen in den Common-Bereich eines Arbeitsplatzes.
3	Transport von Daten aus dem Common-Bereich eines Platzes in eine oder mehrere Variablen.
4	Ausgabe einer Nachricht an einem anderen Bildschirm oder Schreiben/Lesen einer Datenträger-Archiv-Nummer.
6	Prüfziffernberechnung einer alphanumerisch dargestellten Zahl.
7	Prüfsummenbildung über einen String.
20	Lesen, Setzen oder Löschen eines Bits in einem String.
21	Umwandlung des Inhalts einer numerischen Variablen in eine Dualzahl.
22	Umwandlung einer Dualzahl und Übertragen des Wertes in eine numerische Variable.
23	Suchen einer Zeichenkombination in einem String.
24	Umkehrung der Zeichen-Reihenfolge in einem String.
25	Datumskonvertierung in Industriedatum.
26	Selektive Zeichenersetzung in Stringvariablen.
28	Konvertierung von Binärwerten in andere Zahlensysteme.
29	Lesen oder Ändern der Program Abort Flag und der Port Control Area.
31	Codeumwandlung.
32	Datumsbearbeitung.

Die CALL-Anweisungen

CALL-Nummer	Funktion
35	Magnetplatten-Konfiguration.
51	Unterstützung der Bildschirm-Programmierung.
59	Konvertierung von Zahlen von und nach IBM-Format.
60	Packen eines Strings.
61	Entpacken eines Strings.
62	Prüfung, ob der Inhalt eines Strings numerisch ist. Syntaxprüfung eines Strings.
63	Prüfung, ob der Inhalt eines Strings ein gültiger numerischer Wert ist.
64	Prüfen und Umstellen eines Datums in einem String.
65	Sortieren von Begriffen in einem String.
66	Erweiterte Codewandlung.
70	Magnetband-Programmierung.
72	Übertragen des Inhalts von Variablen in einem String.
73	Übertragen des Inhalts eines Strings in mehrere Variablen.
74	Lesen eines Eintrags der Gerätezuodnungstabelle eines BA.
80	Magnetbandcassetten-Programmierung.
84	Meldung lesen, anzeigen, ins Logbuch eintragen.
90	Bearbeitung von String-Variablen.
91	Bit-Invertierung in Teilstrings.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

CALL-Nummer	Funktion
93	Platteninformation ermitteln.
96	Zulässigkeitsprüfung für Dateinamen und Lesen von Informationen über Dateien.
97	Informationen aus Dateikennsätzen lesen.
98	Systemkommandos senden.
99	Datum und Uhrzeit übernehmen/ändern.

Die Call-Anweisungen

8.2 CALL 1

Bildschirmaufbereitung für eine Eingabe, Prüfung eingegebener Daten und Formatierung.

Es werden zwei Formate unterschieden:

Format Syntax

1 CALL <NAusdr>, <NVar1>, <String1>, <String2>, <NumPar>, <SVar2>

2 CALL <NAusdr>, <NVar1>, <String1>, <SVar1>, <NVar2>, <String3>, <String4>, <SVar2>

Semantik Format 1 (Bildschirmaufbereitung):

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des aufzurufenden Unterprogramms (in diesem Fall 1).

<NVar1> : Funktionsnummer, muß 0 sein (bei Format 1). Nach Ausführung des CALL enthält die Variable den Wert 1.

<String1> : Parameterstring zur Steuerung der Operation. Die Länge dieser Variablen bzw. dieses Literals muß 12 oder 13 Bytes betragen (siehe Beschreibung).

<String2> : Zeichen für das Eingabefeld. Enthält <String1> in Byte 1 die Zahl 3, wird bei alphanumerischer Eingabe der Inhalt von <String2> auf dem Eingabefeld ausgegeben. Bei numerisch ungepackter Eingabe (Byte 2 von <String1> = 0) wird der Inhalt von <NumPar> ausgegeben.

<NumPar> : Numerischer Wert, der bei ungepackt numerischer Eingabe auf dem Eingabefeld angezeigt wird.

<SVar2> : Arbeitsfeld, Länge >= 18 Bytes.

Falls auf dem Eingabefeld eine Maske ausgegeben wird, wird hinter das letzte Zeichen ein Hintergrundblank ausgegeben, damit die maximale Eingabefeldlänge nicht überschritten wird.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Semantik Format 2 (Prüfung und Formatierung der Eingabe):

<NAusdr>, <String1> und <SVar2> wie Format 1.

<NVar1> : Funktionsnummer, Wert 1 bei Format 2.
Die Variable enthält nach Ausführung den Status der Operation:
-1 = Eingabe war nicht korrekt.
0 = Eingabe war korrekt.
1 = Taste "↻" für Übernahme des vorhandenen Inhalts wurde betätigt.
2 = Taste "↑" für Zurücksetzen zur letzten Eingabe wurde betätigt.
* x = (x-2)-te 3-Zeichen-Funktion aus <String3> wurde eingegeben.

<SVar1> : Eingabestring, Dimension >= maximale Eingabelänge. Der Inhalt dieser Variablen wird nach der Eingabe gegen die Steuerparameter in <String1> geprüft und formatiert. Bei korrekter Eingabe werden:
- numerische Eingaben rechtsbündig mit führenden Nullen in <SVar1> abgestellt und fehlende Nachkommastellen mit Nullen aufgefüllt, bis max. # Nachkommastellen erreicht ist.
- alphanumerische Eingaben linksbündig abgestellt und eventuell rechts mit Blanks bis zur maximalen Eingabelänge aufgefüllt.

Anschließend wird der Eingabestring ohne führende Nullen am Bildschirm angezeigt.

* Bei Dimensionierung > 13 wird BASIC-Fehler
* #38 erzeugt.

<NVar2> : Bei numerischer Eingabe: Eingabewert.
Bei alphanum. Eingabe: Anzahl der eingegebenen Zeichen.
* (Das gilt allerdings nicht bei Eingabe über
* eine Standardfunktionstaste. Dann wird NVar2
* nicht aktualisiert.)

<String3> : Enthält in jeweils 3 Byte die Zeichenfolgen, die statt einer Eingabe als Steuerinformation interpretiert werden sollen (Beispiel: END, SPR usw.).

Die CALL-Anweisungen

⟨String4⟩ : Die Bytes dieses Strings geben an, welche Steuerzeichen statt einer Eingabe zulässig sind. 0 (Null) bedeutet, daß die Funktion erlaubt ist, jedes Zeichen ≠ 0 heißt, die Funktion ist nicht erlaubt.
Zuordnung der Bytes zu den Funktionen:
Byte 1 = Taste "@"
Byte 2 = Taste "↑"
Byte 3 = erste Zeichenfolge in ⟨String3⟩
:
Byte x = (x-2)-te Zeichenfolge in ⟨String3⟩

Beispiel:
⟨String3⟩="SPREND", ⟨String4⟩="0110" bedeuten:

"@" ist erlaubt,
"↑" ist nicht erlaubt,
SPR ist nicht erlaubt,
END ist erlaubt.

Anmerkung 1:

Falls ein Fehler auftritt, wird die entsprechende Meldung aus der Datei gelesen, die auf Kanal 1 (unter TAMOS TF.PARAM) eröffnet ist.

Anmerkung 2:

Nach Ausführung von CALL 1 ist die gemerkte Cursorposition gleich der ersten Eingabefeldposition.

Anmerkung 3:

Die Benutzung von CALL 1 mit Format 2 ohne vorherige Verwendung von CALL 1 mit Format 1 führt zu einem Fehler.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Beschreibung der Steuerparameter in <String1>:

Byte	Wert	Bedeutung
1	0	Keine Eingabemaske ausgeben.
	1	Eingabemaske in Form von Punkten ausgeben.
	2	Wie bei 1, jedoch zuvor 'Clear Foreground'.
	3	Inhalt von <String2>/<NumPar> als Eingabemaske ausgeben. Anmerkung: Dieses Byte wird bei Format 2 auf 0 gesetzt, wenn ein Fehler auftritt.
2	0	Numerische Eingabe.
	1	Alphanumerische Eingabe.
	2	Numerische Eingabe.
3-4	xx	Spaltennummer der Cursorposition.
5-6	xx	Zeilennummer der Cursorposition.
7-8	xx	Maximale Anzahl Vorkommastellen bzw. Zeichen.
9-10	xx	Minimale Anzahl Vorkommastellen bzw. Zeichen.
11	x	Anzahl Nachkommastellen.
12	0	Kein Vorzeichen erlaubt.
	1	Vorzeichen erlaubt.
13	0	Falls Eingabefeld nur 1-2 Zeichen, mit Blanks auf 3 Zeichen ergänzen vor dem Hintergrund-blank, um gegebenenfalls die 3-Zeichen-Steuer-informationseingabe (END, SPR, usw.) zu ermöglichen.
	1	Keine Blanks freihalten. Ist <String1> 12 Bytes lang, wird 0 angenommen.

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Unzulässige(r) Variablentyp oder -Dimensionierung
- Kanal 1 nicht eröffnet
- Unzulässige Werte im Parameterstring
- Maske für Eingabefeld zu lang
- <NVar2> ist 1%, die Eingabe >7999 oder <-7999
- Duplizieren bei numerischer Eingabe mit anschließendem Packen nicht zulässig.

Die Call-Anweisungen

8.3 CALL 2

Transport des Inhaltes von bis zu 11 Variablen in den Common-Bereich eines beliebigen Teilnehmers.

Syntax:

CALL <NAusdr>, <NumPar1> {,<String> / ,<NumPar2>}[11]

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 2).
<NumPar1> : Arbeitsplatznummer des Teilnehmers, in dessen Common-Bereich die Inhalte der folgenden Variablen übertragen werden sollen.
<String> : 1 bis 11 Variable oder Konstanten.
<NumPar2> Die dimensionierte Gesamtlänge aller zu übertragenden Daten darf 512 Byte nicht überschreiten.

Die Übertragung erfolgt wortweise. String-Variable werden ohne Rücksicht auf Grenzzeichen in ihrer dimensionierten Länge übertragen und bei ungerader Länge mit einem Grenzzeichen auf gerade Byteanzahl aufgefüllt.

- Beispiel:

  Übergabe des Inhaltes der Variablen B und A$ in den gemeinsamen Bereich des Phantom-Ports.

```
10 DIM A$(5),3%,B
:
60 CALL 2,1,A$,B
```

Positionen im Common-Bereich:
Bytes 1- 5: A$
Byte 6 : Grenzz.
Bytes 7-12: B

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Die dimensionierte Gesamtlänge der zu übertragenden Daten ist > 512 Byte.
- Die angegebene Arbeitsplatznummer ist nicht gültig.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.4 CALL 3

Transport von Daten aus dem Common-Bereich eines beliebigen Teilnehmers in bis zu 11 Variablen.

Syntax:

CALL <NAusdr>, <NumPar> { ,<SVar> / ,<NVar> }[11]

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 3).

<NumPar> : Arbeitsplatznummer des Teilnehmers, aus dessen Common-Bereich Daten übernommen werden sollen.

<NVar> : 1 bis 11 Variable.

<SVar> Die Daten werden physikalisch übertragen. Stimmen die angegebenen Variablen in Typ oder Länge nicht mit den zu übernehmenden Daten überein, ist das Ergebnis undefiniert.

Die dimensionierte Gesamtlänge aller aufnehmenden Variablen darf 512 Byte nicht überschreiten.

- Beispiel:

  Übernahme von Daten aus dem gemeinsamen Bereich des Phantom-Ports in die Variablen B und A$.

```
:
60 CALL 3,1,B,A$
```

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Die dimensionierte Gesamtlänge der aufnehmenden Variablen ist >512 Byte.
- Die angegebene Arbeitsplatznummer ist nicht gültig.

Die CALL-Anweisungen

8.5 CALL 4

Dieses Unterprogramm dient zur Ausführung der Funktionen:

- Ausgabe von Nachrichten an anderen Arbeitsplätzen
- Schreiben/Lesen der Archivnummer

8.5.1 Ausgabe von Nachrichten an anderen Arbeitsplätzen

Syntax:

CALL ⟨NAusdr⟩, ⟨String⟩ [,⟨NumPar⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des aufzurufenden Unterprogramms (in diesem Fall 4).

⟨String⟩ : String, dessen Inhalt auf dem Bildschirm des adressierten Platzes ausgegeben werden soll.

⟨NumPar⟩ : Adressierter Arbeitsplatz. Ist die Nummer nicht angegeben, wird die Nachricht am Masterplatz angezeigt.

Diese Anweisung wird nur dann korrekt ausgeführt, wenn der adressierte Platz im INPUT-mode steht. Jeder andere Zustand bewirkt bei der Ausführung BASIC-Fehler 38. Die Anweisung ist deshalb so oft zu wiederholen, bis sie erfolgreich ausgeführt werden kann.
Die Ausgabe auf dem Bildschirm erfolgt in der aktuellen Bildschirmzeile ab der ersten möglichen Eingabeposition des INPUT. Daher sollten in ⟨String⟩ geeignete Bildschirm-Funktionen (z.B. TAB) vor der eigentlichen Nachricht enthalten sein. In jedem Fall wird die unvollendete Eingabe des Empfängers gelöscht.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

- Beispiel:

  Ausgabe einer Meldung auf Position 15 in Zeile 24 (Nachrichtenzeile) an Platz 2. Die Ausgabe wird wiederholt, bis die Anweisung fehlerfrei durchgeführt wird.

```
10 DIM M$(50)       /* STRING ZUR AUFNAHME DER NACHRICHT
:
50 LET M$="←207←←376←←211←←376←←221←←217←
←230←IN EINER STUNDE DATENSICHERUNG!!!←376←←212←"
60 IF ERR 0 GOTO 900     /* FEHLER # 38??
70 CALL 4,M$,2
80 IF ERR 0 GOSUB 9000   /* STANDARD-FEHLERBEHANDLUNG
:
900 IF SPC8<>38 CHAIN ../* PROGRAMMABBRUCH
910 SIGNAL 3,100         /* 10 SEKUNDEN WARTEN
920 GOTO 70              /* ERNEUTER VERSUCH
```

8.5.2 Schreiben/Lesen der Archiv- und der LU-Nummer

Syntax:

```
CALL <NAusdr> ,[<NVar1> , <NVar2>] ,{<NVar3>  ,}
                                    {<Dezzahl>}
                <SVar1> [,<SVar2>]
```

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 4).

<NVar1> : Numerische Variable (1 %), in der bei Mischanlagen das Driverkennzeichen vorgegeben werden muß, falls sich die Funktion auf andere als das Systemlaufwerk bezieht.
Es bedeuten:

SMD/CDH/FDU	8
CA	9
CMD	10
CDD	11
FD 8	12
MDL	13
FD 5 1/4	14

<NVar2> : Numerische Variable (1 %), in der nach Ausführung der Funktion die Driver-Nummer laut CONFIG abgestellt wird.

Die CALL-Anweisungen

⟨NVar3⟩ : Physikalische Einheit des Datenträgers, auf
⟨Dezzahl⟩ den die Archivnummer geschrieben bzw. von dem das Kennzeichen gelesen wird.

⟨SVar1⟩ : Stringvariable, Länge ⟩= 3 Byte, die die gelesene bzw. zu schreibende Archivnummer enthält.

⟨SVar2⟩ : Stringvariable, Länge ⟩= 2 Byte, die nach der Ausführung der Funktion die LU-Nummer des adressierten Datenträgers enthält.

Schreiben des Kennzeichens:

Ist die Länge von ⟨SVar1⟩ ≠ 0, werden die ersten beiden Bytes des Strings auf den Datenträger geschrieben, dessen physikalische Einheitennummer vorgegeben ist.

Lesen des Kennzeichens:

Ist ⟨SVar1⟩ ein Leerstring, wird das Kennzeichen des Datenträgers, der durch ⟨NVar1⟩ und ⟨NVar3⟩/ ⟨Dezzahl⟩ adressiert wird, in den String übertragen. Dabei wird die Common-Area des ausführenden Arbeitsplatzes überschrieben!

- Beispiele:

  - Die Archivnummer 99 wird auf die Magnetplatte mit der physikalischen Einheit 1 geschrieben.

```
:
100 LET U=1       /* GERÄTENUMMER LADEN
105 LET K$="99"   /* KENNZEICHEN LADEN
110 CALL 4,U,K$   /* KENNZEICHEN SCHREIBEN
```

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

- Das Kennzeichen wird von der Platte mit der physikalischen Einheitennummer 0 gelesen.

```
:
100 LET U=0      /* GERÄTENUMMER LADEN
105 LET K$=""    /* K$ = LEERSTRING
110 CALL 4,U,K$ /* KENNZEICHEN LESEN
```

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Schreib- oder Lesefehler Magnetplatte.
- Stringvariable für weniger als drei Byte dimensioniert.
- <NVar1> und/oder <NVar2> nicht als 1%-Variable dimensioniert.
- Ungültige Gerätenummer.
- PU-Nummer zu hoch.

Die CALL-Anweisungen

8.6 CALL 6

Prüfziffernberechnung einer alphanumerisch dargestellten Zahl.

