NIXDORF
COMPUTER

Systemliteratur

Nixdorf 8870/1

System-Software

Business-Basic

Entwicklungsbereich Software Nr. 34001.00.4.93-01

Ihre Aufnahme in die Verteilerliste für den Änderungsdienst erfolgt nur, wenn Sie diese Karte einschicken.

Bitte senden Sie Änderungen und Ergänzungen zu diesem Literaturteil an die folgende Anschrift.

Betreuende Nixdorf-Niederlassung (unbedingt angeben):

Name: _____

in Firma: _____

oder Bereich NCAG: _____

Verkehrsnummer:
34001.00.4.93-01

Ausgabedatum der letzten Änderung
lt. Organisationsblatt (unbedingt angeben):

_____

Postkarte

Nixdorf Computer AG
Abt. ZSI
Fürstenallee 7

D-4790 Paderborn
West-Germany

Bitte ausschneiden und in die Tasche im Handbuchrücken einstecken.

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	01.01.1975	
Neuauflage	01.06.1978	Rel.: 3.2
Änderung 1	15.01.1979	Rel.: 3.3

Auszutauschen sind die Seiten:

0-1 bis 0-13, 2-7/2-8, 3-11/3-12, 6-7/6-8,
7-7 bis 7-9/, 7-43 bis 7-46, 7-69 bis 7-77,
9-91/7-92, 7-95 bis 7-100, 9-11/9-12,
10-91 bis 10-98, 11-1 bis 11-4/, 12-5 bis 12-10,
12-13 bis 12-16, 13-1 bis 13-4, 13-7 bis 13-10

Einzufügen sind die Seiten:

7-9/1, 9-15 bis 9-18, 11-4/1 bis 11-4/5

Es entfallen die Seiten:

7-78 bis 7-82, 7-93/7-94

Übersicht über die Systemliteratur 8870/1

Bedienerhandbuch

Release:	Ausgabedatum:
3.1	E: 01.12.1977
3.2	N: 01.06.1978
3.3	N: 01.01.1979

Hardware-Übersicht, TAMOS-Grundlagen, Grundlagen der Systembedienung, Bedienung der NIROS-Systemkommandos.

Business-Basic

Release:	Ausgabedatum:
	E: 01.01.1975
3.2	N: 01.06.1978
3.3	Ä: 15.01.1979

Beschreibung der Anweisungen, Logik und Programmierung in Basic.

Fehlermeldungen

Release:	Ausgabedatum:
3.2	E: 01.08.1978
3.3	N: 15.01.1979

Übersicht aller Fehlermeldungen des Systems mit Fehlerursachen und möglichen Maßnahmen zur Fehlerbehebung.

Betriebssystem I

Release:	Ausgabedatum:
3.2	01.09.1978 (Vorabvers.)
3.3	E: 15.01.1979

Detaillierte Beschreibung der einzelnen Betriebssystem-Komponenten.

TAMOS

Release:	Ausgabedatum:
3.3	E:01.01.1979

Ausführliche Beschreibung des TAMOS-Konzeptes, Bedienung des Manager-Selektors.

E: Erstauflage N: Neuauflage Ä: Anderung

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. VP03-ZSI
Fürstenallee 7

4790 Paderborn

NIXDORF
COMPUTER

Einführung

Das Programmiersystem im Betriebssystem NIROS

Syntax

Datendarstellung/Datenformate

Programmstruktur

Ein-/Ausgabekonzept

Ein-/Ausgabeprogrammierung

Inter-Task-Kommunikation

Funktionen in BASIC

Beschreibung der Anweisungen

CALL Unterprogramme

Schnittstellen

Fehlermeldungen/Tabellen

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Einführung

1 Einführung

Die Programmiersprache BASIC (Beginners All-Purpose Symbolic Instruction Code) wurde zu Beginn der sechziger Jahre mit dem Ziel entwickelt, eine leistungsstarke und trotzdem einfach zu erlernende Programmiersprache für den Einsatz auf Time-sharing Systemen zu schaffen.

Ihre Merkmale sind:

Eine einfache Grammatik, die auf wenigen Anweisungen in englischer Sprache basiert.

Funktionen, die eine Bearbeitung von Strings (Zeichenketten), Matritzen und arithmetischen Ausdrücken ermöglichen.

Aufbereitungsfunktionen, die Programmtests und Programmänderungen unterstützen.

Einsatz eines Interpreters (statt Compiler), der die Möglichkeit bietet, Programme interaktiv zu erstellen und zu testen.

Ein Programmteil kann unmittelbar, nachdem er eingegeben wurde, ausgeführt, modifiziert, erweitert oder auch gelöscht werden.
Die bei anderen Programmiersprachen notwendige Compilation entfällt also durch den Interpreter.

Einführung

Die in diesem Handbuch beschriebene Version "Business-Basic" enthält alle Charakteristiken, die Basic praktisch zur universellen Dialogsprache gemacht haben.

Neben guten Eigenschaften für den Einsatz in Schulungen oder zur technisch-wissenschaftlichen Problemlösung ist "Business-Basic" um Funktionen erweitert worden, die sich speziell zur Realisierung kommerzieller Applikation eignen.

Diese Erweiterungen sind:

- Bearbeitung von Dateien.
- Dezimalarithmetik mit erweiterter Genauigkeit.
- Druckaufbereitung über Masken.
- Programmverkettung (Segmentierung).
- Maskierung von Programmfehlern.
- Inter-Task Kommunikation.

Im Gegensatz zu anderen Programmiersprachen wie z. B. COBOL unterliegt BASIC keiner festgelegten Norm, kann also von Hersteller zu Hersteller erhebliche Unterschiede aufweisen, die nicht nur in systemspezifischen Ursachen begründet sein müssen.

Das Programmiersystem

2. Das Programmiersystem Business-BASIC im Betriebssystem NIROS

Das "Business-Basic"-Programmiersystem unter NIROS 3.2 setzt sich aus den folgenden Processoren zusammen:

- BASIC
- RUN
- RUNMAT
- SAVE
- KILL und
- COPY

Im Gegensatz zu den meisten Programmiersprachen, die Übersetzungsprogramme voraussetzen, werden Business-Basic-Anweisungen vom RUN-Processor zur Laufzeit des Programmes interpretiert.

Die sich daraus ergebenden Vorteile sind:

- Kein Zeitverlust durch Übersetzungsläufe.
- Programmierung im Dialogverkehr.
- Wegfall umfangreicher Programmlisten, die vom jeweiligen Übersetzungsprogramm erstellt werden.
- Leichtes Austesten und Ändern, weil nur eine Programmversion (das Quellprogramm) zu warten ist.

Das Programmiersystem

2.1 Der Basic-Processor

Der System-Processor "BASIC" ist anzuwählen um:

- neue "Business-Basic" Programme einzugeben.
- bestehende Programme zu laden und zu pflegen, (d.h. Programmänderungen durchzuführen).
- Testprogrammläufe zu veranlassen.

Darüberhinaus bietet dieser Processor die Möglichkeit, eine Anweisung sofort nach ihrer Eingabe auszuführen, was als "Direktausführungs-Modus" bezeichnet wird.

Nachdem der Basic-Processor durch ein Systemkommando angewählt wurde, befindet er sich im Basic-Kommandomodus, d.h. am Bildschirmarbeitsplatz können die Funktionen des Processors im Dialog per Kommando angewählt werden.

Der Basic-Kommandomodus ist am Bildschirm dadurch ausgewiesen, daß der Cursor am Anfang einer Leerzeile positioniert ist.