Prüfziffern garantieren eine hohe Datensicherheit.
Mit dem CALL 6 stehen zwei verschiedene Methoden zur Prüfziffernberechnung zur Verfügung.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar⟩, ⟨NVar⟩, ⟨String1⟩ [,⟨String2⟩]
[,⟨String3⟩]

⟨NAusdr⟩: Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (6).

⟨NumPar⟩ : Funktionsnummer (1%):

⟨NumPar⟩	Prüfziffer
0	B - Z;*F; modB
1	Z;*F;modB

Erläuterung siehe unten.

⟨NVar⟩ : In ⟨NVar⟩ wird nach Ausführung des CALL 6 die Prüfziffer übergeben (1%).

⟨String1⟩ : In diesem String ist die Zahl abzustellen, für die die Prüfziffer errechnet werden soll. Es sind nur Ziffern und Blanks zugelassen, Blank wird als 0 interpretiert.

⟨String2⟩ : In diesem String (Länge 1 Byte) kann eine Basis zwischen 2 und 11 vorgegeben werden, wobei 10 und 11 als A bzw. B verschlüsselt werden müssen. Wird diese Angabe weggelassen, wird A eingesetzt.

⟨String3⟩ : In diesem String kann eine Faktorenliste vorgegeben werden. Die Faktoren können beliebige Ziffern oder A (für 10) oder B (für 11) sein. Wird keine Faktorenliste angegeben, wird "1" eingesetzt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Erläuterung der Funktionen:

B ist eine ganzzahlige Basis im Bereich 2 bis 11 und wird in ⟨String2⟩ übergeben.

Z;*F wird folgendermaßen aus der in ⟨String1⟩ übergebenen Zahl und der in ⟨String3⟩ übergebenen beliebigen Faktorenliste gebildet:

1. Ziffer(Zahl)*1.Ziffer(Faktoren)+2.Ziffer(Zahl)*2.Ziffer(Faktoren)+3.Ziffer(Zahl)*3.Ziffer...

Ist die Faktorenliste kürzer als die Zahl, wird sie nach Abarbeitung wieder von vorn begonnen.

Die so berechnete Summe wird durch die Basis B ganzzahlig mit Rest dividiert. Dieser Rest ist Z;*F;modB.

Sowohl bei Funktion 0 als auch bei 1 liegt die Prüfziffer also im Bereich 0 bis B-1.

- Beispiel:

  Zu der Zahl 1234567 und der Faktorenliste "OAB" soll zur Basis 8 die Prüfziffer nach Funktion 0 berechnet werden.

  Z;*F;=1*0+2*10+3*11+4*0+5*10+6*11+7*0=169

  169:8= 21 Rest 1, daß heißt Z;*F;mod 8=1

  Die Prüfziffer nach Funktion 0 ist also 8-1=7.

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Eine numerische Variable nicht mit 1% dimensioniert.
- ⟨String2⟩ länger als 1 Byte.
- Zeichen im Zahlenstring keine Ziffer und kein Blank.
- Unzulässiges Zeichen in der Basis oder der Faktorenliste.
- Funktionsnummer ⟨0 oder ⟩1.

Die CALL-Anweisungen

8.7 CALL 7

Prüfsummenbildung über einen String

Syntax:

CALL 7, ⟨NVar1⟩,⟨SVar⟩,⟨NVar2⟩

⟨NVar1⟩ : In dieser numerischen Variablen wird die Anzahl zu prüfender Bytes vorgegeben (Dimensionierung mind. 2%).

Nach Ausführung der Funktion enthält ⟨NVar1⟩ die Prüfsumme über ⟨SVar⟩.

⟨SVar⟩ : Stringvariable, über die die Prüfsumme gebildet werden soll, und zwar bis zur durch ⟨NVar1⟩ vorgegebenen Länge. ⟨SVar⟩ kann auch binäre Werte enthalten.

⟨NVar2⟩ : Statusvariable, gibt nach Ausführung der Funktion Auskunft über die Korrektheit der Prüfsummenbildung.

⟨NVar2⟩ = 0 :Prüfsummenbildung erfolgte korrekt
⟨NVar2⟩ ⟨⟩ 0 : Fehler

Bei Syntax-Fehlern wird der BASIC-Fehler # 38 gemeldet.

8.8 CALL 20

Lesen, Setzen oder Löschen eines Bits in einem String. Die Bits werden von links nach rechts beginnend mit 0 durchnumeriert. Mit Hilfe dieses CALL kann auf kleinstem Raum eine Reihe von Merkern eingerichtet werden.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨SVar⟩, ⟨NumPar2⟩ [,⟨NVar⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 20).

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

⟨NumPar1⟩ : Funktionsnummer:

Nummer	Funktion
0	Ermitteln des ersten gesetzten Bits in ⟨SVar⟩ ab der in ⟨NumPar2⟩ vorgegebenen Bitposition. Die ermittelte Adresse wird in ⟨NVar⟩ zur Verfügung gestellt.
1	Das Bit, dessen Position in ⟨NumPar2⟩ vorgegeben ist, wird gesetzt (erhält den Wert 1). Bei dieser Funktion muß ⟨NVar⟩ nicht codiert sein.
2	Das Bit, dessen Adresse in ⟨NumPar2⟩ vorgegeben ist, wird gelöscht (erhält den Wert 0). Bei dieser Funktion muß ⟨NVar⟩ nicht codiert sein.

⟨SVar⟩ : String-Variable, in der Bits bearbeitet werden sollen. Die maximale Stringlänge beträgt 4096 Byte.

⟨NumPar2⟩ : Bit-Position in ⟨SVar⟩ (von links mit 0 beginnend - Bedeutung abhängig von der in ⟨NumPar1⟩ vorgegebenen Funktion):

⟨NumPar1⟩	Bedeutung
0	Position des ersten Bits, ab dem der Vergleich gestartet werden soll.
1	Adresse des zu setzenden Bits.
2	Adresse des zu löschenden Bits.

Aufgrund der Begrenzung von ⟨SVar⟩ auf eine Stringlänge von 4096 Byte kann die Bit-Position ⟨NumPar2⟩ die Werte 0 bis 32767 annehmen

Die Call-Anweisungen

<NVar> : (nur erforderlich, wenn <NumPar1> = 0) In dieser Variablen wird die Adresse des ersten gesetzten Bits übergeben. Wird kein gesetztes Bit gefunden, erhält <NVar> den Wert -1.

- Beispiel:

  Suchen des ersten gesetzten Bits im String A$ ab Position A. Die Position des ersten gesetzten Bits wird in B erwartet. Anschließend wird das erste gesetzte Bit gelöscht.

```
:
105 CALL 20,0,A$,A,B
110 IF B=-1 GOTO 9000 /* KEIN BIT IN A$ GES.
115 LET F=2           /*FUNKT. BIT LÖSCHEN
120 CALL 20,F,A$,B    /* BIT LÖSCHEN
```

Fehlermöglichkeit: ERROR # 38

Bedeutung:

Vorgegebene Bit-Adresse negativ.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.9 CALL 21

Umwandlung des Inhaltes einer beliebigen Variablen in eine Dualzahl.

Syntax:

CALL <NAusdr>,{<NVar1> / <Dezzahl1>},<SVar>[,{<NVar2> / <Dezzahl2>}]

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 21).

<NVar1> : Der Inhalt dieser Dezimalzahl wird in eine <Dezzahl1> 1 oder 2 Byte große Dualzahl umgewandelt und darf nicht größer als 255 (1 Byte) oder 65535 F (2 Byte) und negativ sein.

<SVar> : In dieser String-Variablen wird die Dualzahl (ohne anschließendes Grenzzeichen) abgestellt. Für 2 Byte Umwandlung muß <SVar> für mindestens 2 Byte dimensioniert sein.

<NVar2> <Dezzahl2> : Byte-Adresse, ab der die Dualzahl in <SVar> abgestellt werden soll.
- Ist die Byte-Adresse positiv, muß der Wert von <NVar2> im Bereich 1 bis dimensionierte Länge von <SVar>-1 liegen und ungerade sein. Bei geradem Wert wird der nächstkleinere ungerade eingesetzt.
  Mit solch einer positiven Byte-Adresse wird der Inhalt von <NVar1>/<Dezzahl1> in eine 2 Byte große Dualzahl umgewandelt.
- Ist die Byte Adresse negativ, wird mit dem entsprechenden positiven Displacement der Inhalt von <NVar1>/<Dezzahl1> in eine 1 Byte große Dualzahl umgewandelt.
  Die Byte-Adresse muß im Bereich von 1 bis dimensionierte Länge von <SVar> liegen und darf einen beliebigen Wert annehmen.

* Ist <NVar1>/<Dezzahl1> nicht angegeben oder =0, wird 1 eingesetzt und 2 Byte Umwandlung durchgeführt.

Die CALL-Anweisungen

- Beispiel:

  - Abstellen des Inhaltes von A als Dualzahl in A$ ab Byte 9:

```
:
110 CALL 21,A,A$,9  /* WANDELN IN 2 BYTE DUALZAHL UND
                       ABSTELLEN
```

  - Abstellen des Inhaltes von A als Dualzahl in A$ im Byte 9.

```
:
110 CALL 21,A,A$,-9 /* WANDELN IN 1 BYTE DUALZAHL UND
                       ABSTELLEN
```

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Zahl ist > als 65535 für 2 Byte Umwandlung
- Zahl ist > als 255 für 1 Byte Umwandlung
- Die Dualzahl ist negativ
- Byte-Adresse ist > als die dimensionierte Länge von <SVar> oder <Svar>-1 für 2 Byte Umwandlung

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.10 CALL 22

Umwandeln und Übertragen einer 2 Byte großen Dualzahl aus einem String in eine beliebige numerische Variable.

Syntax:

CALL ⟨NAusdr⟩,{⟨NVar1⟩ / ⟨Dezzahl1⟩},{⟨SVar⟩ / ⟨SLit⟩},⟨NVar2⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 22).

⟨NVar1⟩ : Byte-Adresse, ab der die zu übertragende
⟨Dezzahl1⟩ Dualzahl in ⟨SVar⟩/⟨SLit⟩ steht.
- Ist die Byte-Adresse positiv muß der Wert von ⟨NVar1⟩ im Bereich von 1 bis dimensionierte Länge von ⟨SVar⟩-1 liegen und ungerade sein. Bei geradem Wert wird der nächstkleinere ungerade eingesetzt.
  Mit solch einer positiven Byte-Adresse werden aus ⟨SVar⟩/⟨SLit⟩ 2 Byte in eine numerische Variable übertragen.
- Ist die Byte-Adresse negativ, wird mit dem entsprechenden positiven Displacement 1 Byte aus ⟨SVar1⟩/⟨SLit⟩ in eine numerische Variable umgewandelt.
  Die Byte-Adresse muß im Bereich von 1 bis dimensionierte Länge von ⟨SVar⟩ liegen und kann einen beliebigen Wert annehmen.

Ist ⟨NVar1⟩/⟨Dezzahl⟩ nicht angegeben oder =0, wird 1 eingesetzt und 2 Byte Umwandlung durchgeführt.

⟨SVar⟩ : Aus diesem String wird ab der vorgegebenen
⟨SLit⟩ Adresse der Inhalt von 2 oder 1 Byte in eine numerische Variable übernommen.

⟨NVar2⟩ : Zielvariable, die den in die entsprechende Darstellung umgewandelten Wert der Dualzahl aus ⟨SVar⟩/⟨SLit⟩ aufnimmt.

Die CALL-Anweisungen

- Beispiel:

  - Übernahme der in A$ ab Byte 5 stehenden Dualzahl in die Variable F.

```
:
110 CALL 22,5,A$,F   /* 2 BYTE ÜBERNEHMEN
```

  - Übernahme der in A$ im Byte 5 stehenden Dualzahl in die Variable F.

```
:
110 CALL 22,-5,A$,F  /* 1 BYTE ÜBERNEHMEN
```

Fehlermöglichkeiten:

BASIC-Fehler	Bedeutung
15	Arithmetischer Überlauf. Ein Wert > 7999 soll in eine 1%-Variable übernommen werden.
38	Inhalt von <NVar1>/<Dezzahl> ist > als die dimensionierte Länge von <SVar> oder <SVar>-1 für 2 Byte Übernahme.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.11 CALL 23

Suchen eines Teilstrings in einem String. Beide Suchrichtungen - von links nach rechts bzw. von rechts nach links - sind möglich.

Syntax:

CALL ⟨NAusdr⟩, ⟨String1⟩, ⟨String2⟩, ⟨NumPar1⟩, ⟨NVar⟩
[,⟨NumPar2⟩ [,⟨NumPar3⟩]]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 23).

⟨String1⟩ : In dieser String-Variablen wird der in ⟨String2⟩ zu suchende Teilstring vorgegeben.

⟨String2⟩ : In dieser String-Variablen wird der in ⟨String1⟩ vorgegebene Teilstring gesucht. Grenzzeichen beenden den Suchvorgang nicht!

⟨NumPar1⟩ : Byteadresse, ab der in ⟨String2⟩ mit dem Suchen begonnen werden soll. Der Wert von ⟨NumPar1⟩ muß im Bereich von 1 bis Länge von ⟨String2⟩ - Länge von ⟨String1⟩ liegen.

⟨NVar⟩ : In dieser Variablen wird nach der Durchführung der Anweisung die Anfangsadresse des Teilstrings in ⟨String2⟩ übergeben. Ist der gesuchte Sub-String nicht in ⟨String2⟩ vorhanden, erhält ⟨NVar⟩ den Wert -1.

⟨NumPar2⟩ : Mit dieser Ausgabe wird festgelegt, wie oft der Suchvorgang wiederholt werden soll; d.h. bei Angabe des Wertes N wird der Suchvorgang erst nach N+1 maligem Auffinden des gesuchten Teilstrings beendet und die Byteadresse an ⟨NVar⟩ übergeben.
Nur ganze, positive Werte sind sinnvoll.
⟨NumPar2⟩ darf nicht größer als 1% dimensioniert werden.

Die CALL-Anweisungen

⟨NumPar3⟩ : Wert	Bedeutung: gesucht wird ein String gleicher Länge wie in ⟨String1⟩ und
0/4	= gleich dem String in ⟨String1⟩
1/5	⟨= kleiner als oder gleich dem String in ⟨String1⟩
2/6	⟩= größer als oder gleich dem String in ⟨String1⟩
3/7	⟨⟩ ungleich dem String in ⟨String1⟩

Größer oder kleiner beim Stringvergleich bedeutet größerer oder kleinerer Wert der ASCII-Codierung, wobei die Zeichen der Strings einzeln, bei Wert 0 bis 3 von links beginnend, bzw. bei Wert 4 bis 7 von rechts nach links, verglichen werden.

- Beispiele:

Suchen der Zeichenkombination "/*" in dem String A$ ab Byte 1:

```
Inhalt A$: "1234505555/*003004005"
:
100 LET B$="/*"        /* VERGLEICHSWERT
110 CALL 23,B$,A$,1,B /* SUCHEN

Ergebnis: B=11
```

Suchen des sechsten dreistelligen Teilstrings kleiner oder gleich "ALT" in dem angegebenen String von links beim ersten Byte beginnend:

```
110 CALL 23,"ALT","ALTNEU123ALT!456",1,B,5,1

Ergebnis: B=13
```

(Die Teilstrings, die auf "kleiner oder gleich" geprüft wurden sind:
ALT,LTN,TNE,NEU,EU1,U12,123,23A,3AL,ALT,LT!,T!4,!45;
die unterstrichenen sind die sechs gesuchten.)

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Suchen des ersten Teilstrings "ALT" im String des vorigen Beispiels, wobei die Suche bei Byte 14 beginnend von rechts nach links erfolgen soll:

```
210 CALL 23;"ALT","ALTNEU123ALT!456",14,B,0,4
```

Ergebnis: B=10
(Die geprüften Teilstrings sind: T!4,LT!,ALT)

Fehlermöglichkeit: ERROR #38

Bedeutung:

Byte-Adresse ist kleiner als 1 oder größer als die dimensionierte Länge von ⟨String2⟩.

Die CALL-Anweisungen

8.12 CALL 24

Umkehrung des Inhaltes eines Strings. Dadurch wird das letzte Zeichen im String zum ersten Zeichen, das vorletzte zum zweiten Zeichen usw.

Syntax:

CALL ⟨NAusdr⟩, ⟨SVar⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 24).

⟨SVar⟩ : Der Inhalt dieser String-Variablen wird so umgeordnet, daß das erste Zeichen in das letzte Byte, das zweite Zeichen in das vorletzte Byte usw. übertragen wird. Tritt ein Grenzzeichen auf, wird der String nach links mit Leerzeichen aufgefüllt.

- Beispiel:

Umkehrung des Inhaltes von A$.