Das Programmiersystem

2.1.1 Kommandos im Basic-Processor

Zur Bearbeitung von BASIC-Programmen stehen dem Anwender die folgenden Funktionen zur Verfügung:

- DEBUG
- DELETE
- DUMP
- HELP
- LIST
- LOAD
- NEW
- RENUMBER
- RUN
- SIZE

Zur syntaktischen Beschreibung dieser Funktionen wird im folgenden die gleiche Metasprache verwendet wie zur Beschreibung der "Business-Basic"-Sprache (siehe Pkt. 3.1.1).

Das Programmiersystem

2.1.1.1 Eingabe von Anweisungen

Bei ihrer Eingabe werden die Anweisungen gemäß ihren Zeilennummern in das im Hauptspeicher (Partition) stehende Programm eingefügt.
Existiert bereits eine Anweisung mit der gleichen Zeilennummer, wird diese durch die neue überschrieben.

Aufbau:

<ZL-NR> <ANWEISUNG> CR

Funktion:

<ZL-NR> = Zeilennummer. Eine positive Ganzzahl im Bereich von 1 bis 9999.

<Anweisung> = Eine gültige Basic-Anweisung, wie sie im Kapitel 3 (Syntax) beschrieben sind.

CR = Eingaben werden durch Drücken der Taste "CR" abgeschlossen.

Bei der Eingabe der Anweisungen sind die hier beschriebenen Bedingungen der Syntax unbedingt einzuhalten.

Eine Reihe von Prüfungen, außer der syntaktischen, können erst bei der Ausführung des Programmes (von "RUN") durchgeführt werden, weil z. B. einige Anweisungen dynamisches Verhalten zeigen und im Verlauf des Programmes modifiziert werden.

2.1.1.2 "DEBUG"-Kommando

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

Aufbau:

[<ZL-NR.>] DEBUG

Funktion:

<ZL-NR.> = Zeilennummer der Anweisung, womit der Testlauf begonnen werden soll. Ist <ZL-NR.> nicht angegeben, startet DEBUG mit der ersten Anweisung des Programmes.

<ZL-NR.> ist nur dann sinnvoll, wenn bereits ein initialisierendes "RUN"-Kommando durchgeführt wurde (siehe auch Pkt. 2.1.1.10).

DEBUG = Kommandowort.

DEBUG initialisiert den Einzelschritt-Test (DEBUG-Modus), führt die erste bzw. die unter <ZL-NR> angegebene Anweisung aus und meldet vor der Ausführung der nächsten Anweisung:

STOP AT <ZL.-NR.>

Der Basic-Processor befindet sich danach wieder im Kommando-Modus, und der Bediener kann durch Drücken der Taste "CR" die Ausführung der nächsten Anweisung auslösen.

Jedes andere Kommando ungleich DEBUG (z. B. die Eingabe einer Basic-Anweisung, oder sobald mit DEBUG eine CHAIN- oder LINK-Anweisung ausgeführt wurde) löscht den DEBUG-Modus.

Beispiel: 300 DEBUG

Beginnen des DEBUG-Modus bei der Zeilennummer 300 und Ausführen dieser Anweisung.

ERROR # 6 "NO SUCH LINE NUMBER"
wird gemeldet, wenn die im DEBUG-Kommando angegebene Zeilennummer nicht vorhanden ist.

Das Programmiersystem

2.1.1.3 "DELETE"-Kommando

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

Aufbau:

[<ZL-NR.1>] DELETE [[<ZL-NR.2>]]

Funktion:

<ZL-NR.1> = Zeilennummer der Anweisung, die als erste gelöscht werden soll.
Ist nur <ZL-NR.1> ohne "DELETE" angegeben, wird lediglich diese eine Anweisung gelöscht.
Fehlt diese Angabe, wird ab der ersten Anweisung im Programm gelöscht.

DELETE = Kommandowort.

<ZL-NR.2> = Zeilennummer der letzten zu löschenden Anweisung.
Fehlt die Angabe <ZL-NR.2>, wird bis zur letzten Anweisung (inclusive) im Programm gelöscht.

Ist nur "DELETE" angegeben, wird das gesamte Programm das sich z.Zt. im Hauptspeicher (Partition) befindet, gelöscht.

Es erzeugt in dieser Form den gleichen Zustand wie das Kommando "NEW".

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.

Beispiel: 100 DELETE 200

Löschen der Anweisungen von Zeilennummer 100 bis 200 (inclusive).

Das Programmiersystem

2.1.1.4 "DUMP"-Kommando

Ausgabe des im Hauptspeicher (Partition) stehenden BASIC-Programmes in eine beliebige Textdatei auf der Platte bzw. auf einen beliebigen Drucker.

Aufbau:

[<ZL-NR.>] DUMP [<LU>/] <DATEINAME> [!][<ZL-NR.>][↑<ZEILE/SEITE>]

Funktion:

<ZL-NR.> = Zeilennummer der ersten Anweisung die ausgegeben werden soll. Fehlt diese Angabe, beginnt die Ausgabe mit der ersten Anweisung des Programmes.

DUMP = Kommandowort.

<LU>/ = Die Nummer der logischen Platten-Einheit, auf der die Textdatei liegt. Ist LU = 0, kann diese Angabe entfallen.

<DATEINAME> = Der Name der Textdatei, in die ausgegeben werden soll, oder die Bezeichnung des Drucker-Drivers, auf dem das Programm ausgedruckt werden soll.

! = Die Angabe "!" ist dann erforderlich, wenn die Ausgabe in eine Textdatei erfolgt, die bereits vorhanden ist und ersetzt werden soll.

<ZL-NR.> = Die Nummer der letzten auszugebenden Anweisung.

↑<ZEILEN/SEITE> = Wird "↑" und die Anzahl Zeilen je Seite angegeben, erfolgt der "DUMP" mit Seitentrennung.

Fehlt diese Angabe, wird der "DUMP" ohne Seitentrennung durchgeführt. Dies gilt sowohl für die Ausgabe auf einem Drucker als auch für die Ausgabe in einer 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.

Das Programmiersystem

Beispiel: 500 DUMP $LPT 148

Ausgabe des Programmes ab der Anweisung mit der Zeilennummer 500 auf dem Nadeldrucker mit Seitentrennung nach 45 Zeilen.

Folgende Fehler sind bei der Ausführung des DUMP-Kommandos möglich:

ILLEGAL FILENAME
Der angegebene Dateiname ist nicht zulässig, weil z.B. unerlaubte Zeichen verwendet wurden.

FILENAME IN USE AND NO "!" SUPPLIED
Der angegebene Dateiname wird bereits benutzt.

FILENAME IN USE, OLD FILE BEING BUILT OR REPLACED
Eine Datei mit dem angegebenen Namen wird z. Zt. angelegt oder ersetzt.

FILENAME IN USE BY A DIFFERENT ACCOUNT
Der angegebene Dateiname ist bereits in einem anderen Teilnehmerkonto vergeben worden.

LOGICAL UNIT NOT ACTIVE
Die mit <LU/> bezeichnete logische Einheit ist nicht aktiv (nicht installiert), oder der Drucker kann nicht eröffnet werden (z.B. nicht eingeschaltet).

LOGICAL UNIT DOES NOT HAVE ENOUGH FREE BLOCKS
Auf der angegebenen logischen Einheit sind nicht genügend Blöcke vorhanden, um die Datei aufzunehmen.

ACCOUNT DOES NOT HAVE ENOUGH FREE BLOCKS
Das Konto des aufrufenden Benutzers verfügt nicht über genügend Blöcke, um das Programm aufzunehmen.

ERROR # 12
Das Programm ist list-/ kopiergeschützt.

2.1.1.5 "HELP"-Kommando

Ausgabe des Fehlertextes der angegebenen Fehlernummer (ERROR #) bzw. der zuletzt aufgetretenen Fehlernummer.

Aufbau:

HELP [<FEHLERNUMMER>]

Funktion:

HELP = Kommandowort.

<FEHLERNUMMER> = Ist eine bestimmte Fehlernummer angegeben, erscheint der Text, der dieser Nummer entspricht.
Wird keine Fehlernummer angegeben, erscheint der Fehlertext der zuletzt aufgetretenen ERROR-Nummer.

Beispiel: Eingabe: HELP 1 CR

Ausgabe: SYNTAX ERROR

Ist die eingegebene Nummer nicht belegt, erfolgt die Meldung:

NO SUCH MESSAGE NUMBER.

Für Fehlernummern größer 136 erfolgt keine Textausgabe.

2.1.1.6 "LIST"-Kommando

Ausgabe des z. Zt. im Hauptspeicher stehenden Basic-Programmes am Bildschirm.

Aufbau:

[<ZL-NR.1>] LIST [<ZL-NR.2>]

Funktion:

<ZL-NR.1> = Zeilennummer der ersten auszugebenden Anweisung. Fehlt diese Angabe, beginnt die Ausgabe mit der ersten Anweisung des Programmes.

LIST = Kommandowort.

<ZL-NR.2> = Zeilennummer der letzten auszugebenden Anweisung. Fehlt diese Angabe, wird bis zur letzten Anweisung ausgegeben.

Fehlen die Angaben <ZL-NR.1> und <ZL-NR.2> wird das gesamte Programm ausgegeben.
Durch Drücken der Taste "ESC" kann die Ausgabe beendet werden.

Beispiel: 100 LIST

Ausgabe des Programmes ab der Anweisung mit der Zeilennummer 100 bis zum Programmende bzw. bis zum Abbruch der Ausgabe durch Drücken der Taste "ESC".

2.1.1.7 "LOAD"- Kommando

Laden eines Basic-Programmes aus einer Textdatei in den Hauptspeicher.

Aufbau:

LOAD [<LU>/]<DATEINAME>

Funktion:

LOAD = Kommandowort.

<LU>/ = Die Nummer der logischen Platteneinheit, auf der die Datei liegt. Ist LU = 0, kann die Angabe entfallen.

<DATEINAME> = Name der Textdatei, aus der das Programm geladen werden soll.

Befindet sich zum Zeitpunkt der Ausführung ein anderes Programm (oder Programmteil) im Hauptspeicher, werden beide Programme gemäß ihren Zeilennummern zusammengemischt. Treten in beiden Programmteilen identische Zeilennummern auf, werden die betreffenden Nummern und die zugehörigen Anweisungen des im Speicher befindlichen Programmes durch die der Textdatei ersetzt.

Diese Möglichkeit des Mischens von im Hauptspeicher befindlichen Programmteilen mit in Textdateien befindlichen Programmteilen unterstützt das Einbinden von Standardroutinen in Applikationsprogramme.

Fehlermöglichkeiten:

ILLEGAL FILENAME
Der angegebene Dateiname ist unerlaubt, weil z.B unzulässige Zeichen verwendet wurden.

INCORRECT FILE TYPE
Die angegebene Datei ist keine Textdatei.

FILE NOT FOUND
Auf der angesprochenen logischen Einheit ist keine Datei mit dem angegebenen Namen vorhanden.

LOGICAL UNIT NOT ACTIVE
Die mit <LU>/ spezifizierte logische Einheit ist nicht aktiv (nicht installiert).

FILE IS READ PROTECTED
Die Datei ist lesegeschützt.

Bei der Ausführung des LOAD-Kommandos werden die gleichen Prüfungen auf Syntax und Format durchgeführt wie bei der Eingabe von Anweisungen über die Tastatur.

Werden dabei Fehler festgestellt, erfolgt am Bildschirm eine Anzeige der entsprechenden Fehlernummer (ERROR #). Die Routine arbeitet trotzdem weiter, lädt die falschen Anweisungen allerdings nicht in den Hauptspeicher.

2.1.1.8 "NEW"- Kommando

Dieses Kommando löscht und initialisiert den Hauptspeicher (die Partition) des aufrufenden Benutzers.

Aufbau:

NEW = Kommandowort.

Hinweis:

Es werden immer nur die ersten drei Zeichen einer Eingabe geprüft, das heißt also, daß jede beliebige Eingabe, deren erste Zeichen "NEW" sind, den Hauptspeicher löscht.

2.1.1.9 "RENUMBER" Kommando

Neunumerierung der Zeilennummern eines Basic-Programmes.

"RENUMBER" wird im allgemeinen nach umfangreichen Änderungen in Basic-Programmen durchgeführt, hauptsächlich dann, wenn durch Einfügen neuer Anweisungen keine Lücken zwischen den Zeilennummern vorhanden sind und weitere Einfügungen dadurch unmöglich werden.

Aufbau:

[<ZL-NR.>] RENUMBER [<STEP>]

Funktion:

<ZL-NR.> = Die Zeilennummer, welcher die erste Anweisung im Programm zugeteilt werden soll.
Ist <ZL-NR.> nicht angegeben, erhält die erste Anweisung die Zeilennummer <STEP>.

RENUMBER = Kommandowort.

<STEP> = Gibt die Schrittweite (Differenz) zwischen zwei Zeilennummern an.
Ist <STEP> nicht angegeben, wird als Standard-Schrittweite "10" eingesetzt.

Die maximale Schrittweite läßt sich durch folgende Formel herleiten:

<STEP> = INT (9999/(ANZAHL ANWEISUNGEN + 1))

Entsteht bei Neunumerierung bedingt durch einen Parameter eine Zeilennummer > 9999, wird mit einer Schrittweite von 1 numeriert d.h. die erste Anweisung erhält die Zeilennummer "1", die zweite "2", etc.

Beispiel: 1000 RENUMBER 5

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

Das Programmiersystem

Fehler:

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

ERROR # 6 AT <ZL-NR.>

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

Das Programmiersystem

2.1.1.10 "RUN"- Kommando

Ausführung eines beliebigen Abschnittes des im Hauptspeicher (Partition) befindlichen BASIC-Programmes.

Aufbau:

[<ZL-NR.1>] RUN [<ZL-NR.2>]

Funktion:

<ZL-NR.1> = Zeilennummer innerhalb des Programmes, ab der gestartet werden soll. Diese Angabe ist nur zulässig, wenn 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 (Initial-RUN).

RUN = Kommandowort.

<ZL-NR.2> = Die Zeilennummer, an der das Programm angehalten werden soll. Ist sie erreicht, wird die Meldung:

STOP AT <ZL-NR.>

ausgegeben. Diese Anweisung ist noch nicht ausgeführt.
Ist <ZEILEN-NR.2> nicht angegeben, wird das Programm bis zu seinem logischen Ende ausgeführt. Dies gilt jedoch nur für den Fall, wenn im auszuführenden Programmteil keine "CHAIN"-Anweisung auftritt.

Beispiel: RUN 300

Starten eine Programmes am Programmanfang. Das Programm wird bei erreichen der Zeilennummer 300 angehalten.

Das Programmiersystem

Fehlermöglichkeiten:

Wird unter <ZL-NR.1> eine Größe angegeben, die im Programm nicht definiert ist, erfolgt die Meldung:

ERROR # 6

und das System befindet sich wieder im Basic-Kommandomodus.

Alle weiteren Fehlermeldungen, die während der Programmausführung möglich sind, werden im Kap. "Basic-Fehlerliste" beschrieben.

Mit Ausnahme einiger Fehler, die nicht maskierbar sind (z.B. NO SUCH LINE NUMBER), wird das Programm nicht abgebrochen.

2.1.1.11 "SIZE"- Kommando

Dieses Kommando dient dazu, die aktuelle Größe des im Hauptspeicher (Partition) befindlichen Programmes festzustellen.