Inhalt von A$ vorher:

```
                G
|1|2|3|4|5|6|7|8|9|0|Z|1|2|3|4|5|
```

```
:
100 CALL 24,A$
```

Inhalt von A$ nachher:

```
| | | | | | |0|9|8|7|6|5|4|3|2|1|
```

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.13 CALL 25

Datumsumwandlung in Industriedatum und umgekehrt. Es bestehen die folgenden Möglichkeiten:

- Umwandlung des Systemdatums in Industriedatum.
- Umwandlung eines vom Anwenderprogramm vorgegebenen Datums in Industriedatum.
- Umwandlung eines vom Anwenderprogramm vorgegebenen Industriedatums in ein Datum in der Form TTMMJJ.

Syntax:

CALL ⟨NAusdr⟩, ⟨SVar⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 25).

⟨SVar⟩ : String-Variable, in der das umzuwandelnde Datum vorgegeben wird. Ist ⟨SVar⟩ ein Leerstring, wird das Systemdatum umgewandelt. ⟨SVar⟩ muß für mindestens 23 Byte dimensioniert sein.
Nach Ausführung der Anweisung steht in ⟨SVar⟩ von Byte 1 bis 5 das Industriedatum oder von Byte 1 bis 6 das Datum mit dem Format TTMMJJ.

Wird in ⟨SVar⟩ ein Datum vorgegeben, werden 2 Formatarten akzeptiert:

- "MMM TT JJJJ"
- "TTMMJJ"

- zu Format 1

Die drei Angaben MMM TT JJJJ sind jeweils durch ein Leerzeichen voneinander zu trennen.

MMM = Monat: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.
TT = Tag
JJJJ = Jahr

Die CALL-Anweisungen

- zu Format 2

TTMMJJ - Datumsdarstellung in 6 Bytes.

TT = Tag
MM = Monat (01 - 12)
JJ = Jahr

Das hiernach aufbereitete Industriedatum hat folgendes Format:

"JJTTT"

JJ = Jahr
TTT = laufender Tag im Jahr

Bei der letzten Umwandlungsmöglichkeit des CALL 25 wird vom Anwenderprogramm ein Industriedatum in der oben beschriebenen Form in ein 8 Byte Datum in der Form TTMMJJ umgewandelt.

Bei den Umwandlungen wird überprüft, ob das vorgegebene Datum zulässig ist. Schaltjahre werden bei der Prüfung berücksichtigt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

- Beispiel:

  Umwandlung des Systemdatums in Industriedatum.

```
  :
100  DIM A$(25)        /* DIMENSIONIERUNG
  :
200 LET A$= ""         /* <SVar> = LEERSTRING
210 CALL 25,A$         /* DATUM AUFBEREITEN
```

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Datum in falschem Format vorgegeben.
- Unmögliches Datum.
- <SVar> < 23 Bytes dimensioniert.

Die CALL-Anweisungen

8.14 CALL 26

Selektive Zeichenersetzung in Stringvariablen.

Syntax:

CALL ⟨NAusdr⟩,⟨NumPar⟩,⟨String1⟩,⟨SVar⟩,⟨String2⟩,⟨String3⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 26).

⟨NumPar⟩ : Funktionsnummer = 0

⟨String1⟩ : Zu verändernder String. Der veränderte Inhalt wird nach ⟨SVar⟩ übertragen, während ⟨String1⟩ unverändert bleibt.

⟨SVar⟩ : Stringvariable, in die der veränderte Inhalt von ⟨String1⟩ übertragen werden soll.

⟨String2⟩ : Zeichen, die beim Auftreten in ⟨SVar⟩ durch die entsprechenden Zeichen von ⟨String3⟩ ersetzt werden sollen.

⟨String3⟩ : Einzusetzende Zeichen

⟨String1⟩ und ⟨SVar⟩ müssen gleich dimensioniert sein. ⟨String3⟩ muß die gleiche Länge wie ⟨String2⟩ haben, da durch die beiden Strings die Ersetzungstabelle gebildet wird. Ihre Länge darf 256 nicht überschreiten.

- Beispiel:

  Ersetzen der Zeichen 1, 3 und 7 durch A, B und C im String A$ und Übertragen des Inhalts nach B$.

```
10 DIM A$(20),B$(20)
:
50 LET A$="1234567890",A$
:
200 CALL 26,F,A$,B$,"137","ABC"
```

  Inhalt von B$: A2B456C890A2B456C890

Call 26, k, C$, C2$, N$, N1$

i% k = kode = ϕ
A C$ = te converteren string
A C2$ = string waarin geconverteerde info komt te staan.
A N$ = BRON string met mogelijke tekens
A N1$ = bestemming waarin vervangingstekens staan.

VOOR IBM met bovenpolsing

N$ = 0123456789üABCDEFGHI öjklMNOPQR
N1$ = 0123456789 0123456789 0123456789
DEC 0-9 ; pos 0-9 ; neg 0-9 !

Die CALL-Anweisungen

8.15 CALL 28

Konvertierung von Binärwerten in Zahlen eines beliebigen Zahlensystems zur Basis 2 bis 36 und Darstellung der Ergebnisse in druckbarer Form (ASCII) .

Syntax:

CALL ⟨NAusdr⟩, ⟨String1⟩, ⟨NumPar1⟩, ⟨SVar⟩, ⟨NumPar2⟩,
⟨NumPar3⟩, ⟨NumPar4⟩, ⟨NumPar5⟩, [⟨NumPar6⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 28).

⟨String1⟩ : Vorgabe des Binärwerts in einem oder zwei Worten.

⟨NumPar1⟩ : Vorgabe des Displacements (in Bytes) in Bezug zum Anfang von ⟨String1⟩, bei dem der Binärwert in ⟨String1⟩ beginnt.

⟨SVar⟩ : Nimmt die konvertierte Zahl auf.

⟨NumPar2⟩ : Vorgabe des Displacements in ⟨SVar⟩, ab dessen Position die konvertierte Zahl druckbar abgelegt werden soll.

⟨NumPar3⟩ : Vorgabe der Basis (2-36) des Zahlensystem, in das der Binärwert konvertiert werden soll.

⟨NumPar4⟩ : Größe des Binärwerts in Worten (1 oder 2).

⟨NumPar5⟩ : Vorgabe der Mindestanzahl von in ⟨SVar⟩ abzustellenden Ziffern.
Ergibt die konvertierte Zahl weniger Ziffern als vorgegeben, werden führende Blanks vorangestellt. Bei negativem Vorgabewert werden führende Nullen statt Blanks vorangestellt. Übersteigt die Ziffernzahl der konvertierten Zahl den vorgegebenen Wert, wird trotzdem die ganze Zahl abgestellt, soweit dies die dimensionierte Länge von ⟨SVar⟩ zuläßt.

⟨NumPar6⟩ : =1: Umwandlung des durch ⟨NumPar1⟩ referierten Bytes. ⟨NumPar4⟩ bleibt unberücksichtigt.
=0: Keine Auswirkung.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Anmerkung:

Displacements können ganzzahlige, positive Werte annehmen. Der Wert 0 wird wie der Wert 1 als Displacement 1 interpretiert.

Fehlermöglichkeiten:

ERR#	Bedeutung
15	<SVar> konnte nicht alle abzustellenden Ziffern aufnehmen. Der Rest der Zahl wurde unterdrückt.
38	- nicht alle Variablen vorgegeben - falsche Parameter vorgegeben

Die CALL-Anweisungen

8.16 CALL 29

Mit diesem CALL kann man im Format 1 das PROGRAM ABORT FLAG (PAF) und die PORT CONTROL AREA (PCA) beeinflussen. Im Format 2 wird ermöglicht, festzustellen, ob ein Port inaktiv ist oder unter TAMOS bzw. SCOPE aktiv ist.

CALL 29 - Format 1 (Bearbeitung der Port Control-Area)

Der Eintrag im PAF bestimmt, was nach einem Programmabbruch geschehen muß. Ein BASIC-Programm kann damit in einzelne Abschnitte unterteilt werden, die unterschiedliche Reaktionen bei einem Programmabbruch erforderlich machen. PAF und PCA bilden einen Bereich in der Datei TF.PORT (je einer für jeden BA), der in den ersten beiden Bytes das PAF, in den folgenden 18 Bytes den Namen des Programms (<LU>/<Progname>), das bei Programmabbruch gestartet wird, wenn PAF=2 ist, und in 492 folgenden Bytes frei verwendbare Informationen enthält.

Die PCA kann mit unterschiedlichen Variablentypen in beliebiger Reihenfolge gelesen und beschrieben werden (analog zur COMMON-Area mit CALL 2/3). Numerische Werte können direkt, ohne Zwischenspeichern auf Stringvariable, an das RESTART-Programm oder an das Reparaturprogramm weitergereicht werden. Restriktionen analog zur COMMON-Area: der CALL 29 darf maximal 12 Variable (insgesamt) verwenden. Einschließlich des PAF's können 512 Byte in die PCA geschrieben bzw. aus ihr gelesen werden. Die Übertragung der Variablen geschieht wortorientiert.

Für das Format 1 ergibt sich folgende Aufrufschnittstelle:

$$\text{CALL 29 , } \left\{\begin{matrix}\langle\text{NVar1}\rangle\\ \langle\text{Ganzzahl1}\rangle\end{matrix}\right\} , \left\{\begin{matrix}\langle\text{NVar2}\rangle\\ \langle\text{Ganzzahl2}\rangle\end{matrix}\right\} , \left\{\begin{matrix}\langle\text{NVar3}\rangle\\ \langle\text{SVar}\rangle\end{matrix}\right\}^{10}$$

<NVar1> bzw. <Ganzzahl1> : Funktionsnummer:
1 = PAF und PCA lesen
2 = PAF und PCA schreiben

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

<NVar2> bzw. <Ganzzahl2>: Statusvariable
Bei Fkt. 1 ist dies die Zielvariable für den Inhalt von PAF.
Bei Fkt. 2 ist dies die Quellvariable, d.h. hier muß vorher der zu schreibende Status abgestellt werden. Konsequenz bei Programmabbruch:
0 = Rücksprung in Selektor
1 = Auswahl, ob Neustart oder Rücksprung in Selektor
2 = Wenn möglich Aufruf des Reparaturprogramms, das in PCA eingetragen ist, sonst Rekonstruktion der 2. Generation
3 = Unbedingt Rekonstruktion 2. Generation erforderlich
4 = Neustart (ohne Frage)

<NVar3> bzw. <SVar>: Bei Fkt. 1 ist dies die Zielvariable für den Inhalt von PCA.
Bei Fkt. 2 wird der Inhalt von <String> nach PCA geschrieben. In dem Fall, daß ein Reparaturprogramm aufgerufen werden soll, enthält der String Benutzerinformationen, die an ein Reparatur- oder Restart-Programm über die PCA (Port Control Area) weitergegeben werden sollen. Die ersten 18 Bytes müssen den Namen des Reparaturprogramms enthalten (<LU>/Progname>).
Die Benutzerinformation kann maximal 510 Byte lang sein und darf auch in mehrere num. und alphanum. Variable zerfallen. Der CALL 29 erlaubt entsprechend dem oben beschriebenen Format 10 Variable als Benutzerinformation. (Die CALL-Schnittstelle erlaubt insgesamt die Übergabe von 12 Variablen, wovon bei dem beschriebenen Format des CALL 29 bereits 2 fest vergeben sind, so daß für die Benutzerinformation 10 Variable zur Verfügung stehen.)

Achtung:

Ein Programm mit dieser Abbruchbehandlung darf nur über TAMOS aufgerufen werden.

Die CALL-Anweisungen

CALL 29 - Format 2 (TAMOS-Merker lesen)

In diesem Format ermöglicht der CALL 29 festzustellen, ob ein Port inaktiv ist oder unter TAMOS bzw. SCOPE aktiv ist.
Der CALL 29 - Format 2 besitzt folgende Aufrufschnittstelle:

CALL 29, ⟨NVar1⟩, ⟨NVar2⟩

⟨NVar1⟩ : Funktionsnummer = 4
Nach der Durchführung des CALL 29 enthält ⟨NVar1⟩ den Port-Status:
-1 = Port nicht aktiv
0 = Port unter SCOPE aktiv
1 = Port unter TAMOS aktiv.

⟨NVar2⟩ : Portnummer. Bei der Vorgabe einer unzulässigen Portnummer wird BASIC-Fehler 38 erzeugt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.17 CALL 31

Codeumwandlung mit Hilfe von Code-Tabellen

Syntax:

CALL ⟨NAusdr⟩, ⟨NVar1⟩, ⟨String⟩, ⟨SVar1⟩, ⟨SVar2⟩, ⟨NVar2⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 31).

⟨NVar1⟩ : Vorgabe der Anzahl der zu konvertierenden Bytes
Nach Ausführung enthält diese Variable die Anzahl der tatsächlich konvertierten Bytes.

⟨String⟩ : Quellstring, der die zu konvertierenden Zeichen enthalten muß.

⟨SVar1⟩ : Zielstring, der die konvertierten Zeichen nach korrekter Ausführung enthält.

⟨SVar2⟩ : Hier muß die Code-Tabelle zur Umformung vorgegeben werden. Diese Variable muß so groß dimensioniert sein, daß sie die ganze Code-Tabelle aufnehmen kann.

⟨NVar2⟩ : Statusvariable
0 = fehlerfreie Ausführung
1 = ⟨NVar1⟩ ist nicht positiv oder zu groß

- Beispiel:

```
  :
 50 DIM C$(256)
  :
800 OPEN #3,"ASCII.EBCDIC"
810 MATREAD #3,0;C$;          /* CODE-TABELLE LESEN
820 CLOSE #3
830 CALL 31,A,Q$,Z$,C$,N2     /* ASCII -> EBCDIC
```

Die CALL-Anweisungen

8.18 CALL 32

Datumsbearbeitung:

Gültigkeitsprüfung, Konvertierung, Systemzeit lesen oder Systemzeit setzen.

Syntax:

Format 1 für Funktion 0

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨String⟩ [,⟨NumPar2⟩], ⟨NVar⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 32).

⟨NumPar1⟩ : Funktionsnummer (in diesem Fall 0).

⟨String⟩ : String, der das zu prüfende Datum enthalten muß.

⟨NumPar2⟩ : Angabe des Displacements des Datums in ⟨String⟩. Ohne Angabe wird der Wert 1 übernommen.

⟨NVar⟩ : Statusvariable.

Format 2 für die Funktionen 1-5 und 10

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨String⟩ [,⟨NumPar2⟩], ⟨SVar⟩
[,⟨NumPar3⟩], ⟨NVar⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 32).

⟨NumPar1⟩ : Funktionsnummer (in diesem Fall 1-5 oder 10).

⟨String⟩ : Vorgabe des zu konvertierenden Datums.

⟨NumPar2⟩ : Angabe des Displacements des Datums in ⟨String⟩. Ohne Angabe wird der Wert 1 übernommen.

⟨SVar⟩ : Zielstring für das konvertierte Datum.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

<NumPar3> : Angabe des Displacements des Datums in <SVar>. Ohne Angabe wird der Wert 1 übernommen.

<NVar> : Statusvariable

Format 3 für die Funktionen 8 und 9

CALL <NAusdr>, <NumPar1>, <SVar1> [,<NumPar2>]
[,<SVar2> [,<NumPar3>]], <NVar>

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 32).

<NumPar1> : Funktionsnummer (in diesem Fall 8 oder 9).

<SVar1> : bei Fkt. 8 : Zielstring für Ergebnisdatum
bei Fkt. 9 : Vorgabe des Datums.

<NumPar2> : Angabe des Displacements des Datums in <SVar1>. Ohne Angabe wird der Wert 1 übernommen.

<SVar2> : bei Fkt. 8 :Vorgabe der internen Systemzeit(SD), die in EAD umgewandelt wird. Ohne Angabe Übernahme der Systemzeit.

bei Fkt. 9 :Ergebnisdatum
Ohne Codierung nur Übernahme als interne Systemzeit

<NumPar3> : Angabe des Displacements des Datums in <SVar2>. Ohne Angabe wird der Wert 1 übernommen.

<NVar> : Statusvariable.

Die CALL-Anweisungen

Funktionen:

Fkt.-Nr.	Eing.	Ausg.	Fkt.-Beschreibung
0	IAD	IAD	Gültigkeitsprüfung
1	ED	IAD	Konvertierung, Gültigkeitsprüfung
2	IAD	ED	Konvertierung
3	IAD	IBD	"
4	IBD	IAD	"
5	IAD	KW	"
8	-	EAD	interne Systemzeit lesen
	SD	EAD	Konvertierung
9	EAD	-	Gültigkeitsprüfung (mit Uhrzeit), Systemzeit setzen
	EAD	SD	Konvertierung
10	IAD	ED	Konvertierung, Trennzeichen unterdrückt

IAD : Interne ASCII-Darstellung
Datumsdarstellung in der Form "JJMMTT" in 6 Bytes ASCII-Code.

ED : Externe Darstellung
Aufbereitung eines Datums in der Form, die durch den Formatkontrollstring im System festgelegt ist, d.h. "TT", "MM" und "JJ" in einer bestimmten Reihenfolge. Dieses Format kann entweder als 6 Byte String oder als 8 Byte String mit Trennzeichen dargestellt werden.