Aufbau:

SIZE = Kommandowort.

Funktion:

Die Programmgröße wird in folgender Form ausgegeben:

SIZE= <ANZAHL X>WORDS,TOTAL=<ANZAHL Y>WORDS(<ANZAHL Z>KB)

<ANZAHL X> = Die Anzahl Worte, die durch die Anweisungen + Datenbereiche belegt sind (betrifft nur bereits dimensionierte Variablen).

<ANZAHL Y> = Die Anzahl Worte des Programmes (<ANZAHL X>) einschließlich der Hilfsbereiche (Stacks und Variablenliste).

<Anzahl Z> = Bezeichnet die minimale Partitiongröße (KB), in der das Programm ablaufen kann. Diese Angabe ist jedoch nur realistisch, nachdem alle Dimensionierungen des Programmes durchlaufen wurden.
Der hier angezeigte Wert kann im Kommando SAVE (Sichern des Programmes) angegeben werden (siehe auch Pkt.: 2.3.3 SAVE-Processor).

Das Programmiersystem

2.2 Direktausführung von "Business-Basic" Anweisungen

Basic-Anweisungen, die OHNE Zeilennummer eingegeben werden, werden nicht in das im Hauptspeicher stehende Programm insertiert, sondern jeweils sofort nach der Eingabe ausgeführt.

Aufbau:

<ANWEISUNG> = Basic-Anweisung (siehe auch Pkt. 2.1.1.1).

Funktion: = Syntaktische Prüfung, interpretative einmalige Ausführung der Anweisung.

Durch Direktausführung einer Anweisung wird das zur Zeit im Hauptspeicher stehende Programm nicht verändert. Der Datenbereich des Programmes steht uneingeschänkt zur Verfügung. Es können also Daten ausgegeben (PRINT), verändert (LET) und Variable angelegt werden.

Diese Funktion hat also die Aufgabe, den Programmtest per Dialog zu unterstützen.

Davon ausgenommen, weil nicht direkt ausführbar, sind:

- BUILD #
- CLOSE #
- DATA
- DEF
- END
- FOR
- GOSUB
- GOTO
- MAT READ #
- MAT WRITE #
- NEXT
- ON
- OPEN #
- READ #
- RETURN
- SEARCH #
- STOP
- WRITE #

Wird eine dieser Anweisungen eingegeben, erscheint die Meldung:

ERROR # 55

Das Programmiersystem

2.3 Die Programmbibliothek

Business-Basic Programme, die über einen längeren Zeitraum für wiederholte Verwendung gesichert werden sollen, werden in die Programmbibliothek auf einer Magnetplatte ausgelagert. Diese Bibliothek kann auf jeder beliebigen Platte des Systems stehen, also nicht nur auf der Systemplatte, die das Betriebssystem enthält.

Der Begriff Bibliothek ist in diesem Zusammenhang rein hypothetisch, da weder ein spezielles Verzeichnis noch ein geschlossener Plattenbereich vorhanden ist. Die Zugehörigkeit einer Datei zur Programmbibliothek wird lediglich durch ein Kennzeichen im Dateikennsatz (Header) der Programmdatei angezeigt.
Basic-Programme werden organisatorisch in dynamischen Dateien gespeichert, so daß Programm-Modifikationen ohne Bearbeitung der Gesamtdatei erfolgen können.

Um Basic-Programme entweder in die Bibliothek einzubringen oder aus der Bibliothek zu löschen, stehen die Processoren "SAVE" (Sichern) und "KILL" (Löschen) zur Verfügung.
Ein weiterer Processor, "COPY", ermöglicht das Kopieren von Programmen.

Die Funktionen und Arbeitsweisen dieser Processoren werden in den folgenden Kapiteln erläutert.

Das Programmiersystem

2.3.1 Der "RUN" - Processor

Der "RUN"-Procesor führt die mit dem "BASIC"-Processor erstellten Programme interpretativ aus. Er repräsentiert damit den sogenannten "Basic-Interpreter".

Der Aufruf von RUN erfolgt aus dem Systemkommandomodus und hat die Form:

RUN [<LU>/] <DATEINAME>

oder

[<LU>/] <DATEINAME>

Dabei wird das mit <DATEINAME> bezeichnete Programm in die Partition des aufrufenden Arbeitsplatzes geladen und ausgeführt.

<LU>/ muß nur angegeben werden, wenn die logische Platteneinheit, von der geladen wird, ungleich 0 ist.

Die Kommandovariante

RUN

gilt nur für die Ausführung des Programmes, das sich derzeit in der Partition des aufrufenden Arbeitsplatzes befindet.

Der RUN-Processor seinerseits ruft im Verlauf eines Programmes den BASIC-Processor beim Auftreten einer der folgenden Fälle auf:

Wenn im RUN-Kommando eine Stop-Bedingung angegeben ist (s. Pkt. 2.1.1.10 <ZL-NR.>).

Wenn eine Stop- oder End-Anweisung auftritt.

Wenn das Programmende erreicht ist.

Wenn ein nicht maskierbarer Programmfehler auftritt.

Wenn CTL C, ESC oder CTL Y gedrückt wird, ohne daß eine IF ERROR 0 -Bedingung wirksam ist.

2.3.2 Der "RUNMAT"-Processor

Der "RUNMAT"-Processor dient zur interpretativen Ausführung sämtlicher Matrix-Operationen (MAT-Anweisungen) mit Ausnahme von MAT READ # und MAT WRITE # (diese werden von RUN ausgeführt).

Der Aufruf von RUNMAT geschieht implizit durch den RUN-Processor.

Das Programmiersystem

2.3.3 Der "SAVE"- Processor

Das Sichern von Basic-Programmen erfolgt durch Auslagern des im Hauptspeicher stehenden Programmes auf eine Magnetplatte.

Das Auslagern wird vom "SAVE"-Processor durchgeführt, indem er den Inhalt des Hauptspeichers (Partition) in eine Datei auf einer Magnetplatte überträgt. Im Kennsatz dieser Datei vermerkt "SAVE", daß es sich um ein Basic-Programm handelt. Außerdem wird im Dateikennsatz die Eigentümerkennung des kommandoerteilenden Benutzers eingetragen.
Der "SAVE"- Procesor kann entweder aus dem Kommandomodus (SCOPE) durch den Bediener oder durch ein Basic-Programm per CHAIN-Anweisung aufgerufen werden.

Aufbau:

SAVE [‡<PART>‡] [<<PP>>] [<COST>] [<LU>/] [<DATEINAME>[!]]

Funktion:

SAVE = Kommandowort.

‡<PART>‡ = Partitiongröße in KB.
Das Partitionkonzept in NIROS 3.2 unterstützt die Verwaltung von Partitions unterschiedlicher Größe. Dazu ist es erforderlich, die für den Ablauf eines Basic-Programmes benötigte Partitiongröße im Kennsatz der Basic-Programmdatei abzustellen.
Ist die Angabe ‡<PART>‡ nicht vorhanden, setzt der "SAVE"- Processor in den Datei-Kennsatz eine Kennung, daß dieses Programm nur in einer Partition ablaufen kann, die so groß wie die größte konfigurierte Partition (Active-File-Größe) ist. Die Active-File-Größe, die zum Zeitpunkt des "SAVE" konfiguriert ist, kann unterschiedlich zu der Active-File-Größe sein, die zum Zeitpunkt des Programmlaufes konfiguriert ist. Letztere Größe muß jedoch ausreichen, um das Programm ausführen zu können.