Als Trennzeichen sind hier zugelassen:

Ø | , | - | . | / | : | ;

Z.B.: "MM-TT-JJ" oder "MMTTJJ"

Diese Formatdarstellung gilt für Funktion:

1 ED -> IAD

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

IBD : Interne binäre Darstellung
Speicherung eines Datums in 3 Bytes in der Form "JJMMTT", wobei Jahre, Monate und Tage binär verschlüsselt jeweils in einem Byte abgestellt werden.

SD : Systemzeit-Darstellung
Byte 1,2: Stunden seit 1.1.1980 (binär)
Byte 3,4: Teile der aktuellen Stunde in 10tel Sekunden (binär)

EAD: Erweiterte ASCII-Darstellung
Die Darstellung erfolgt in 12 Bytes ASCII-Code in der Form "JJMMTThhmmss".

KW : Das Datum wird in die entsprechende Kalenderwoche konvertiert. Die Ausgabe erfolgt in 4 Bytes ASCII-Code in der Form "JJWW".

Statusmeldungen in <NVar>:

Status	Bedeutung
0	Funktion fehlerfrei durchgeführt
1	ungültiges Datum erkannt
2	- angegebenes Displacement < 1 - zu wenig Platz im Zielstring zur Aufnahme des Datums - zu wenig Platz im Quellstring für ein gültiges Datum
3	Nur bei Funktion 0, 1 oder 9 Das Datum 999999 wurde vorgegeben bei Fkt. 9 wird das Systemdatum bei dieser Angabe nicht verändert.
4	Nur bei Funktion 0, 1 oder 9 6 Blanks wurden als Datum vorgegeben bei Fkt. 9 wird das Systemdatum bei dieser Angabe nicht verändert.

Fehlermöglichkeit: ERROR #38:

- unvollständige Parameterliste
- ungültige Funktionsnummer
- ungültige Kennung für die Darstellung eines Zeitraums.

Die CALL-Anweisungen

8.19 CALL 35

Magnetplattenkonfiguration

Der CALL 35 liefert Informationen aus der Datei CONFIG über die derzeit konfigurierten Magnetplatten.

Syntax:

CALL ⟨NAusdr⟩, ⟨SVar⟩, ⟨NVar⟩

⟨NAusdr⟩ Nummer des CALL-Unterprogramms, in diesem Fall 35

⟨SVar⟩ String, in dem die Informationen abgestellt werden

⟨NVar⟩ Statusvariable
0 - Funktion wurde fehlerfrei durchgeführt
1 - SVar ist zu klein dimensioniert

In SVar werden nach der Durchführung der Funktion je nach konfiguriertem Laufwerkstyp

- die Länge des Eintrags für diesen Typ im String
- die Anzahl der Physikalischen Einheiten je Laufwerk dieses Typs
- und je Physikalischer Einheit eine Fest-/Wechselkennung

übergeben

Unterschieden werden folgende Typen:

- 0 = SMD/FDU
- 1 = CA
- 2 = CMD
- 3 = CDD
- 4 = FD 8
- 5 = MDL
- 6 = FD 5 1/4
- 8 = CDH

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

SVar hat nach dem Aufruf des CALL folgenden Inhalt:

Byte	Inhalt
1,2	erster konfigurierter Laufwerkstyp
3,4	Eintragslänge in Byte für diesen Typ
5,6	- Anzahl Physikalische Einheiten für Laufwerk 0
7,k	Fest-/Wechselplattenkennung der PU's (je PU ein Byte, F=Fest, W=Wechsel)
k+1,k+2	- Anzahl Physikalische Einheiten für Laufwerk 1 (falls vorhanden)
k+3,l	Fest-/Wechselplattenkennung der PU's
l+1,m	- analog für alle weiteren Laufwerke dieses Typs
:	:
m+1,m+2	zweiter konfigurierter Laufwerkstyp
m+3,n	analog zum ersten Laufwerkstyp
:	:
:	:
n+1,p	alle weiteren aktiven Laufwerkstypen
:	:
:	:
p+1	Grenzzeichen

Hinweis : Die Eintragslänge in Byte für den jeweiligen konfigurierten Laufwerkstyp bezieht sich auf den gesamten Eintrag, einschließlich der 2 Byte für den Laufwerkstyp und der 2 Byte für die Eintragslänge selbst.

Die CALL-Anweisungen

- Beispiel:

  Es besteht folgende Plattenkonfiguration:

Drivenummer	Typ	Physikalische Einheiten/Drive
0	SMD	2
1	SMD	2
2	FDU	4
0	CA	1
1	CA	1
0	CMD	2
0	CDD	2
0	FD 8	1
1	FD 8	1
0	MDL	2
4	FD 5	1

  Nach Durchführung des CALL 35 bei der obigen Konfiguration hat ⟨SVar⟩ folgenden Inhalt:

Byte	!	Inhalt	
1 , 2	!	00	SMD
3 , 4	!	18	Eintragslänge
5 , 6	!	02	PU's pro Laufwerk 0
7 , 8	!	WW	Wechselplatten
9 ,10	!	02	PU's pro Laufwerk 1
11,12	!	WW	Wechselplatten
13,14	!	04	PU's pro Laufwerk 2
15,18	!	FFFF	Festplatten
19,20	!	01	CA
21,22	!	08	Eintragslänge
23,24	!	02	PU's
25,26	!	FW	Fest/Wechsel
27,28	!	02	CMD
29,30	!	08	Eintragslänge
31,32	!	02	PU's
33,34	!	FW	Fest/Wechsel

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Byte	Inhalt	
35,36	03	CDD
37,38	08	Eintragslänge
39,40	02	PU's
41,42	FF	Festplatte
43,44	04	FD 8
45,46	10	Eintragslänge
47,48	01	PU
49,49	W	Wechsel
50,51	01	PU
52,52	W	Wechsel
53,54	05	MDL
55,56	08	Eintragslänge
57,58	02	PU's
59,60	FF	Festplatte
61,62	06	FD 5
63,64	07	Eintragslänge
65,66	01	PU
67,67	W	Wechsel

Die CALL-Anweisungen

8.19-1 CALL 43

Dieser CALL dient zur Unterstützung der Vorgangsbearbeitung. Er übernimmt die Parameter und stellt sie in die Vorgangsliste ab. Zusätzlich werden das Eintragsdatum und der Absender festgehalten. Der Vorgang wird als wartend klassifiziert.

Syntax:

CALL <NAusdr>, <NVar1>, <NVar2>, <NVar3>, <SVar1>, <SVar2>, <SVar3>, <SVar4>, <SVar5>, <NVar4>, <NVar5> [, <SVar6>]

<Nausdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 43).

<NVar1> : Diese Variable enthält die Funktionsnummer:

1: Vorgang anlegen
2: Vorgang lesen

<NVar2> : Diese Variable enthält die Vorgangsnummer.

Für Funktion 1: Rückgabeparameter
Für Funktion 2: Vorgabeparameter

<NVar3> : Status:

0: Funktion erfolgreich durchgeführt
1: fehlerhafte Parameterleiste
2: ungültige Funktionsnummer
4: LU nicht aktiv, fehlerhafte RDA
5: Vorgangsliste nicht gefunden
6: Vorgang existiert nicht (Funktion 2)
10: Empfänger existiert nicht (Funktion 1)

<SVar1> : Vorgangsbeschreibung, Länge 20 Zeichen

<Svar2> : Programmname/Programmnummer, Länge 18 Zeichen

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

<SVar3> : Programmparameter, Länge 40 Zeichen

(1,5) Runmode
(6,13) Benutzte Logische Einheiten, analog Übergabe der Standardparameter durch TAMOS
(14,19) Selektorparameter, analog Übergabe durch TAMOS
(20,40) for future use

Nicht benutzte Parameter müssen durch Blanks aufgefüllt werden. Ein Grenzzeichen beendet die Übernahme

<SVar4> : Vorgang Status-Flag, Länge 10 Zeichen

(1,1) "1": Vorgang nicht löschbar
(2,10) nicht belegen, for future use

<SVar5> : Termin, Länge 10 Zeichen

Format: YYMMDDHHMM oder Grenzzeichen

<NVar4> : USER-Nr. bzw. OE-Nr. des Empfängers

<NVar5> : Priorität

<SVar6> : Parameter zu freien Verfügung, Länge 350 Zeichen wird im COMMON-Bereich des Anwenderprogramms hinter den TAMOS-Parametern abgestellt

Wenn eine der ersten drei Variablen keine numerische Vari ist, wird BASIC Fehler #38 erzeugt.

Die CALL-Anweisungen

8.19-2 CALL 44

Dieser CALL löscht einen Vorgang aus der Vorgangsliste. Dabei muß die absolute Vorgangsnummer angegeben werden. Vor dem Aufruf der Funktion sollte aus Sicherheitsgründen mit Hilfe der Funktion "Vorgang lesen" (s. CALL 43) einige für den Vorgang signifikante Merkmale überprüft werden.

Syntax:

CALL <NAusdr>, <NVar1>, <NVar2>, <NVar3>

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms, in diesem Fall 44

<NVar1> : Funktionsnummer

1: Vorgang löschen

<NVar2> : Vorgangsnummer (absolute Nummer in der Vorgangsliste)

<NVar3> : Status:

0: Funktion erfolgreich durchgeführt
1: ungültige Funktionsnummer
4: LU nicht aktiv, fehlerhafte RDA
5: Vorgangsliste nicht gefunden
6: Vorgang existiert nicht
9: Vorgang nicht löschbar

Basic-Fehler #38 wird erzeugt, wenn eine der drei Variablen keine numerische Variable ist.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.20 CALL 51

Aufbereitung von Strings mit Oktal-Codes zur Unterstützung der Bildschirm-Programmierung.

Achtung:

CALL 51 ist ausgelegt für Bildschirme mit 27 Zeilen zu je 74 Zeichen (Hazeltine).

Syntax:

CALL ⟨NAusdr⟩, ⟨SVar1⟩, ⟨SVar2⟩, ⟨SVar3⟩, ⟨SVar4⟩, ⟨SVar5⟩, ⟨SVar6⟩, ⟨SVar7⟩, ⟨SVar8⟩, ⟨SVar9⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 51).

⟨SVar1⟩ : String-Variable, Länge 74 Byte.
In den Bytes 1 bis 73 werden bei der Ausführung die Werte 1 bis 73 (201 bis 220, 361 bis 364 und 225 bis 311 oktal) abgestellt. Byte 74 erhält den Wert 0 (200 oktal). Diese Werte dienen zur Vorgabe der Cursorpositionen (Position und Zeile).

⟨SVar2⟩ : String-Variable, Länge 2 Byte.
⟨SVar2⟩ nimmt die Steuerzeichen für Cursorpositionierung (376, 221 oktal) auf.

⟨SVar3⟩ : String-Variable, Länge 6 Byte.
⟨SVar3⟩ dient zur Aufnahme von 6 DEL-Zeichen (377 oktal).

⟨SVar4⟩ : String-Variable, Länge 2 Byte.
⟨SVar4⟩ nimmt bei der Ausführung die Steuerzeichen für Bildschirm löschen (376, 234 oktal) auf.

⟨SVar5⟩ : String-Variable, Länge 2 Byte.
⟨SVar5⟩ dient zur Aufnahme der Steuerzeichen für Vordergrundfelder Löschen (376, 235 oktal).

⟨SVar6⟩ : String-Variable, Länge 2 Byte.
⟨SVar6⟩ dient zur Aufnahme der Steuerzeichen für Start Hintergrundausgabe (376, 231 oktal).

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

⟨SVar7⟩ : String-Variable, Länge 2 Byte.
⟨SVar7⟩ dient zur Aufnahme der Steuerzeichen für Start Vordergrundausgabe (376, 237 oktal).

⟨SVar8⟩ : String-Variable, Länge 2 Byte.
⟨SVar8⟩ dient zur Aufnahme der Steuerzeichens für Zeile einfügen (376, 232 oktal).

⟨SVar9⟩ : String-Variable, Länge 2 Byte.
⟨SVar9⟩ dient zur Aufnahme der Steuerzeichen für Zeile löschen (376, 223 oktal).

Die Anwahl der Bildschirm-Funktionen erfolgt nach der Durchführung des CALL 51 durch die Ausgabe der entsprechenden Strings mit der Anweisung PRINT.

Eine Besonderheit stellen nur die in ⟨SVar3⟩ abgestellten DEL-Zeichen dar: sie dienen dazu, für eine Zeit von 5 bis 6 Millisekunden, die von den Funktionen:

- Bildschirm löschen,
- Vordergrundfelder löschen,
- Zeile einfügen und
- Zeile löschen

benötigt wird, ein erneutes Ansprechen des Bildschirmes zu verhindern.

- Beispiel:

  Durchführung des CALL 51 mit anschließendem:

  - Bildschirm löschen,
  - Cursor auf Position 10 in Zeile 2 positionieren.

```
:
150 CALL 51,N$,N1$,N2$,N3$,N4$,N5$,N6$,N7$,N8$
160 PRINT N3$;N2$;N1$;N$(10,10);N$(2,2);
```

Die CALL-Anweisungen

8.21 CALL 59

Konvertierung von numerischen Daten in gepackte Dezimalzahlen gemäß IBM-Format und umgekehrt. Es können nur ganze Zahlen konvertiert werden. Die beiden Konvertierungsrichtungen unterscheiden sich durch die Syntax:

Format	Syntax
1	CALL ⟨NAusdr⟩, ⟨String⟩, ⟨NVar⟩
2	CALL ⟨NAusdr⟩, ⟨NumPar⟩, ⟨SVar⟩

Format 1 ist bei der Konvertierung aus IBM-Format zu wählen.
Format 2 ist bei der Konvertierung in IBM-Format zu wählen.

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (59).

⟨String⟩ : Gepackte Dezimalzahl gemäß IBM-Format. Jedes Byte enthält zwei BCD-Ziffern, die letzte Tetrade muß das Vorzeichen enthalten: 1100 = +, 1101 = -. Die Ziffern schließen sich rechtsbündig an das Vorzeichen an. Soll eine Zahl mit weniger Ziffern als vorhandene Tetraden aufgenommen werden, müssen die ersten Tetraden 0 enthalten.

⟨NVar⟩ : Numerische Variable (1-4%), die den von IBM- nach BASIC-Format konvertierten Wert aufnimmt. Stehen mehr IBM-Ziffern zur Verfügung, als die Variable an signifikanten Ziffern aufnehmen kann, gehen die überzähligen Ziffern (auf Kosten der Genauigkeit) verloren.

⟨NumPar⟩ : Numerischer Wert, der gemäß IBM-Format gepackt werden soll.

⟨SVar⟩ : String-Variable, die den in IBM-Format gepackten Wert aufnehmen soll. Die dimensionierte Länge darf folgenden durch ⟨NumPar⟩ bestimmten Wert nicht unterschreiten (Nachkommastellen werden nicht berücksichtigt):

INT((Anz. Ziff.+2)/2)

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

- Beispiele:

  - Konvertieren der in einer 2%-Variablen abgestellten Zahl

    +0,123456E+06 = 123456

    in gepackte Darstellung gemäß IBM-Format.

```
10 DIM A$(4),2%,N                /* DIMENSIONIERUNG
 :                               /* AUFBEREITUNG VON N
70 CALL 59,N,A$                  /* KONVERTIERUNG
```

    Inhalt von A$ nach der Konvertierung:

```
0000 0001  0010 0011  0100 0101  0110 1100

  0    1     2    3     4    5     6    +
```

  - Konvertieren der im IBM-Format gepackten Zahl in A$

    +98765432

    in eine 2%-Variable (N).

    Inhalt von A$ (Länge 5 Bytes):

```
0000 1001  1000 0111  0110 0101  0100 0011  0010 1100

10 DIM A$(5),2%,N
 :
70 CALL 59,A$,N
```

    Inhalt nach der Konvertierung:

    +0,987654E+08 = 98765400

    Weil nur in eine 2%-Variable konvertiert wurde (die nur 6 signifikante Ziffern aufnehmen kann), ging die Genauigkeit der beiden letzten Stellen verloren. Es wäre besser in eine 3%-Variable konvertiert worden.

Fehlermöglichkeit: ERROR # 38

- BASIC-Zahl zu groß für IBM-Format-String.
- IBM-Format-String zu lang für 1%-Variable.

Die CALL-Anweisungen

8.22 CALL 60

Konvertierung von numerischen ASCII-Zeichen in 4-Bit-Format laut Umwandlungstabelle. Die Ausgabe erfolgt ebenfalls in einen String (gepackter String).
Durch dieses Unterprogramm können auf externen Datenträgern numerische Daten platzsparend abgestellt werden.

Syntax:

CALL ⟨NAusdr⟩, ⟨String⟩, ⟨SVar⟩ [,⟨NVar⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (60).

⟨String⟩ : Der Inhalt dieses Strings wird in 4-Bit-Format umgewandelt und nach ⟨SVar⟩ übertragen. Dieser String darf nur numerische ASCII-Zeichen und die Sonderzeichen Ø . , + - enthalten.