Existiert im Dateikennsatz bereits eine Angabe zur Partition-Größe aufgrund

eines vorangegangenen "SAVE" mit der Option ǂ<PART>ǂ, so wird diese Angabe übernommen, wenn sie nicht erneut angegeben ist.
Wird eine kleinere Partitiongröße als erforderlich angegeben, erscheint später beim Laden des Programmes TRAP =0 oder TRAP ǂ30 am jeweiligen Arbeitsplatzbildschirm.

<<PP>> = Schutzstufen-Zuordnung

Die beiden inneren Klammern: <> müssen immer angegeben werden.

<PP>
- └ Benutzer der selben Privilegebene
- └ Benutzer niedriger Privilegebene

Eine Schutzmöglichkeit gegenüber Benutzern höherer Privilegebenen besteht nicht.

Folgende Schutzstufen können vergeben werden:

P = 0 = kein Schutz.

1 = Kopierschutz. Andere Benutzer werden daran gehindert, das Programm aufzulisten oder es unter anderem Namen sicherzustellen.

2 = Schreibschutz. Andere Benutzer werden daran gehindert, die Basic-Programmdatei zu löschen oder Attribute zu ändern.

3 = Kopier- und Schreibschutz.

4 = Leseschutz. Andere Benutzer werden gehindert das Programm zu benutzen.

5 = Lese- und Kopierschutz.

6 = Lese- und Schreibschutz.

7 = Lese-, Schreib- und Kopierschutz.

Wird beim Sichern eines bestehenden Programmes keine Schutzstufe angegeben, bleibt der bis dahin aktuelle Schutz wirksam.

Das Programmiersystem

Wird beim erstmaligen Sichern eines Programmes (neue Programme) kein Schutzkennzeichen vorgegeben, gilt standardmäßig <77> .

<COST> = Der Betrag, mit dem das Konto eines anderen Benutzers für den Zugriff auf diese Programmdatei belastet wird. Die dafür vorgeschriebene Form ist:

$DDD.PP

DDD = Vorkommabetrag von 0 - 999

PP = Nachkommabetrag von 0 - 99

Der eingegebene Nachkomma-Betrag wird immer auf einen vollen 10er Wert abgerundet.

Fehlt diese Angabe, bleiben bei bestehenden Programmen die bis dahin aktuellen Einträge gültig, wogegen bei neuen Programmen die Kosten standardmäßig auf 0 gesetzt werden.

<LU>/ = Die Nummer der logischen Einheit, auf die das Programm ausgelagert werden soll. Ist LU = 0 kann diese Angabe entfallen.

<Dateiname> = Der Name, unter dem das Basic-Programm auf der Magnetplatte ausgelagert werden soll. Diese Angabe entfällt, wenn das Programm bereits auf eine Magnetplatte ausgelagert ist und z.B. nach Programmänderungen unter dem gleichen Namen erneut gesichert werden soll.

! = Muß angegeben werden, wenn eine bereits bestehende Programmdatei ersetzt werden soll, z.B. ein nicht mehr benötigtes Programm soll durch ein neu erstelltes ersetzt werden, das den gleichen Namen hat.

Beispiele: SAVE

Ist die kürzeste Form zum Sichern eines Programmes, das bereits in der Bibliothek vorhanden ist. Das Programm muß zuvor in den Hauptspeicher geladen werden, um den Programmnamen dort zu vermerken. Alle

aktuellen Werte (Prot., Cost...)bleiben erhalten.

SAVE #10#<33> $100.00 FAKTUR

Eine erweiterte Form des SAVE-Kommandos, wodurch ein Programm, das 10 KB Partition benötigt, mit der Schutzstufe 33, dem Belastungsbetrag 100,00 und unter dem Namen FAKTUR ausgelagert wird.

Die möglichen Fehlermeldungen bei der Durchführung des "SAVE"-Kommandos:

? EMPTY FILE, NOT SAVED
Es ist kein Programm vorhanden, das gesichert werden kann (die Partition ist leer).

FILE IS COPY PROTECTED, NOT SAVED
Das zu sichernde Programm ist kopiergeschützt.

ILLEGAL FILENAME, NOT SAVED
Der angegebene Dateiname ist unzulässig (z.B. wurden unzulässige Zeichen benutzt).

FILENAME IN USE FOR DIFFERENT TYPE FILE, NOT SAVED
Der angegebene Dateiname ist bereits durch eine andere Dateiart belegt (z.B. Textdatei). Dieser Fehler tritt nur auf, wenn "!" angegeben ist.

FILENAME IN USE AND NO "!" SUPPLIED, NOT SAVED
Der angegebene Name wird bereits für eine andere Datei benutzt.

FILENAME IN USE, OLD FILE BEING BUILT OR REPLACED, NOT SAVED
Eine Datei mit dem angegebenen Namen wird gerade angelegt oder ersetzt.

FILENAME IN USE BY A DIFFERENT ACCOUNT, NOT SAVED
Der angegebene Dateiname ist bereits von einem anderen Konto belegt.

SYNTAX ERROR IN COST OR PROTECTION, NOT SAVED
Fehlerhafte Angabe bei Dateikosten (<COST>) oder Schutzstufe (<PROTECTION>).

LOGICAL UNIT NOT ACTIVE, NOT SAVED
Die unter <LU>/ angegebene logische Einheit ist nicht

aktiv (nicht installiert).

LOGICAL UNIT DOES NOT HAVE ENOUGH FREE BLOCKS, NOT SAVED
Auf der logischen Einheit, auf der das Programm gesichert werden soll, sind nicht genügend Plattenblöcke frei.

ACCOUNT DOES NOT HAVE ENGOUGH FREE BLOCKS, NOT SAVED
Das Konto des Benutzers verfügt nicht über genügend BLöcke, um das Programm zu sichern.

2.3.4 Der "KILL"- Processor

Zum Löschen von Basic-Programmen aus der Programmbibliothek steht der Processor "KILL" zur Verfügung. KILL entfernt den Namen des zu löschenden Programmes aus dem Platteninhaltsverzeichnis.
Der KILL-Processor kann entweder durch den Bediener aus dem Kommandomodus (SCOPE) oder durch ein Basic-Programm per CHAIN-Anweisung aufgerufen werden.

Aufbau:

KILL [<LU>/] <DATEINAME> [,[<LU>/] <DATEINAME>]$_{1}^{n}$

Funktion:

KILL = Kommandowort.

<LU>/ = Die Nummer der logischen Einheit, auf der das zu löschende Programm liegt. Ist LU = 0, kann diese Angabe entfallen.

<DATEINAME> = Der Name, unter dem das zu löschende Basic-Programm ausgelagert ist.

Folgende Fehler können bei Durchführung von "KILL" gemeldet werden:

FILE NOT FOUND #
Es ist keine Datei mit dem angegebenen Namen auf der Magnetplatte vorhanden.
Beim Löschen mehrerer Dateien bezeichnet das #-Symbol die Eingabeposition des Dateinamens, der nicht gelöscht werden kann.

FILE IS WRITE PROTECTED #
Die angegebene Datei ist gegen Überschreiben, d.h. auch gegen Löschen geschützt.
Beim Löschen mehrerer Dateien bezeichnet das #-Symbol die Eingabeposition des Dateinamens der nicht gelöscht werden kann.

LOGICAL UNIT NOT ACTIVE
Die mit <LU>/ bezeichnete logische Einheit ist nicht aktiv (nicht installiert).

Das Programmiersystem

ILLEGAL FILENAME
Der angegebene Dateiname ist nicht zulässig, weil z.B. unzulässige Zeichen benutzt wurden.

2.3.5 Der "COPY"-Processor

Das Kopieren von Basic-Programmdateien erfolgt mit dem "COPY"-Processor. COPY kann entweder durch den Bediener aus dem Kommandomodus (SCOPE) oder durch ein Basic-Programm per CHAIN-Anweisung aufgerufen werden.