⟨SVar⟩ : In dieser String-Variablen wird der in 4-Bit-Format konvertierte Inhalt von ⟨String⟩ abgestellt. Die Übertragung in den Ziel-String wird durch eine Nulltetrade abgeschlossen. ⟨SVar⟩ sollte für mindestens halb soviel Bytes dimensioniert sein, wie Zeichen konvertiert werden.
Die Konvertierung wird beendet durch:
- Erreichen des Stringendes von ⟨String⟩.
- Erreichen der dimensionierten Länge in ⟨SVar⟩.

⟨NVar⟩ : Statusvariable
0: Fehlerfreie Durchführung
1: Dimension von ⟨SVar⟩ zu klein
2: Nicht packbares Zeichen in ⟨String⟩

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Umwandlungstabelle:

Zeichen	4-Bit-Code
Begrenzer	0000
0	0001
1	0010
2	0011
:	:
9	1010
Ø	1011
+	1100
,	1101
-	1110
.	1111

- Beispiel:

```
:
100 LET A$="123.45"
105 CALL 60,A$,B$
```

Inhalt von B$ nach der Durchführung des CALL 60:

0010 0011 0100 1111 0101 0110 0000 0000

Fehlermöglichkeit: ERROR #38

Bedeutung:

Es ist ein unzulässiges Zeichen in ⟨String⟩ aufgetreten.

Die CALL-Anweisungen

8.23 CALL 61

Rück-Konvertierung des Inhalts eines Strings, der durch CALL 60 aufbereitet wurde (entpacken).

Syntax:

CALL ⟨NAusdr⟩, ⟨SVar1⟩, ⟨SVar2⟩ [,⟨NVar⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 61).

⟨SVar1⟩ : Der Inhalt dieser String-Variablen kann durch CALL 60 aufbereitet worden sein. Die in 4-Bit-Format abgestellten Ziffern werden in ASCII-Code umgewandelt und nach ⟨SVar2⟩ übertragen.

⟨SVar2⟩ : In dieser String-Variablen werden die in ASCII-Code umgewandelten numerischen Zeichen aus ⟨SVar1⟩ abgestellt. ⟨SVar2⟩ muß für soviel Byte dimensioniert sein, wie Zeichen konvertiert werden.
Die Konvertierung wird beendet durch:
- Auftreten einer Tetrade, in der alle Bit auf 0 gesetzt sind.
- Erreichen der dimensionierten Länge von ⟨SVar1⟩.

⟨NVar⟩ : Statusvariable
0: Fehlerfreie Durchführung
1: Dimension von ⟨SVar2⟩ kleiner als Dimension von ⟨SVar1⟩

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

- Beispiel:

  Konvertierung eines durch CALL 60 aufbereiteten Inhalts der Variablen A$ in die Variable B$.

  Der Inhalt von A$ in binärer Darstellung:

  0010 0011 0100 0101 0110 0001 0001 0000

```
:
10 DIM A$(4),B$(10)      /* DIMENSIONIERUNG
:
200 CALL 61,A$,B$        /* KONVERTIERUNG
```

  Ergebnis: B$="1234500"

  Fehlermöglichkeit: ERROR # 38

  Bedeutung:

  Zielvariable <SVar2> zu klein dimensioniert.

Die CALL-Anweisungen

8.24 CALL 62

Prüfen, ob der Inhalt eines Strings ausschließlich aus Ziffern besteht (Format 1).

Beliebige Syntaxprüfung eines Strings (Format 2).

Format	Syntax
1	CALL ⟨NAusdr⟩,⟨String1⟩, [,⟨NVar2⟩]
2	CALL ⟨NAusdr⟩,⟨String1⟩,⟨NVar1⟩,⟨NVar2⟩,⟨String2⟩ [,⟨String3⟩] [,⟨NumPar1⟩ [,⟨NumPar2⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 62).

⟨String1⟩ : String, dessen Inhalt überprüft wird.

⟨NVar1⟩ : Displacement des ersten zu untersuchenden Zeichens in ⟨String1⟩.
Ist der Wert gleich 0, wird auf Displacement 1 mit der Prüfung begonnen.

⟨NVar2⟩ : Statusvariable.

⟨String2⟩ : Vorgabe einer Menge von Zeichen, die jeweils als erstes Zeichen in ⟨String1⟩ erlaubt sind.
Es gibt zwei Möglichkeiten:
1. Aufzählung aller erlaubten Zeichen (ohne Trennzeichen)
2. ASCII-oktal aufsteigende Zeichenfolge mit Angabe des ersten und letzten Zeichens. Trennzeichen ist der Oktalcode -377 -.

⟨String3⟩ : Vorgabe einer Zeichenmenge wie in ⟨String2⟩ die für die übrigen Zeichen von ⟨String1⟩ gilt.
Fehlt diese Angabe, wird ersatzweise dafür ⟨String2⟩ übernommen.

⟨NumPar1⟩ : Vorgabe der maximalen Länge der zu überprüfenden Zeichenfolge. Dieser Wert muß positiv sein. Ist er gleich 0 oder fehlt er ganz, wird ⟨String1⟩ bis Stringende geprüft.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

〈NumPar2〉 : Vorgabe der minimalen Länge der zu überprüfenden Zeichenfolge. Dieser Wert muß positiv sein.
Ist der Wert gleich 0, wird auch ein Leerstring akzeptiert. Sonst und bei ganz fehlender Angabe wird der Wert 1 angenommen.

Statusmeldungen

Statusnummern werden in 〈NVar2〉 übergeben mit folgender Bedeutung:

0: Kein Fehler aufgetreten
1: Unzulässiges Zeichen in 〈String1〉
2: Grenzzeichen aufgetreten, bevor die minimal zu überprüfende Länge erreicht ist, der String ist zu kurz.
3: Falsche Parameterinhalte:
- numerische Werte sind größer als 65535 oder negativ
- Minimalzahl 〉 Maximalzahl der zu untersuchenden Zeichen
- 〈String1〉 hat mehr als 2000 Zeichen
- Displacement des 1. zu untersuchenden Zeichens 〉 2000
- Displacement des 1. zu untersuchenden Zeichens zeigt außerhalb des 〈SVar1〉/〈SLit1〉
- Minimalzahl der zu prüfenden Zeichen 〉 2000

4: 〈String2〉 oder 〈String3〉 ist leer

Displacementrückgabe

Nach der Syntaxprüfung wird in 〈NVar1〉 in Abhängigkeit vom Status ein Displacement übergeben.

Status	Displacement
0	das des letzten untersuchten Zeichens
1	das des ersten fehlerhaften Zeichens
2	das des Grenzzeichens
3	Eingangswert oder 0, wenn dieser negativ oder größer als 65535 ist
4	Eingangswert

Die CALL-Anweisungen

- Beispiele:

  - Überprüfen des Inhalts von A$ auf Ziffern

```
:
150 CALL 62,A$
```

  - Überprüfen des Inhalts von A$ auf gültigen Dateinamen

```
 :
 90 LET A$ = "11$ABCDEFGHIJ234AA"
100 LET B$ = "$A←377←Z"
110 LET C$ ="0←377←9.A←377←Z"
120 LET A=3
130 LET C=14
140 LET D=0
150 CALL 62,A$,A,B,B$,C$,C,D
160 PRINT A
170 PRINT B
```

  Ergebnis: A=16 : letztes bearbeitetes Zeichen
  B= 0 : Status

Fehlermöglichkeit: ERROR # 38

Bedeutung:

Parameter fehlen oder falsche Variablentypen wurden vorgegeben.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.25 CALL 63

Prüfen, ob der Inhalt eines Strings ein numerischer Wert ist.
Zusätzlich zu den Ziffern 0 bis 9 sind ein führendes Vorzeichen, - oder +, und an beliebiger Stelle ein Dezimalkomma (Dezimalpunkt) zugelasssen.

Syntax:

CALL ⟨NAusdr⟩, ⟨String⟩, [,⟨NVar⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 63).

⟨String⟩ : String, dessen Inhalt daraufhin überprüft wird, ob er einen numerischen Wert darstellt.

⟨NVar⟩ : Statusvariable
0: ⟨String⟩ enthält numerischen Wert
1: ⟨String⟩ enthält keinen numerischen Wert

- Beispiel:

Prüfen des Inhaltes von A$.

```
:
150 CALL 63,A$
```

Fehlermöglichkeit:

Wenn ⟨NVar⟩ nicht codiert ist: ERROR # 38

Bedeutung:

- In ⟨String⟩ wurde ein unzulässiges Zeichen gefunden.
- Nur ein Vorzeichen oder Dezimalpunkt und keine Ziffer in ⟨String⟩.

Anmerkung:

Ein Leerstring führt zu keiner Fehlermeldung.

Die CALL-Anweisungen

8.26 CALL 64

Prüfen, ob ein String ein gültiges Datum enthält. Für das Datum ist folgendes Format vorgeschrieben:

MM.TT.JJ oder MM,TT,JJ

Ist das geprüfte Datum zulässig, wird es umgestellt und in der Form

JJMMTT

im String abgestellt.

Syntax:

CALL ⟨NAusdr⟩, ⟨SVar⟩ [,⟨NVar⟩]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (64).

⟨SVar⟩ : String-Variable, die für mindestens 8 Byte dimensioniert sein muß. Der Inhalt dieses Strings wird daraufhin überprüft, ob er ein gültiges Datum in der oben genannten Form enthält. Ist das Datum gültig, wird es in ⟨SVar⟩ in die Form JJMMTT umgestellt.

⟨NVar⟩ : Statusvariable
0: Auftrag fehlerfrei ausgeführt
1: Ungültige Monatsangabe
2: Ein Trennzeichen ist weder Punkt noch Komma oder die beiden Trennzeichen sind ungleich
3: Tagesangabe nicht erlaubt
4: Jahresangabe nicht erlaubt

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

- Beispiel:

  Überprüfung und Umstellung des in A$ abgestellten Datums.

```
:
10  DIM A$(8)
:
100 CALL 64,A$
```

Inhalt von A$	vorher	nachher
	02.01.78	780201

Fehlermöglichkeit:

Wenn <NVar> nicht codiert ist: ERROR # 38

Bedeutung:

- <SVar> für weniger als 8 Byte dimensioniert.
- Auf einer der Stellen 1,2,4,5,7 oder 8 wurde ein nichtnumerisches Zeichen gefunden.
- Das 3. oder 6. Zeichen ist kein Punkt oder Komma.
- Punkt und Komma gemischt in einem String.
- Monat < 1 oder > 12.
- Tag < 1 oder > 30 bzw. 31. Der 29. Februar ist nur in Schaltjahren zugelassen.

Die CALL-Anweisungen

8.27 CALL 65

Sortieren von Begriffen, die in einem String abgestellt sind, in aufsteigender Ordnung (ASCII-Code).

Syntax:

CALL ⟨NAusdr⟩, ⟨NVar1⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨SVar1⟩, ⟨SVar2⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 65).

⟨NVar1⟩ : Diese numerische Variable dient zur Übergabe des Status vom Unterprogramm an das BASIC-Programm. Der Inhalt von ⟨NVar1⟩ muß beim Aufruf des Unterprogramms 0 sein.

⟨NumPar1⟩ : Anzahl der zu sortierenden Begriffe (⟩= 1).

⟨NumPar2⟩ : Sortierbegrifflänge (⟩= 1).

⟨SVar1⟩ : Der Inhalt dieser String-Variablen wird entsprechend der in ⟨NumPar1⟩ und ⟨NumPar2⟩ vorgegebenen Parameter in aufsteigender Folge sortiert. Nach Beendigung des Sortiervorganges wird kein Grenzzeichen abgestellt! Die Mindestlänge von ⟨SVar1⟩ ergibt sich aus ⟨NumPar1⟩ * ⟨NumPar2⟩.

⟨SVar2⟩ : Diese String-Variable dient als Arbeitsfeld. Die Mindestlänge beträgt ⟨NumPar2⟩ + 8.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

- Beispiel:

  Sortieren von in A$ abgestellten Sortierbegriffen. Die Länge eines Sortierbegriffes ist 5 Zeichen. Die Anzahl der zu sortierenden Begiffe ist 10:

  Inhalt A$ vorher:

  Sortierbegrifflänge (5)

  TOWER1234512355FIELDCYCLESLEEPBRAIN77777TRAINTRUNK...

  10 Sortierbegriffe

```
10 DIM A$(100)          /* DIM. SORTIERSTRING
15 DIM W$(13)           /* DIM. ARBEITSFELD
:
100 LET S=0             /* STATUS AUF 0
110 CALL 65,S,10,5,A$,W$ /* SORTIEREN
120 ON S GOSUB ...      /* STATUS AUSWERTEN
```

  Inhalt A$ nachher:

  1234512355777778RAINCYCLEFIELDSLEEPTOWERTRAINTRUNK...

Statusmeldungen in <NVar1>

<NVar1>	Bedeutung
0	Sortierung wurde ohne Fehler durchgeführt
1	Parameterfehler: - Fehlende(r) Parameter - Falscher Variablentyp - <NVar1> bei Aufruf <> 0 - <NumPar1> und/oder <NumPar2> negativ oder >32767 - Inhalt von <NumPar1> oder <NumPar2> kleiner 1
2	<NumPar1> oder <NumPar2> = 0
3	<SVar1> zu klein dimensioniert
4	<SVar2> zu klein dimensioniert
5	Ein Variablenname wurde mehrfach verwendet.

Die CALL-Anweisungen

8.28 CALL 66

Seit NIROS 5.1/00 steht mit dem CALL 66 eine Funktion zur erweiterten Codewandlung zur Verfügung. Bei der erweiterten Codewandlung kann jeweils 1 Quellbyte in bis zu 13 Zielbytes konvertiert werden.

Die Funktion zur erweiterten Codewandlung wird in BASIC-Programmen wie folgt aufgerufen:

Syntax:

CALL 66, ⟨NVar1⟩, ⟨NVar2⟩, ⟨SVar1⟩, ⟨SVar2⟩, ⟨SVar3⟩, ⟨SVar4⟩, ⟨SVar5⟩, ⟨NVar3⟩

⟨NVar1⟩ : Vor Aufruf der Funktion ist in ⟨NVar1⟩ die Anzahl zu bearbeitender Quellbytes abzustellen. Da der Zielstring um das 13fache expandieren kann, ist es möglich, daß nicht der gesamte Quellstring bearbeitet werden kann. Daher wird nach Ausführung der Funktion die Anzahl der bearbeiteten Quellbytes in ⟨NVar1⟩ abgestellt.

⟨NVar2⟩ : Vor Aufruf der Funktion ist in ⟨NVar2⟩ die dimensionierte (!) Länge des Zielstrings zu übergeben. Nach Durchführung der Funktion enthält ⟨NVar2⟩ die Anzahl der Zielbytes.

⟨SVar1⟩ : Quellstring

⟨SVar2⟩ : Zielstring

⟨SVar3⟩ : Tabellen, die vor Aufruf der Funktion zu füllen sind: Bytes 1-256 enthalten die Zielcodetabelle, Bytes 257-512 enthalten die Zeigertabelle.

⟨SVar4⟩ : Ersatzstringtabelle für die Ersatzstrings 1-36.

⟨SVar5⟩ : Ersatzstringtabelle für die Ersatzstrings 37-72.

⟨NVar3⟩ : Status-Variable
0 = Funktion erfolgreich
1 = Funktion nicht erfolgreich, weil ⟨NVar1⟩ oder ⟨NVar2⟩ unzulässige Werte enthalten.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Bei Formatfehlern wird BASIC-Fehler 38 gemeldet.

Zweckmäßigerweise sollten die Tabellen ⟨SVar3⟩, ⟨SVar4⟩ und ⟨SVar5⟩ aus Dateien gelesen werden.

- Beispiel:

  In diesem Programmbeispiel werden die Beispieltabellen aus der nachfolgend beschriebenen "Arbeitsweise des CALL 66" zugrunde gelegt.

```
:
100 IF  ERR 0 STOP
110 DIM Q$(20),Z$(260),C1$(512),C2$(512),C3$(512)
120 OPEN #3,"XY.CODE"
130 MAT  READ #3;C1$;
140 MAT  READ #3;C2$;
150 MAT  READ #3;C3$;
160 CLOSE #3
170 INPUT 'CR',"QUELLE  : ",Q$
180 IF Q$="" CHAIN ""
190 LET Q= LEN Q$
200 LET Z=260
210 CALL 66,Q,Z,Q$,Z$,C1$,C2$,C3$,S
220 IF S STOP
230 PRINT 'CR';"ZIEL    : ";Z$(1,Z);'CR';
240 GOTO  170
```

Die CALL-Anweisungen

Arbeitsweise des CALL 66

Die Arbeitsweise des CALL 66 wird im folgenden anhand eines Beispiels erläutert.

Aufgabenstellung:

Es soll für Texte in ASCII-Darstellung eine Codewandlung derart durchgeführt werden, daß Kleinbuchstaben in die entsprechenden Großbuchstaben, Umlaute in Doppelbuchstaben (z.B. "Ä" in "AE") und "ß" in "SS" konvertiert werden.