Aufbau:

COPY [<LU1>/] <DATEINAME1> [!] { <- / = } [<LU2>/] <Dateiname2>

Funktion:

COPY	= Kommandowort.
<LU1>/	= Nummer der logischen Platteneinheit, auf der die Ziel-Datei entweder schon angelegt ist oder angelegt werden soll.
<DATEINAME1>	= Der Name der zu ersetzenden bzw. neu anzulegenden Datei (Ziel-Datei).
!	= Muß angegeben werden, wenn eine bereits bestehende Programmdatei ersetzt werden soll.
← oder =	= Trennt die Zieldatei-Angabe von der Quelldatei-Angabe.
<LU2>/	= Nummer der logischen Platteneinheit, auf der die zu kopierende Datei (Quell-Datei) liegt. <LU2> kann = <LU1> sein.
<DATEINAME2>	= Der Name der zu kopierenden Datei (Quell-Datei).

Beispiel: Kopieren eines Basic-Programmes mit dem Namen "FAKTOR" von LU 0 nach LU 1. Auf LU 1 existiert bereits ein Basic-Programm mit diesem Namen.

```
COPY 1/FAKTOR!=FAKTOR
        oder
COPY 1/FAKTOR!←FAKTOR
```

Das Programmiersystem

Folgende Fehler können bei der Durchführung von COPY gemeldet werden:

LOGICAL UNIT NOT ACTIVE
Die mit <LU2> bezeichnete logische Platteneinheit ist nicht aktiv. Zusätzlich zur Fehlermeldung wird der entsprechende Dateiname ausgegeben.

FILENAME IN USE AND NO "!" SUPPLIED
Der als Zieldatei angegebene Dateiname ist bereits belegt und "!" ist nicht angegeben.

FILE IS BEING BUILT, REPLACED OR DELETED
Versuch eines COPY in sich selbst.
Ziel- und Quelldatei sind identisch.

ILLEGAL FILE TYPE FOR DESTINATION
Als Quelldatei wurde ein Driver-Name angegeben.

SOURCE AND DEST NOT COMPATIBLE FOR A COPY
Als Zieldatei wurde ein Driver-Name angegeben.

FILENAME IN USE FOR A DIFFERENT TYPE FILE
Die Quell- oder Zieldatei ist keine Basic-Programmdatei.

FILENAME IN USE BY A DIFFERENT ACCOUNT
Der Name der Ziel-Datei ist bereits unter einem anderen Konto angelegt.

LOGICAL UNIT DOES NOT HAVE ENOUGH FREE BLOCKS
Die logische Platteneinheit, welche die Ziel-Datei aufnehmen soll, verfügt nicht über genügend freie Blöcke.

ACCOUNT DOES NOT NAVE ENOUGH FREE BLOCKS
Das Konto des Benutzers verfügt nicht über genügend freie Blöcke, um die Ziel-Datei aufzunehmen.

Das Programmiersystem

2.4 Aufruf der Processoren

Die Möglichkeiten des Aufrufes einer der Processoren:

- BASIC
- RUN
- RUNMAT
- SAVE
- KILL
- COPY

sind für jeden Processor unterschiedlich.

BASIC, RUN, SAVE, KILL und COPY können durch den Bediener aus dem Kommandomodus (SCOPE) aufgerufen werden. RUNMAT wird nur implizit durch RUN aktiviert, wenn eine "MAT-Anweisung mit Ausnahme von MAT READ # und MAT WRITE # zur Ausführung ansteht.

2.4.1 Der Aufruf des "BASIC" - Processors

- Aus dem Kommandomodus

BASIC {[<LU>/] [<DATEINAME>]}

BASIC = Kommandowort.

<LU>/ = Die Nummer der logischen Platteneinheit, auf der die Programmdatei, die geladen werden soll, liegt. Ist LU = 0, kann diese Angabe entfallen.

<DATEINAME> = Name des Programmes, das in die Partition des jeweiligen Arbeitsplatzes geladen werden soll.

Ist kein Dateiname angegeben, kann das zuletzt gelaufene Basic-Programm bearbeitet werden, vorausgesetzt, es ist noch in der Partition vorhanden.

Das Programmiersystem

- Aus dem "RUN" - Processor

  Der "RUN" - Processor ruft den "BASIC" - Processor auf, wenn:

  - im "RUN"-Kommando eine STOP-Bedingung angegeben ist (siehe auch Pkt. 2.1.1.10 <ZL-NR>).
  - eine STOP- oder END-Anweisung auftritt.
  - das Programmende erreicht ist ("READY").
  - ein nicht maskierbarer Basic-Fehler auftritt.
  - eine "CHAIN"-Anweisung nach "BASIC" auftritt.
  - CTL C oder ESC gedrückt wird und keine IF ERR 0-Bedingung wirksam ist.

2.4.2 Der Aufruf des "RUN" - Processors

- Aufruf aus dem Kommandomodus (SCOPE)

  RUN

  Wird nur "RUN" angegeben, gelangt das Basic-Programm zur Ausführung, das sich in der Partition des Arbeitsplatzes befindet.

  RUN [<LU>/] <DATEINAME> oder

  [<LU>/] <DATEINAME>

  starten das Programm mit dem angegebenen Namen, d.h. das angegebene Programm wird in die Partition des Arbeitsplatzes geladen. <LU>/ muß nur angegeben werden, wenn die Nummer der logischen Platteneinheit, von der geladen wird, ungleich 0 ist.

- Aufruf aus dem Basic-Processor durch das Basic-Kommando "RUN"

(Siehe Pkt. 2.1.1.10)

2.4.3 Der Aufruf des "SAVE" - Processors

Der "SAVE"-Processor kann aus dem Kommandomodus (siehe Pkt. 2.3.3) oder aus einem Basic-Programm per "CHAIN"-Anweisung (siehe Pkt. 3.3.3) aufgerufen werden.

2.4.4 Der Aufruf des "KILL" - Processors

Der "KILL"-Processor kann aus dem Kommandomodus (siehe Pkt. 2.3.4) oder aus einem Basic-Programm per "KILL"-Anweisung (siehe Pkt. 3.3.14) aufgerufen werden.

2.4.5 Der Aufruf des "COPY" - Processors

Der "COPY"-Processor kann aus dem Kommandomodus (siehe Pkt. 2.3.5) oder aus einem Basic-Programm per "CHAIN"-Anweisung (siehe Pkt. 3.3.3) aufgerufen werden.

3. Die Syntax

Die Syntax legt die Regeln zur Bildung von Zeichenfolgen in einer Sprache - in diesem Falle der Programmiersprache

"BUSINESS-BASIC " -

fest.

3.1 Die Metasprache

Zur Beschreibung der syntaktischen Regeln von "Bussines Basic" dient eine sogenannte Metsprache (Hilfssprache). Die Metasprache hat gegenüber der natürlichen Sprache den Vorzug der komplexeren Darstellung. Die in der Metasprache benutzten Symbole gehören nicht zur Sprache "Business Basic", es sei denn, es ist in einer Regel ausdrücklich angemerkt, daß ein bestimmtes Symbol zur "Business Basic"-Sprache gehört.

3.1.1 Die Symbole der Metasprache

Die syntaktischen Regeln von "Business Basic" werden definiert in Form von Zuordnungen, wobei eine Regel aus einer linken Seite, dem Zuordnungssymbol und einer rechten Seite besteht.

::= = Zuordnungssymbol.
Dieses Symbol der Metasprache trennt die linke von der rechten Seite einer Regel. Es bedeutet, daß die linke Seite der Regel durch die rechte Seite der Regel definiert wird.