Hexacodes:

```
A ... Z   =  C1 ... DA
a ... z   =  E1 ... FA
Ä, Ö, Ü   =  DB, DC, DD
ä, ö, ü   =  FB, FC, FD
ß         =  EO
```

Analog zur Arbeitsweise des CALL 31 wird zunächst eine Zielcodetabelle benötigt. Die Wertigkeit eines Quellbytes definiert dabei eine Adresse in der Zielcodetabelle, deren Inhalt festlegt, ob das Quellbyte durch ein einzelnes Zielbyte oder durch mehrere Zielbytes zu ersetzen ist.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

- Beispiel Zielcodetabelle:

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
3	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
4	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
5	50	51	52	53	53	55	56	57	58	59	5A	5B	5C	5D	5E	5F
6	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
7	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
8	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
9	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
													****			
D	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	FF	FF*	FF*	DE	DF
			****											****		
E	FF	C1*	C2*	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
			****													
F	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	FA	FF	FF	FF	FE	FF

Enthält die Adresse der Zielcodetabelle einen Wert ungleich 'FF', so wird dieser Wert als Ersetzungszeichen genommen. Beispielsweise wird der Kleinbuchstabe "b"='E2' ersetzt durch 'C2'="B".

Enthält die Adresse der Zielcodetabelle den Wert 'FF' (im obigen Beispiel die Adresse 'DD'), so ist in einer Zeigertabelle an der selben Adresse ein Zeiger auf den Ersatzstring abzustellen.

Die CALL-Anweisungen

- Beispiel Zeigertabelle:

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
													****			
D	00	00	00	00	00	00	00	00	00	00	00	01	02	*03*	00	00
													****			
E	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	00	00	00	00	00	00	00	01	02	03	00	05

Der Buchstabe "Ü"='DD' ist beispielsweise durch den dritten String zu ersetzen, der in einer gesonderten Ersatzstringtabellegeführt wird.

Die Ersatzstringtabelle besteht aus Teilstrings zu jeweils 14 Byte. Das erste Byte eines Teilstrings definiert binär die Länge des Ersatzstrings; in Byte 2 bis 14 wird der maximal 13-stellige Ersatzstring abgestellt.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Bei einer notwendigen Änderung einer beliebigen Codetabelle ist diese zuvor mit MAT READ und MAT WRITE in eine Datei XCTx zu übertragen.

Die Dateien XCTx sind Mikroprozessor-Dateien und bestehen aus einem Satz zu 512 Byte; sie enthalten somit zwei Codetabellen auf den Positionen 1-256 und 257-512. In der Bildschirmmaske werden diese beiden Codetabellen als Eingabe- und Ausgabetabelle bezeichnet.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

8.29 CALL 70

CALL 70 bildet die Schnittstelle zwischen dem BASIC-Interpreter RUN und dem Kanalprogramm $MTX für Magnetbandeinheiten.
Auf dem Magnetband können nur sequentielle Dateien abgestellt werden. Die Verwaltung der Dateien ist vom Anwenderprogramm aus zu steuern.

Es werden 4 Formate unterschieden:

Format	Syntax
1	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar1⟩
2	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar1⟩, ⟨NumPar3⟩, ⟨NVar2⟩
3	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar1⟩, ⟨NumPar3⟩, ⟨NVar2⟩, ⟨SVar1⟩
4	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar1⟩, ⟨SVar2⟩, ⟨SVar3⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 70).

⟨NumPar1⟩ : Nummer der anzusprechenden Bandstation (0 bis 3).

⟨NumPar2⟩ : Nummer der durchzuführenden Funktion (siehe Beschreibung der Funktionen).

⟨NVar1⟩ : Numerische Variable, in der nach Durchführung der Funktion der Status übergeben wird. ⟨NVar1⟩ muß vor Ausführung der Funktion 0 sein.

Die CALL-Anweisungen

8.28-1 CALL 68

Zur Kommunikation zwischen Anwendung und BS. Dieser CALL teilt dem Arbeitsplatz mit, daß eine neue Konfiguration geladen werden soll. Er arbeitet jedoch nur mit neueren Arbeitsplätzen (DAP 4x, DAP 6 ff.).

Syntax:

CALL <NAusdr>, <NVar1>,<NVar2>

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 68).

<NVar1> : Diese Variable enthält den Offset der gewünschten Konfigurationszeile innerhalb des Konfigurationsmoduls. Gültige Werte: 1 - 255

<NVar2> : In dieser Variable wird ein Status an den Aufrufer übergeben.

Mögliche Stati:

0 : Laden/Nachladen erfolgreich abgeschlossen

1 : Konfigurationsdatei bzw. angegebene Konfiguration nicht vorhanden

2 : Timeout
laden/ Konfiguration unvollständig

4 : AP ist nicht nachladbar (RAP, alter DAP, DAP 4)

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Verwalg CALL 70

<NumPar3> : Anzahl der zu verarbeitenden Bandmarken, Blöcke oder Byte (abhängig von der Funktion).

<NVar2> : Aktuelle Anzahl der verarbeiteten Bandmarken, Blöcke oder Byte.

<SVar1> : String-Variable zur Aufnahme des gelesenen oder zu schreibenden Blocks.

<SVar2> : String-Variable zur Aufnahme einer Eingabe-Codetabelle. Diese Tabelle muß vom Anwenderprogramm geladen werden und 256 Byte lang sein.

<SVar3> : String-Variable zur Aufnahme einer Ausgabe-Codetabelle. Diese Tabelle muß vom Anwenderprogramm geladen werden und 256 Byte lang ein.

Das zu wählende Format ist von der durchzuführenden Funktion abhängig.

Funktionen bei Format 1:

Nummer	Funktion
1	Band auf BOT (begin of tape) zurückspulen.
2	Bandgerät "off-line" schalten.
6	Bandmarke schreiben.
7	Gap (Bandlücke) schreiben.
10	Bandmarke schreiben und Band auf BOT spulen.
11	Bandgerät zur Verarbeitung ohne Codeumwandlung eröffnen.
12	Bandgerät zur Verarbeitung ohne Codeumwandlung eröffnen und Band auf BOT zurückspulen.
13	Bandgerät schließen.
14	Bandgerät schließen und "off-line" schalten.

Die CALL-Anweisungen

Funktionen bei Format 2:

Nummer	Funktion
3	Band um die in ⟨NumPar3⟩ vorgegebene Anzahl Blöcke vorsetzen.
4	Band um die in ⟨NumPar3⟩ vorgegebene Anzahl Blöcke zurücksetzen.
8	Band um die in ⟨NumPar3⟩ vorgegebene Anzahl Bandmarken vorsetzen.
9	Band um die in ⟨NumPar3⟩ vorgegebene Anzahl Bandmarken zurücksetzen.

Anmerkung:

Die Anzahl Blöcke/Bandmarken wird in ⟨NumPar3⟩ vorgegeben. ⟨NVar2⟩ enthält nach Ausführung der Operation die tatsächliche Anzahl verarbeiteter Blöcke/Bandmarken. Diese Zahl kann kleiner als die Auftragszahl sein, jedoch wird in diesem Fall in ⟨NVar1⟩ ein Status ≠ 0 übergeben.

Funktionen bei Format 3:

Nummer	Funktion
0	Block lesen. Die Übertragung wird beendet bei Erreichen der dimensionierten Länge von ⟨Svar1⟩ oder der in ⟨NumPar3⟩ vorgegebenen Anzahl Zeichen.
5	Block schreiben. Die Übertragung wird beendet bei Erreichen der dimensionierten Länge von ⟨SVar1⟩ oder durch Erreichen der in ⟨NumPar3⟩ vorgegebenen Anzahl Zeichen.

Anmerkung:

Nachdem die Funktion ausgeführt ist, wird in ⟨NVar2⟩ die tatsächliche Blocklänge beim Lesen bzw. die Anzahl tatsächlich geschriebener Zeichen übergeben. Dieser Wert muß nicht mit dem in ⟨NumPar3⟩ vorgegebenen Wert übereinstimmen.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Funktionen bei Format 4:

Nummer	Funktion
11	Magnetbandstation zur Verarbeitung mit Codewandlung eröffnen. Die Angabe ⟨SVar3⟩ muß nur bei der Ausgabe auf Magnetband angegeben sein.
12	Wie Funktion 11, jedoch mit Zurückspulen des Bandes auf BOT.

Anmerkung:

Die Umwandlungstabellen müssen vom BASIC-Programm in ⟨SVar2⟩ und ⟨SVar3⟩ zur Verfügung gestellt und dürfen nach der Eröffnung des Magnetbandes nicht mehr verändert werden. Folgende Standard-Codetabellen stehen in relativen Dateien im System zur Verfügung:

Dateiname	Tabelle
ASCII7.ASCII8	Umwandlung von ASCII-Code ohne Bit 8 in ASCII-Code mit gesetztem Bit 8
ASCII8.ASCII7	Umwandlung von ASCII-Code mit gesetztem Bit 8 in ASCII-Code ohne gesetztem Bit 8
ASCII.EBCDIC	Umwandlung ASCII-Code mit gesetztem Bit 8 in EBCDIC-Code
EBCDIC.ASCII	Umwandlung EBCDIC-Code in ASCII-Code mit gesetztem Bit 8

Statusübergabe

Nach der Ausführung einer Magnetband-Operation ist unbedingt der Status in ⟨NVar1⟩ zu überprüfen. Folgende Werte können in ⟨NVar1⟩ durch CALL 70 übergeben werden:

⟨NVar1⟩	Bedeutung
0	Operation war erfolgreich
1	Bandgerät nicht verfügbar (bereits eröffnet)
2	Bandgerät nicht bereit (off-line)
3	Band ist schreibgeschützt (kein Schreibring)
4	Schreib-/Lesefehler
5	EOF (Bandmarke gelesen)
6	EOT (Bandende-Spiegel)
7	Unzulässige Operation

Die CALL-Anweisungen

8.30 CALL 72

Übertragen des Inhalts von bis zu 11 numerischen Werten oder Strings in einen String. Die Übertragung erfolgt ohne Rücksicht auf Typ und Format der Quelldaten.

Syntax:

CALL <NAusdr>, <SVar1> [,<NumPar>]^{11} [,<String>]

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 72).

<SVar1> : Diese String-Variable dient zur Aufnahme der (bis zu 11) im folgenden angegebenen Variablen und Konstanten. Ist hier eine numerische Variable angegeben, erfolgt keine Fehlermeldung!

<NumPar> <String> : Maximal 11 beliebige numerische Werte oder Strings, die nach <SVar1> übertragen werden sollen. Die Übertragung wird durch das Erreichen des dimensionierten Endes in der Zielvariablen oder des Endes des letzten Quellwertes beendet. Grenzzeichen in Quellstrings beenden die Übertragung nicht.
Alle übertragenen Werte schließen bündig aneinander an.

- Beispiel:

Übertragung des Inhaltes der Variablen A,B,C,A$,B$,C$ in den String M$.

```
:
10  DIM A$(9),B$(9),C$(9)
15  DIM 2%,A,B,C
20  DIM M$(39)                  /*ZIELSTRING
:
100 CALL 72,M$,A$,B$,C$,A,B,C /*ÜBERTRAGUNG
```

Inhalt der Variablen M$ nach Ausführung:

Byte	1...9	10...18	19...27	28...31	32...35	36...39
	A$	B$	C$	A	B	C

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.31 CALL 73

Übertragen des Inhalts eines Strings in bis zu 11 Variablen. Die Übertragung erfolgt ohne Rücksicht auf Typ und Format der Zielvariablen.

Syntax:

CALL <NAusdr>, <SVar1> { ,<SVar> / ,<NVar> }[11]

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 73).

<SVar1> : Der Inhalt dieser String-Variablen wird in bis zu 11 beliebige Variablen übertragen.

<SVar>
<NVar> : Maximal 11 Variablen, in die der String übertragen werden soll. Die Übertragung wird beendet durch:
- Erreichen des dimensionierten Endes im Quellstring.
- Erreichen des dimensionierten Endes in der letzten Zielvariablen.

• Beispiel:

Der Inhalt des Strings M$ wird in die Variablen A,B,C,A$,B$,C$ übertragen.

```
:
10 DIM A$(10),B$(10),C$(10)    /*ZIELVARIABLE
15 DIM 2%,A,B,C                /*ZIELVARIABLE
20 DIM M$(42)                  /*QUELLVARIABLE
:
100 CALL 73,M$,A,B,C,A$,B$,C$ /*ÜBERTRAGUNG
```

Die CALL-Anweisungen

8.32 CALL 74

Lesen eines Eintrags der Gerätezuordnungstabelle eines BA.

Syntax:

CALL ⟨NAusdr1⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨SVar1⟩, ⟨SVar2⟩, ⟨SVar3⟩

⟨NAusdr1⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 74).

⟨NumPar1⟩ : Angabe der Nr. des BA, aus dessen Gerätezuordnungstabelle gelesen werden soll.

⟨NumPar2⟩ : Angabe der Nr. des Eintrags in der Gerätezuordnungstabelle, der gelesen werden soll. Sie muß zwischen 0 und 12 liegen.
Wird statt einer Eintragsnummer der Wert "-1" eingesetzt und in ⟨SVar1⟩ der Programmname eines Gerätes vorgegeben, liefert die Unterroutine den zugeordneten Systemnamen und die Parameter des Gerätes.

⟨SVar1⟩ : >= 14 Byte
Hier wird der im Eintrag vermerkte Programmname abgestellt. Ist der Eintrag leer, wird ein Leerstring abgestellt.

⟨SVar2⟩ : >= 14 Byte
Hier wird der im Eintrag vermerkte Systemname abgestellt. Ist der Eintrag leer, wird ein Leerstring abgestellt.

⟨SVar3⟩ : >= 10 Byte
Hier werden alle weiteren Parameter abgestellt. Ist der Eintrag leer, wird ein Leerstring abgestellt.

Fehlermöglichkeit: ERROR # 38

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Bedeutung:

- Falscher Variablentyp
- Wert von <NumPar1> zu groß oder < 0
- Wert von <NumPar2> > 12 oder < 0
- String-Var. falsch dimensioniert

- Beispiel:

Die Druckerzuordnung ("$LPT") für den Phantom-Port (BA1) soll festgestellt werden.

```
  10 DIM N$(14),E$(14),P$(10)
   :
1000 FOR I=0 TO 12
1010 CALL 74,1,I,N$,E$,P$
1020 IF N$="$LPT" GOTO 2000
1030 NEXT I
1040 REM KEINE DRUCKERZUORDNUNG
   :
 200 REM E$=ZUGEWIESENER DRUCKER
       P$=DRUCKERPARAMETER
```

GESPREKNOTITIE

ink.	telef.
uitg.	bezoek

datum

naam en adres

tijd

CALL 79

paraaf

gesproken met

telefoonnr.

onderwerp

Ophalen soort systeem

FORMAT:

CALL 79, Ø, C, T$, S

C = VAR. met max. aantal blokken deze dient ingegeven te zijn 2.0.2

S = statusvariabele
als S=2 voldoet de proef.

behandeling

ATLANTA
Bedrijfsformulieren
Bestelnr. A 5705-08

Als C=

9536 → fixed disk 5 Mio

13052 → cartridge 5 Mio

15936 → Fixed disk 8 Mio

16385 → phoenix 8 Mio

25856 → phoenix 13 Mio

31872 → Fixed disk 16 Mio

40960 → storage 21 Mio

64480 → storage 33 Mio

Die CALL-Anweisungen

8.33 CALL 80

CALL 80 bildet die Schnittstelle zwischen dem BASIC-Interpreter RUN und dem Driver $CAS für die Magnetbandcassette.

Auf einer Cassette können nur sequentielle Dateien abgestellt werden. Die Verwaltung der Dateien ist vom Anwenderprogramm aus zu steuern.

Es wird zwischen drei Formaten unterschieden:

Format	Syntax
1	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar1⟩
2	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar1⟩, ⟨NumPar3⟩, ⟨NVar2⟩, ⟨SVar⟩
3	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar1⟩, ⟨SVar1⟩, ⟨SVar2⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 80).

⟨NumPar1⟩ : Nummer der MBC-Einheit (muß 0 sein).

⟨NumPar2⟩ : Funktionsnummer (siehe Funktionsbeschreibung).

⟨NVar1⟩ : Numerische Variable, in der nach Durchführung der Funktion der Status übergeben wird. ⟨NVar1⟩ muß vor der Ausführung der Funktion 0 sein.

⟨NumPar3⟩ : Anzahl der zu lesenden/schreibenden Bytes (abhängig von der Funktion). Der Wert darf die maximale Blocklänge von 512 Bytes nicht überschreiten.

⟨NVar2⟩ : In dieser numerischen Variablen wird, nachdem die Funktion durchgeführt ist, die aktuelle Anzahl der gelesenen oder geschriebenen Bytes Übergeben.

⟨SVar⟩ : String-Variable zur Aufnahme der gelesenen oder geschriebenen Datenblöcke.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

〈SVar1〉 : String-Variable (256 Byte) zur Aufnahme der Eingabe-Codetabelle. Die Codetabelle muß vorher vom Anwenderprogramm geladen werden.

〈SVar2〉 : String-Variable (256 Byte) zur Aufnahme der Ausgabe-Codetabelle. Die Codetabelle muß vorher vom Anwenderprogramm geladen werden.