<BEZEICHNUNG> = Syntaktische Einheit.
Eine in spitze Klammern eingeschlossene Bezeichnung bildet eine sogenannte syntaktische Einheit. Syntaktische Einheiten stehen stellvertretend für eine definierte Menge von Elementen der "Business Basic"-Sprache. So repräsentiert z.B. die syntaktische Einheit

<ANWEISUNG>

die Menge aller "Business Basic" Anweisungen.

Die linke Seite einer Regel besteht

Die Syntax

grundsätzlich aus einer einzigen syntaktischen Einheit. Syntaktische Einheiten können jedoch auch auf der rechten Seite einer Regel auftreten.

LET = Terminalsymbole.
Einzelzeichen oder Worte in Großbuchstaben (hier als Beispiel LET), die nicht in spitze Klammern eingeschlossen sind, bilden die sogenannten Terminal- oder Endsymbole.
Sie gehören zur "Business Basic"-Sprache und nicht zur Metasprache.
Terminalsymbole können nur auf der rechten Seite einer Regel auftreten.

[ ] = Eckige Klammern (Optionalklammern).
Eckige Klammern sind Symbole der Metasprache. Sie können nur auf der rechten Seite einer Regel stehen.
Eine Angabe innerhalb dieser Klammern kann auftreten, braucht es aber nicht.

Beispiel: <VARIABLENNAME> ::= <BUCHSTABE> [<ZIFFER>]

{ } = Geschweifte Klammern (Alternativklammer).
Geschweifte Klammern sind Symbole der Metasprache. Sie können nur auf der rechten Seite einer Regel stehen.
Von den in geschweiften Klammern angegebenen Zeichen kann nur jeweils eines alternativ auftreten.

Beispiel:

<VARIABLE> ::= { <NUMERISCHE VARIABLE> / <STRING VARIABLE> }

$[\ ]_M^N$ Indizes an Optional- oder Alternativklammern.
Der oben an eine Klammer angefügte Index gibt an, wie oft diese höchstens, und der unten angefügte Index gibt an, wie oft diese mindestens vorkommt.

3

Die Syntax

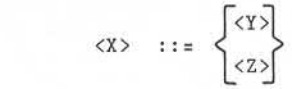
Beispiel:

Die Klammer:

beschreibt eine Zeichenkombination von 1 bis 3 Buchstaben und/oder Ziffern.

| Alternativstrich (entweder oder)
Aus Gründen der besseren Lesbarkeit wird in einigen Regeln statt

der Alternativstrich verwendet.
Z.B.



= <X> ::= <Y> | <Z>

Die Syntax

3.2 Das Basic-Programm

Ein Basic-Programm ist eine nach Zeilennummern sortierte Folge von Anweisungen.

<BASICPROGRAMM> ::= [<ZL-NR><ANWEISUNG>]$_{1}^{n}$

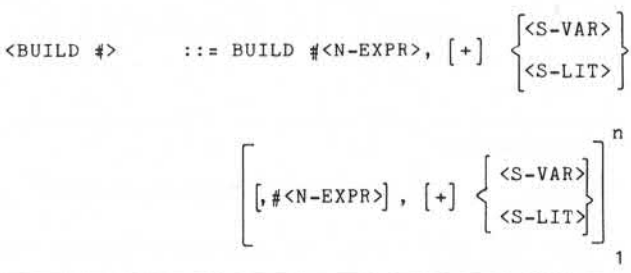
<Anweisung> ::=

<CALL>	<CHAIN>	<DATA>	<DEF>
<DIM>	<END>	<FOR>	<GOSUB>
<GOTO>	<IF>	<LET>	<NEXT>
<ON>	<RANDOM>	<READ>	<REM>
<RESTOR>	<RETURN>	<SIGNAL>	<STOP>
<BUILD>	<CLOSE>	<INPUT>	<KILL>
<LINK>	<DEALLO>	<PRINT>	
<PRINT #>	<READ #>	<OPEN #>	
<WRITE #>	<SEARCH #>		
<MAT READ #>	<MAT WRITE #>	<MAT>	

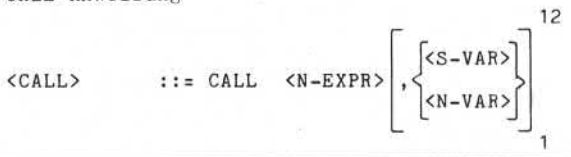
Die Syntax

3.3 Basic-Anweisungen

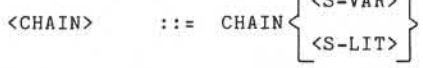
3.3.1 BUILD #-Anweisung



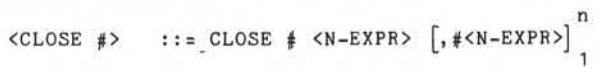
3.3.2 CALL-Anweisung



3.3.3 CHAIN-Anweisung



3.3.4 CLOSE #-Anweisung



	Die Syntax

3.3.5 DATA-Anweisung

<DATA> ::= DATA [{ 1 2 3 4 } %,] [{ + - }] <N-LIT> [, [{ + - }] <N-LIT>]n 1

3.3.6 DEF-Anweisung

<DEF> ::= DEF FN<BU>(<N-VAR>)=<N-EXPR>

3.3.7 DIM-Anweisung

<DIM> ::= DIM { <BU> [<ZI>] { $(<N-EXPR>) / [(<N-EXPR> [,<N-EXPR>])] } / { 1 2 3 4 } % }

[, { <BU> [<ZI>] { $(<N-EXPR>) / [(<N-EXPR> [,<N-EXPR>])] } / { 1 2 3 4 } % }]N 1

Die Syntax

3.3.8 END-Anweisung

<END> ::= END

3.3.9 FOR-Anweisung

<FOR> ::= FOR <N-VAR>=<N-EXPR> TO <N-EXPR> [STEP <N-EXPR>]

3.3.10 GOSUB-Anweisung

<GOSUB> ::= GOSUB <ZL-NR>

3.3.11 GOTO-Anweisung

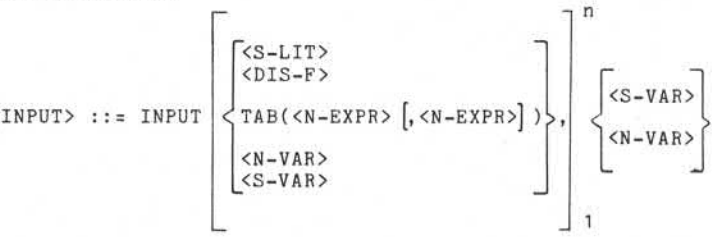
<GOTO> ::= GOTO <ZL-NR>

3.3.12 IF-Anweisung

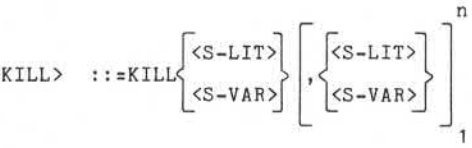
<IF> ::= IF { ERR O[<ANWEISUNG>] | <S-EXPR> [<V-OP><S-EXPR>] [<ANWEISUNG>] | <N-EXPR> [<V-OP><N-EXPR>] [<ANWEISUNG>] }