Das zu wählende Format ist von der durchzuführenden Funktion abhängig.

Funktionen bei Format 1:

Nummer	Funktion
11	Die MBC-Einheit für die Verarbeitung eröffnen und dem Teilnehmer zuweisen, der sie eröffnet hat. Der Deckel der Cassetten-Einheit wird geöffnet, und "$CAS" wartet 105 Sekunden zum Einlegen der Cassette und Schließen des Deckels. Danach wird das Band auf die BOT-Marke (begin of tape) zurückgespult. Wird innerhalb der Wartezeit keine Cassette eingelegt, wird Status 5 gemeldet.
13	Die Verarbeitung abschließen, die Cassette auf den physikalischen Startpunkt zurückspulen und auswerfen. Die MBC-Einheit steht damit anderen Teilnehmern zur Verfügung.
3	Cassetten-Band um einen Block vorsetzen, d.h. das Band wird auf das nächste Gap positioniert. Folgt auf den aktuellen Block kein weiterer, wird nach 25 Sekunden Status 5 gemeldet.
4	Cassetten-Band um einen Block zurücksetzen, d.h. auf das letzte Gap positionieren. Ist kein Block vorhanden (aktueller Block = erster Block), wird nach 25 Sekunden Status 5 gemeldet.
1	Cassetten-Band auf BOT (begin of tape) zurückspulen.
2	Deckel der MBC-Einheit öffnen und Cassette auswerfen.

Die CALL-Anweisungen

Funktionen bei Format 2:

Nummer	Funktion
0	Datenblock lesen und in ⟨SVar⟩ übertragen. Wenn die Cassette mit Format 3 eröffnet ist, wird die entsprechende Codeumwandlung durchgeführt. Die Übertragung wird beendet, sobald die dimensionierte Länge von ⟨SVar⟩ oder die in ⟨NumPar3⟩ vorgegebene Anzahl Byte erreicht ist (⟨NumPar3⟩ darf die maximale Blocklänge von 512 Bytes nicht überschreiten). Nach Durchführung der Funktion wird in ⟨NVar2⟩ die Anzahl der tatsächlich übertragenen Bytes übergeben. Nach der Übertragung ist das Band auf dem Gap hinter dem zuletzt gelesenen Block positioniert. Treten Fehler beim Lesen auf, wird der Vorgang bis zu achtmal wiederholt. Sind die Leseversuche ohne Erfolg, wird Status 4 übergeben.
5	Block schreiben, d.h. der Inhalt von ⟨SVar⟩ wird ab der aktuellen Position auf das Band geschrieben. Ist die Cassette mit Format 3 eröffnet, wird die entsprechende Codeumwandlung durchgeführt. Die Übertragung wird beendet, sobald die dimensionierte Länge von ⟨SVar⟩ oder die in ⟨NumPar3⟩ vorgegebene Anzahl Byte erreicht ist. Nach der Übertragung wird in ⟨NVar2⟩ die Anzahl der tatsächlich geschriebenen Byte übergeben und das Band auf dem Gap hinter dem zuletzt geschriebenen Block positioniert. Bei Schreibfehlern wird der Vorgang bis zu achtmal wiederholt. Bleibt die Funktion ohne Erfolg, wird Status 4 übergeben.

Funktion bei Format 3:

Nummer	Funktion
11	Die MBC-Einheit wird zur Verarbeitung mit Codeumwandlung eröffnet.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Anmerkung:

Nach dem Eröffnen dürfen die Variablen <SVar1> und <SVar2>, in denen die Codetabellen abgestellt wurden, nicht mehr verändert werden.

Die folgenden Standard-Codetabellen stehen im System in relativen Dateien zur Verfügung:

Dateiname	Tabelle
ASCII7.ASCII8	Umwandeln von ASCII-Code ohne Bit 8 in ASCII-Code mit gesetztem Bit 8
ASCII8.ASCII7	Umwandeln von ASCII-CODE mit gesetztem Bit 8 in ASCII-Code ohne Bit 8
ASCII.EBCDIC	Umwandeln von ASCII-CODE mit gesetztem Bit 8 in EBCDIC-Code
EBCDIC.ASCII	Umwandeln von EBCDIC-Code in ASCII-Code mit gesetztem Bit 8.

Statusübergabe

Nach Ausführung einer Cassetten-Operation ist der in <NVar1> übergebene Status unbedingt zu überprüfen.

Status	Bedeutung
0	Operation war erfolgreich
1	Die MBC-Einheit ist nicht verfügbar (von anderem Teilnehmer bereits eröffnet)
2	Die MBC-Einheit ist nicht bereit
3	Die Cassette ist schreibgeschützt
4	Schreib-/Lesefehler
5	Zeit überschritten
6	EOT (Bandendemarke erreicht)
7	Unzulässige Operation

Die CALL-Anweisungen

8.34 CALL 84

Meldung aus Nachrichtendatei lesen, anzeigen und/oder ins Systemlogbuch eintragen.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NVar⟩ [,⟨NumPar2⟩ [,⟨NumPar3⟩]]

,⟨SVar⟩ [,⟨String⟩, ⟨NumPar4⟩, ⟨NumPar5⟩ [,⟨NumPar6⟩]]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 84).

⟨NumPar1⟩ : Funktionsnummer

Funktion	0	1	2	3	4	5	6
Meldung aus Nachrichtendatei lesen	X	X	X	X			
in der Nachrichtenzeile auf Bildschirm anzeigen		X		X			
ins Systemlogbuch abstellen			X	X			
Klartext in Nachrichtenzeile auf dem Bildschirm anzeigen					X		X
Klartext ins Systemlogbuch abstellen						X	X

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

〈NVar〉 : Übergebene Statusmeldungen:
0: Kein Fehler aufgetreten
1: Falsche Parameter übergeben
2: Die Nachrichtendatei, aus der gelesen werden soll, ist nicht vorhanden, bzw. in 〈SVar〉 ist kein gültiger Dateiname vorgegeben, oder die angegebene Datei ist keine Nachrichtendatei.
3: Die LU, auf der die angegebene Nachrichtendatei liegen soll, ist nicht aktiv.
4: Auf dem angegebenen Kanal ist keine Datei eröffnet, oder die eröffnete Datei ist keine Nachrichtendatei.
5: Die angegebene Tabellenbasis und Meldungsnr. zusammen überschreiten die Dateigröße, bzw. die angegebene Meldungsnr. ist größer als die höchste vorhandene Fehlernr. in der Datei MESSAGES.

〈NumPar2〉 : Typ-, bzw. Fachbereichsnr.
(nur bei Fkt. 2,3,5,6)
Falls auch 〈NumPar3〉 angegeben ist: Typnr., mit der die Meldung ins Systemlogbuch abzustellen ist, sonst Fachbereichsnummer.
Fehlen 〈NumPar2〉 und 〈NumPar3〉, wird die Typnr. 4 vergeben, die besagt, daß es sich um eine Meldung der Anwendersoftware handelt, und als Typspezifikation wird die Benutzer-Nr. eingetragen. Liegt der Wert von 〈NumPar2〉 außerhalb des zulässigen Bereichs von 0-255, wird der Wert 4 eingesetzt.
Ist eine Fachbereichsnr. angegeben, wird beim Loggen die Typnr. vom Unterprogramm berechnet. Ist die Typnr. nicht im Typschlüsselverzeichnis für die Auswertung des Systemlogbuchs verzeichnet, so ist auch das Typkürzel nicht vorhanden.

〈NumPar3〉 : Typspezifikation (0-255)
(nur bei Fkt. 2,3,5,6)
Angabe der Typspezifikation, mit der die Meldung ins Systemlogbuch abgestellt werden soll. Fehlt diese Angabe oder liegt sie außerhalb des zulässigen Bereichs von 0-255, wird die Benutzer-Nr. eingetragen.

inkomend / uitgaand

TELEFOON-
BEZOEK- } **NOTITIE**

naam .....

adres .....

woonplaats .....

telefoon nr. .....

gesproken met .....

aangenomen / ontvangen } door .....

datum 9-4-'87

tijd

code

onderwerp of bijzonderheden

CALL 89 / heeft met PHP te maken.

CALL 89, P, 1, P1
- 1 = teken
- P1 = prioriteitscode
- P = poort nrs

CALL 89, P, Ø, P1
- Ø = schrijven

	te behandelen door	datum	paraaf

ATLANTA
Bedrijfsformulieren
Bestelnr. A 5705-01

Die CALL-Anweisungen

⟨SVar⟩ : Meldung oder Klartext.
Diese Variable muß mit mindestens 55 Byte dimensioniert sein. (50 Byte für Meldung + 1 Byte für Blank + 4 Byte für Meldungsnummer) Bei den Funktionen 4-6 ist hier der Klartext einzusetzen. Der Text sollte nicht länger als 242 Byte sein, da nur die ersten 242 Bytes eines Textes ins Systemlogbuch eingetragen werden.

⟨String⟩ : Dateiname (nur bei Fkt. 0,1,2,3)
Angabe der Nachrichtendatei, aus der eine Meldung gelesen werden soll, in der Form LU-Nr./Dateiname. Fehlt die Angabe der LU-Nr., wird 0 angenommen. Bei einer leeren Variablen wird die Datei "MESSAGES" übernommen.

⟨NumPar4⟩ : Tabellenbasis (nur bei Fkt. 0,1,2,3)
Angabe der Tabellenbasis, relativ zu der die Meldungsnr. ausgewertet werden soll (s. auch "Logbuch und Nachrichtendatei").
Ist ⟨String⟩ leer, wird dieser Parameter allerdings nicht ausgewertet, sondern vom Unterprogramm selbst bestimmt.

⟨NumPar5⟩ : Meldungsnummer (nur bei Fkt. 0,1,2,3)
Angabe der Nummer für die Meldung, die gelesen werden soll.

⟨NumPar6⟩ : Kanalnummer (nur bei Fkt. 0,1,2,3)
Soll auf die Nachrichten über einen offenen Kanal zugegriffen werden, um ein günstiges Zeitverhalten des Unterprogramms zu erreichen, so ist die betreffende Nachrichtendatei vom BASIC-Programm zu eröffnen und die Kanalnr. mit ⟨NumPar6⟩ zu übergeben. Ist die Nachrichtendatei nicht eröffnet, darf diese Angabe nicht gemacht werden.

Fehlermöglichkeit: ERROR # 38

Bedeutung:

Falscher Parameter, Syntaxfehler beim Unterprogrammaufruf.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.35 CALL 90

Unterstützung der Bearbeitung von String-Variablen.

Es werden vier Formate unterschieden:

Format	Syntax
1	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨SVar1⟩, ⟨NumPar2⟩ {,⟨NumPar⟩}^{9}
2	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨SVar1⟩, ⟨NumPar2⟩, ⟨SVar2⟩
3	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨SVar1⟩, ⟨NumPar2⟩ [,⟨NumPar3⟩]
4	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨SVar1⟩, ⟨NumPar2⟩, ⟨SVar2⟩ [,⟨NumPar3⟩ [,⟨NumPar4⟩]]

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 90).

⟨NumPar1⟩ : Funktionsnummer (siehe Funktionsbeschreibung).

⟨SVar1⟩ : String-Variable, die den Ziel- oder Quell-String enthält (abhängig von der Funktion).

⟨NumPar2⟩ : Numerischer Wert, der das erste zu bearbeitende Byte in ⟨SVar1⟩ bezeichnet. Der Wert muß ungerade und positiv sein und darf nicht größer als die dimensionierte Länge von ⟨SVar1⟩ sein.

⟨NumPar⟩ : Ein bis neun numerische Variablen, die als Zielvariablen oder als Quellwerte (Variablen oder Konstanten) definiert sind (abhängig von der Funktion).

⟨SVar2⟩ : String-Variable, die als Quell-String dient.

Die CALL-Anweisungen

⟨NumPar3⟩ : Letztes zu bearbeitendes Byte in ⟨SVar1⟩. Der Wert muß ungerade und größer oder gleich dem in ⟨NumPar2⟩ angegebenen Wert sein.

Ist ⟨NumPar3⟩ nicht angegeben, wird der String bis zum dimensionierten Ende bearbeitet, ebenso, wenn ⟨NumPar3⟩ größer als die dimensionierte Länge von ⟨SVar1⟩ ist.

⟨NumPar3⟩ bei Format 4 : Erstes zu übertragendes Byte in ⟨SVar2⟩. Dieser Wert muß positiv sein und darf die dimensionierte Länge von ⟨SVar2⟩ nicht überschreiten. Fehlt diese Angabe, wird ⟨SVar2⟩ vom ersten Byte an übertragen.

⟨NumPar4⟩ : Letztes zu übertragendes Byte in ⟨SVar2⟩. Dieser Wert darf nicht kleiner als ⟨NumPar3⟩ sein. Fehlt diese Angabe, wird bis zum letzten Byte von ⟨SVar2⟩ übertragen.

Funktionsbeschreibungen für die Formate des CALL 90

Funktionen bei Format 1:

Nummer	Funktion
0	Übertragen von ein bis neun numerischen Variablen in eine String-Variable.
1	Übertragen des Inhalts einer String-Variablen in ein bis neun numerische Variablen.

Funktion bei Format 2:

Nummer	Funktion
2	Übertragen des Inhalts von ⟨SVar2⟩ nach ⟨SVar1⟩ ab Adresse ⟨NumPar2⟩. Die Übertragung wird beendet, wenn die dimensionierte Länge des Quell- oder Zielstrings oder ein Grenzzeichen im Quellstring erreicht ist.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Funktion bei Format 3:

Nummer	Funktion
3	Die String-Variable ⟨SVar1⟩ ab Byte ⟨NumPar2⟩ bis Byte ⟨NumPar3⟩ bzw. bis zum dimensionierten Ende wortweise mit Grenzzeichen füllen. Ist ⟨NumPar2⟩ ein gerader Wert, wird der nächst kleinere ungerade Wert angenommen.

Funktionen bei Format 4:

Nummer	Funktion
4	Ein zusammenhängender Teil des Quellstrings ⟨SVar2⟩ wird von Byte ⟨NumPar3⟩ bis Byte ⟨NumPar4⟩ übertragen auf den Zielstring ⟨SVar1⟩ ab Byte ⟨NumPar2⟩. Die Übertragung wird erst nach Erreichen des letzten gewünschten Bytes (⟨NumPar4⟩) oder des dimensionierten Endes des Quell- oder Zielstrings beendet. Grenzzeichen im Teilquellstring werden übertragen und beenden die Übertragung nicht, wenn sie nicht das dimensionierte Ende bezeichnen. Die Übertragung des Teilquellstrings erfolgt byteweise. Das kann zu unerwarteten Ergebnissen führen, wenn Quell- und Zielstring identisch sind.
5	Ein zusammenhängender Teil des Quellstrings ⟨SVar2⟩ wird von Byte ⟨NumPar3⟩ bis ⟨NumPar4⟩ in den Zielstring ⟨SVar1⟩ ab Byte ⟨NumPar2⟩ eingefügt. Das ursprüngliche Byte ⟨NumPar2⟩ und alle folgenden rücken dementsprechend nach rechts.

Die CALL-Anweisungen

- Beispiel:

  Anwendung des CALL 90; Format 4, Funktionsnummer 5:

```
10 DIM A$(3),B$(20)
20 LET A$="ABC"  /* QUELLSTRING
30 LET V=1  /* VON BYTE 1 DES QUELLSTRINGS
40 LET B=3  /* BIS BYTE 3 DES QUELLSTRINGS
50 LET B$="12345678901234"  /* ZIELSTRING
60 LET P=7  /* QUELLSTRING AB BYTE 7 EINFÜGEN
70 PRINT 'CS';"ZIELSTRING VORHER  : ";B$
80 CALL 90,5,B$,P,A$,V,B
90 PRINT "ZIELSTRING NACHHER : ";B$
```

  Ergebnis:

```
ZIELSTRING VORHER  : 12345678901234
ZIELSTRING NACHHER : 123456ABC78901234
```

  Die Zeilen 30, 40 und 60 können auch wegfallen - die Zeile 80 lautet dann:

```
:
80 CALL 90,5,B$,7,A$,1,3
:
```

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- ⟨NumPar2⟩ ist größer als die dimensionierte Länge von ⟨SVar1⟩ oder enthält einen geradzahligen Wert (nicht bei Funktion 3).
- ⟨NumPar3⟩ kleiner als der Wert von ⟨NumPar2⟩.
- ⟨NVar⟩ ist größer als die dimensionierte Länge von ⟨SVar1⟩. ⟨SVar1⟩ ist verändert.
- ⟨NumPar3⟩ ist größer als die dimensionierte Länge von ⟨SVar2⟩ oder enthält einen geradzahligen Wert.
- ⟨NumPar4⟩ kleiner als ⟨NumPar3⟩.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.36 CALL 91

Bit-Funktionen in Strings. In der Syntax werden 2 Formate unterschieden:

Format	Syntax
1	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨SVar⟩, ⟨NumPar2⟩, ⟨NumPar3⟩ [,⟨NumPar4⟩]
2	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨String⟩, ⟨SVar1⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 91).

⟨NumPar1⟩ : Funktionsnummer (Bedeutung siehe Beschreibung der Funktionen).

⟨SVar⟩ : Quell- und Zielstring, in dem Bits invertiert werden (Funktion 0).

⟨NumPar2⟩ : Byte-Adresse, ab der invertiert werden soll (Funktion 0).

⟨NumPar3⟩ : Anzahl Bytes, deren Inhalt invertiert werden soll (Funktion 0).

⟨NumPar4⟩ : Der Inhalt, auf die Anfangsadresse des zu invertierenden Substrings addiert, ergibt die Anfangsadresse des nächsten zu invertierenden Substrings in ⟨SVar⟩. Der Wert muß >= ⟨NumPar3⟩ sein.
Fehlt die Angabe oder ist der Wert = 0, wird 1 eingesetzt.

⟨String⟩ : Erster Quellstring bei den Funktionen 1 bis 6.

⟨SVar1⟩ : Zweiter Quellstring und Zielstring bei den Funktionen 1 bis 6.

Die CALL-Anweisungen

Beschreibung der Funktionen:

Format 1:

Nummer	Funktion
0	Bit-Invertierung von Substrings in einem String. Die Substrings haben gleiche Länge und Anfangsadressen mit gleicher Sprungweite.

- Beispiel:

  Inhalt von A$: "12345678901234567890123456789O"
  Anfangsadresse: 16
  Anzahl zu invertierender Bytes: 3
  Sprungweite der Substrings: 5

  Mit der Anweisung

  ```
  150 CALL 91,0,A$,16,3,5
  ```

  werden in den Bytes 16 - 18, 21 - 23 und 26 - 28 die Bits invertiert.

Format 2:

Nummer	Funktion
1	Bitweise Verknüpfung der Quellstrings mit UND
2	Bitweise Verknüpfung der Quellstrings mit NICHT UND
3	Bitweise Verknüpfung der Quellstrings mit ODER
4	Bitweise Verknüpfung der Quellstrings mit NICHT ODER
5	Bitweise Verknüpfung der Quellstrings mit XOR
6	Bitweise Verknüpfung der Quellstrings mit NICHT XOR

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Erläuterung der Verknüpfungen:

Bit im 1. Quellstring	0	0	1	1
Bit im 2. Quellstring	0	1	0	1
UND	0	0	0	1
NICHT UND	1	1	1	0
ODER	0	1	1	1
NICHT ODER	1	0	0	0
XOR	0	1	1	0
NICHT XOR	1	0	0	1

Fehlermöglichkeit: ERROR # 38

Bedeutung:

Sprungweite < Substringlänge (Funktion 0).

8.37 CALL 93

Übergabe von Informationen über angemeldete Magnetplatteneinheiten an ein BASIC-Programm.

Syntax:

CALL <NAusdr>, <NumPar1>, <NVar2>, <NVar3>, <NVar4>, <NVar5>, <NVar6>

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 93).

Bei den folgenden Variablen gilt die erste Angabe für Multifile-Struktur und die zweite Angabe für Nicht-Multifile-Struktur.

<NumPar1> : = 1%, Angabe der
- Nr. der angemeldeten ILU,
- Nr. der angemeldeten LU,
über die Informationen gewünscht werden.

<NVar2> : = 1%
Diese Variable enthält nach Ausführung:
- die eingestellte Anzahl ILU's pro LU
- den Wert 1

Die CALL-Anweisungen

〈NVar3〉 : 〉= 2%
Diese Variable enthält nach Ausführung:
- die Gesamtzahl adressierbarer Blöcke auf der LU, die durch die angegebene ILU bestimmt ist
- die Anzahl Blöcke der durch 〈NVar1〉/〈Ganzzahl1〉 vorgegebenen LU

〈NVar4〉 : 〉= 2%
Diese Variable enthält nach Ausführung:
- die erste nicht mehr genutzte RDA
- die Anzahl der Blöcke pro LU

〈NVar5〉 : 〉= 2%
Diese Variable enthält nach Ausführung die Anzahl der freien Blöcke.

〈NVar6〉 : = 1%, Statusvariable
= 0: korrekte Bearbeitung
= 1: LU war nicht angemeldet

Fehlermöglichkeit: ERROR # 38

Bedeutung:

Parameter, Typ oder Dimension falsch.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.38 CALL 96

Zulässigkeitsprüfungen für Dateinamen und Informationen über Dateien lesen.

Format	Syntax
1	CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NVar2⟩, ⟨String⟩, ⟨NVar3⟩, ⟨NVar4⟩
2	CALL ⟨NAusdr⟩, ⟨NVar1⟩ ,⟨NVar2⟩, ⟨String⟩, ⟨NVar3⟩, ⟨NVar4⟩, ⟨NVar5⟩, ⟨NVar6⟩, ⟨NVar7⟩, ⟨NVar8⟩, ⟨NVar9⟩, ⟨NVar10⟩, ⟨NVar11⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 96).

⟨NumPar1⟩ : Funktionsnummer: 1-5 mit Format 1
⟨NVar1⟩ 6 mit Format 2
Die Funktionen werden bei Format 1 sequentiell, bei 1 beginnend, bis zur angegebenen Funktion abgearbeitet.
Bei Format 2 wird die LU-Nummer zurückgemeldet.

Funktion 1:

Formale Prüfung des Dateinamens.
Zulässig ist ⟨LU#⟩/ ⟨Dateiname⟩ ⟨T⟩

wobei

⟨T⟩ ::= ; | , | = | ! | & | $ | > | < | -| GZ (←000←) |
Blank (←240←) | 'CR' (←215←) | Null (←200←)

Führende Blanks sind erlaubt und werden überlesen.

Funktion 2:

Prüfung, ob die LU vorhanden ist.
Nachdem Funktion 1 durchlaufen wurde, wird geprüft, ob die angegebene LU installiert ist.

Die CALL-Anweisungen

Funktion 3:

Nachdem die Funktionen 1 und 2 durchlaufen wurden, wird geprüft, ob die Datei im INDEX-File eingetragen ist. Bei negativem Ergebnis wird Status 13 gemeldet.

Funktion 4:

nicht belegt

Funktion 5:

Nachdem die Funktionen 1-3 durchlaufen wurden, wird geprüft, ob die Datei zur Zeit von einem anderen Arbeitsplatz eröffnet worden ist. Dabei gibt es 3 verschiedene Zustände:

- Datei nicht eröffnet
- Datei bereits eröffnet, ohne Sperre: Status 15
- Datei bereits eröffnet, mit Sperre : Status 21

Funktion 6:

Dateiinformationen lesen

⟨NVar2⟩ : Statusvariable

⟨String⟩ : Dateiname - muß mindestens mit 15 Bytes dimensioniert bzw. 15 Bytes lang sein

⟨NVar3⟩ : enthält nach Funktionsausführung:
- das Displacement, das auf das Ende oder das erste fehlerhafte Zeichen im String verweist bei Fkt. 1-5
- die Anwendernummer bei Fkt. 6

⟨NVar4⟩ : bei Fkt. 1-5 :
Angabe der LU-Nr.; diese wird aber nur berücksichtigt, wenn im String keine LU-Nr. ermittelt wird. Bei Fkt. 1 und 2 ist die Angabe -1 erlaubt.
Nach Rücksprung aus dem CALL-Unterprogramm enthält diese Variable die LU-Nr.
Bei Fkt. 6 :
Nach Rücksprung enthält diese Variable einige Dateiparameter (siehe CALL 97) .

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

<NVar5> : enthält nach Ausführung die Dateigröße in Blöcken.

<NVar6> : enthält nach Ausführung den Dateistatus. Ein Wert <> 0 bezeichnet eine formatierte Datei.

<NVar7>
<NVar8>
<NVar10> : Dummy-Variablen, die nur wegen Kompatibilität zu früheren Releases syntaktisch notwendig sind.

<NVar9> : enthält nach Ausführung den Zeitpunkt der Dateierstellung bzgl. 1.1.1980.

<NVar11> : enthält nach Ausführung die Plattenadresse des Dateikennsatzes (dezimal).

Statusmeldungen

Fkt.	Status	Fkt. fehlerfrei	Fehler	Bedeutung
alle	0	X		Funktion fehlerfrei
1	1		X	LU# unzulässig
	2		X	<T> nach LU# fehlt oder unzulässig
	11		X	unzulässiges Zeichen im Dateinamen, kein Dateiname vorgegeben oder Dateiname > 14 Zeichen
2	12		X	LU nicht aktiv
3	13		X	Datei nicht vorhanden
5	15	X		Datei bereits eröffnet ohne Sperre
	21	X		Datei bereits eröffnet mit Sperre
alle	63		X	undefinierter Fehler

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Variablen bzw. Informationen fehlen, falsche Dimensionierung der Variablen bzw. falscher Inhalt.
- Unzulässige LU# in <NVar4> angegeben.
- Nicht vorhandene Funktion in <NumPar1>/<NVar1> angegeben.
- <String> bei Funktion 6 kleiner als 15 Bytes.

Die CALL-Anweisungen

8.39 CALL 97

Lesen von Datei-Informationen aus den Dateikennsätzen über die Datei INDEX. Es können von allen Dateien der angesprochenen Magnetplatten folgende Informationen gelesen werden:

- Dateiname
- Konto
- Anwendernummer
- Dateiart
- Dateigröße in Blöcken (Sektoren)
- Dateistatus
- Zeitpunkt der Dateierstellung (in Stunden, bezogen auf den 1.1.1980).
- Plattenadresse des Dateikennsatzes

Der Zugriff auf die Datei INDEX erfolgt über die relative Satznummer innerhalb der Datei beginnend mit 0.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NVar2⟩, ⟨String⟩, ⟨NVar3⟩, ⟨NVar4⟩, ⟨NVar5⟩, ⟨NVar6⟩, ⟨NVar7⟩, ⟨NVar8⟩, ⟨NVar9⟩, ⟨NVar10⟩, ⟨NVar11⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 97).

⟨NumPar1⟩ : Logische Einheitennummer der Platte, von der die Dateiinformationen gelesen werden sollen.

⟨NVar2⟩ : Relative Satznummer des zu lesenden Satzes in der Datei INDEX. Ist dieser Satz ein Leersatz, wird automatisch die Nummer des nächsten belegten Satzes eingesetzt.
Nach erfolgreicher Durchführung wird der Inalt von ⟨NVar2⟩ um 1 erhöht. Wurde dagegen das Dateiende erreicht, wird -1 übergeben.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

Alle folgenden Variablen werden durch CALL 97 aufbereitet und beinhalten nach der Durchführung des CALL die Dateiinformationen.

⟨String⟩ : Länge ≥ 15 Byte, Übergabe des Dateinamens.

⟨NVar3⟩ : ≥ 2%. In dieser Variablen wird die Angabe

- Benutzernummer

an das Anwenderprogramm übergeben.

⟨NVar4⟩ : In dieser numerischen Variablen (> 2%) sind einige Dateiparameter verschlüsselt. So läßt sich mit der Formel

FRA (⟨NVar4⟩ / 32) * 32

und der folgenden Tabelle der Dateityp bestimmen.

Dateitypen:
0: Permanente Systemdatei
1: Systemdriver/Processor
2: BASIC-Processor/-Programm
3: Stand-alone-Processor/-Prog.
5: Mikroprocessor-Datei
8: COBOL
9: Optimiertes BASIC-Programm (D-File)
10: PASCAL
24: Textdatei
25: Formatierte Datei
26: Relative/Indexdatei
30: Peripheriedriver

⟨NVar5⟩ : Dateigröße in Blöcken (Sektoren).

⟨NVar6⟩ : Dateistatus: ein Wert ≠ 0 bezeichnet eine formatierte Datei.

⟨NVar7⟩ : Dummy-Variablen, die nur noch wegen Kompatibilität zu früheren Releases syntaktisch notwendig sind. Der Wert dieser Variablen ist bedeutungslos.
⟨NVar8⟩
⟨NVar10⟩

⟨NVar9⟩ : Zeitpunkt der Dateierstellung (bezogen auf den 1.1.1980).

⟨NVar11⟩ : Plattenadresse des Dateikennsatzes (dezimal).

Die CALL-Anweisungen

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- <String> ist Variable und wurde für weniger als 15 Byte dimensioniert.
- Angesprochene LU ist nicht aktiv.
- Eine numerische Variable wurde kleiner als 2% dimensioniert.
- Es fehlen Variable.
- <NumPar1> oder <NVar2> negativ.

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.40 CALL 98

Senden eines beliebigen Systemkommandos an einen beliebigen Platz. Dadurch wird die Möglichkeit geboten, jeden Arbeitsplatz ab- und anzumelden und dort ein beliebiges Programm zu starten.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar⟩, ⟨String⟩, ⟨NVar⟩

⟨NAusdr⟩ : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 98).

⟨NumPar⟩ : Arbeitsplatznummer des Teilnehmers, an den das Kommando gesendet werden soll.

⟨String⟩ : Systemkommando, das am Platz ⟨NVar1⟩/⟨Dezzahl1⟩ gestartet werden soll. Zugelassen sind alle Systemkommandos, die im Systemkommandomodus über die Tastatur eingegeben werden können.

⟨NVar⟩ : In dieser Variablen wird nach der Ausführung der Status übergeben:

⟨NVar⟩	Bedeutung
0	Funktion fehlerfrei durchgeführt.
⟨⟩0	Funktion konnte nicht durchgeführt werden (das Kommando wurde nicht an den adressierten Platz übergeben).

An-/Abmelden einer Task:

Zum Starten eines Programmes per CALL-Anweisung ist es erforderlich, die Anweisung CALL 98 zweimal auszuführen. Bei der ersten Ausführung muß an die adressierte Task der Code "0" gesendet werden. Damit wird der adressierte Plat definiert abgemeldet.
Im Anschluß daran kann durch Übersendung eines Systemkommandos ein beliebiges Programm gestartet werden.

Die CALL-Anweisungen

Anmerkung:

Nach Absetzen einer CALL 98-Anweisung muß 2 Sekunden gewartet werden (SIGNAL 3,20), bevor der Status in ⟨NVar⟩ abgefragt wird.

- Beispiel:

  Starten des Programms FAKTURA am Phantom-Port (Port-Nummer 1):

```
:
100 LET S=0                 /* ⟨NVar⟩ AUF 0
110 LET A$= "Ö"             /* CODE FÜR ABMELDEN
115 CALL 98,1,A$,S          /* PHANTOM-TASK ABMELDEN
120 SIGNAL 3,20             /* 2 SEKUNDEN WARTEN
125 IF S GOTO ...           /* STATUSABFRAGE
130 LET A$="FAKTURA"        /* PROGRAMMNAMEN LADEN
135 CALL 98,1,A$,S          /* PROGRAMM STARTEN
140 SIGNAL 3,20             /* 2 SEKUNDEN WARTEN
145 IF S GOTO ...           /* STATUSABFRAGE
```

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."

Die CALL-Anweisungen

8.41 CALL 99

Übernehmen oder Ändern von Systemdatum und Systemzeit.

Syntax:

CALL <NAusdr>, <SVar>

<NAusdr> : Der Wert dieses numerischen Ausdrucks ist die Nummer des CALL-Unterprogramms (in diesem Fall 99).

<SVar> : String-Variable, Länge >= 23 Byte. <SVar> dient zur Übergabe von Datum und Zeit vom System an das Anwenderprogramm bzw. vom Anwenderprogramm an das System.

Übernehmen von Datum/Zeit:

Soll das aktuelle Systemdatum und Systemzeit an das Anwenderprogramm übergeben werden, muß <SVar> ein Leerstring sein.

- Beispiel:

  Systemdatum und Systemzeit übernehmen.

```
:
100 LET A$= ""        /* <SVar> = LEERSTRING
105 CALL 99,A$        /* DATUM/ZEIT ÜBERNEHMEN
```

Datum und Zeit stehen anschließend in folgender Form in A$:

```
|M M M  T T,  J J J J  h h:m m:s s|
```

MMM : Monat: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC

TT : Tag

JJJJ : Jahr

hh : Stunde

mm : Minute

ss : Sekunde

Die CALL-Anweisungen

Systemdatum und Systemzeit ändern:

⟨SVar⟩ muß wie folgt belegt sein:

J	J	J	J	,	M	M	,	T	T	,	h	h	,	m	m	,	s	s

JJJJ : Jahr

MM : Monat

TT : Tag

hh : Stunde

mm : Minute

ss : Sekunde

Als Trennzeichen kann jedes beliebige, nichtnumerische Zeichen verwendet werden.

- Beispiel:

  Ändern des aktuellen Systemdatums. Das neue Datum ist in der geforderten Form einzugeben.

```
:
100 INPUT "DATUM/ZEIT: ",A$ /*DATUM/ZEIT EINGEBEN
105 CALL 99,A$              /*DATUM/ZEIT ÄNDERN
```

Fehlermöglichkeit: ERROR # 38

Bedeutung:

- Datum/Zeit in falschem Format vorgegeben.
- Datum oder Zeit unmöglich.

8

© „Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmustereintragung vorbehalten."