Die Syntax

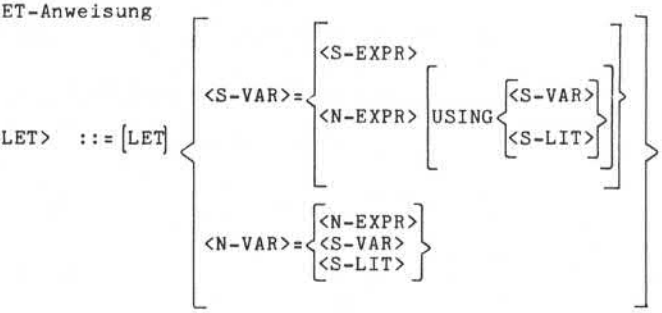
3.3.13 INPUT-Anweisung



3.14 KILL-Anweisung



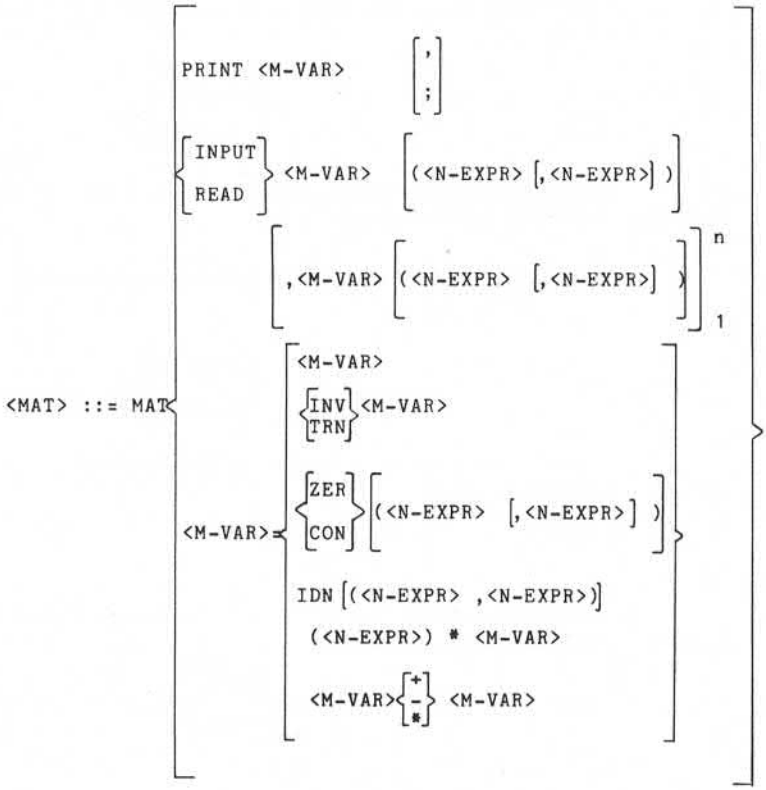
3.3.15 LET-Anweisung



3

Die Syntax

3.3.16 MAT-Anweisung



	Die Syntax

3.3.17 MAT-READ #-Anweisung

<MAT READ #>::= MAT READ #<N-EXPR>[,<N-EXPR> ,<N-EXPR>] ;

{<M-VAR> / <S-VAR>} [;]

3.3.18 MAT WRITE #-Anweisung

<MAT WRITE #>::= MAT WRITE #<N-EXPR> [,<N-EXPR> [,<N-EXPR>] ;

{<M-VAR> / <S-VAR>} [;]

3.3.19 NEXT-Anweisung

<NEXT> ::= NEXT <N-VAR>

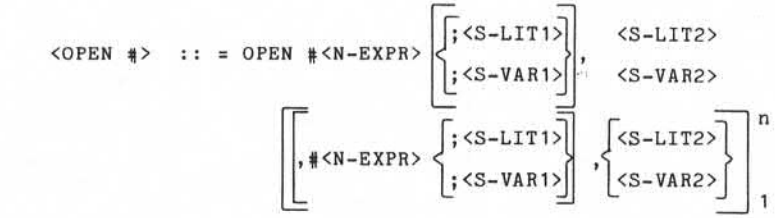
3.3.20 ON-Anweisung

<ON> ::= ON <N-EXPR> {GOTO / GOSUB} <ZL-NR> [,<ZL-NR>]$_{1}^{n}$

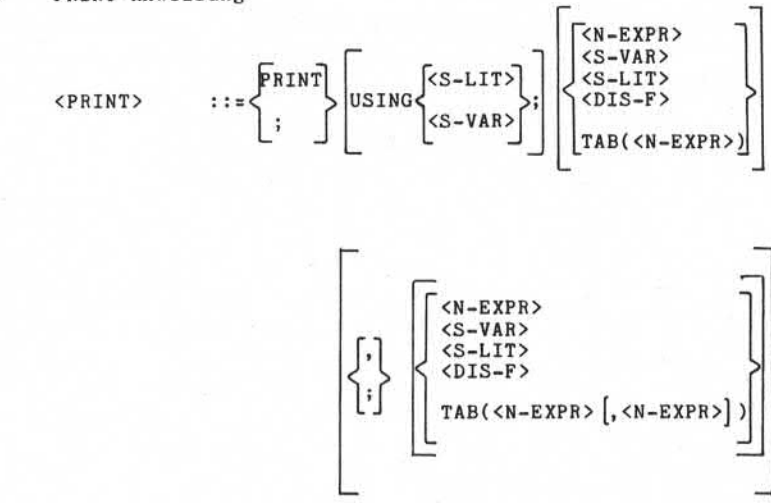
3

Die Syntax

3.3.21 OPEN #-Anweisung



3.3.22 PRINT-Anweisung



Die Syntax

3.3.23 Print #-Anweisung

<PRINT #> ::= [PRINT / ;] #<N-EXPR> [,<N-EXPR> [,<N-EXPR>]] ;

[USING [<S-LIT> / <S-VAR>] ;] [<N-VAR> / <S-LIT> / <S-VAR> / TAB(<N-EXPR>)] [[, / ;] [<N-VAR> / <S-LIT> / <S-VAR> / TAB(<N-EXPR>)]]

3.3.24 RANDOM-Anweisung

<RANDOM> ::= RANDOM <N-EXPR>

3.3.25 READ-Anweisung

<READ> ::= READ <N-VAR> [,<N-VAR>]$_{1}^{n}$

Die Syntax

3.3.26 READ #-Anweisung

<READ #>::= READ #<N-EXPR> [,<N-EXRP> [,<N-EXPR>]] ;

[{ <N-VAR> } , { <N-VAR> }]n [;]
[{ <S-VAR> } , { <S-VAR> }]1

3.3.27 REM-Anweisung

<REM> ::= REM [<ZCHN>]n 0

3.3.28 RESTOR-Anweisung

<RESTOR> ::= RESTOR [<N-EXPR>]

3.3.29 RETURN-Anweisung

<RETURN> ::= RETURN [<N-EXPR>]

3.3.30 SEARCH #-Anweisung

<SARCH #> ::= SEARCH #<N-EXPR>,<N-EXPR>,<N-EXPR>;<S-VAR>,
<N-VAR>,<N-VAR>

	Die Syntax

3.3.31 SIGNAL-Anweisung

<SIGNAL> ::= { 1,<N-EXPR>,<N-EXPR>,<N-EXPR> | 2,<N-VAR>,<N-VAR>,<N-VAR> [,<N-EXPR>] | 3,<N-EXPR> }

3.3.32 STOP-Anweisung

<STOP> ::= STOP

3.3.33 WRITE #-Anweisung

<WRITE #> ::= WRITE #<N-EXPR> [,<N-EXPR> [,<N-EXPR>]] ;

[<N-EXPR> | <S-VAR> | <S-LIT>] [, { <N-EXPR> | <S-VAR> | <S-LIT> }] n..1 [;]

3.3.34 LINK-Anweisung

<LINK> :: = LINK { <S-VAR> | <S-LIT> }

3.3.35 DEALLO-Anweisung

<DEALLO :: = DEALLO { <N-VAR> | <M-VAR> | <S-VAR> } [, { <N-VAR> | <M-VAR> | <S-VAR> }] 92..1

3.4 Elemente der Syntax

Im folgenden sind die in den Anweisungen referierten syntaktischen Elemente definiert.

3.4.1 Variable

3.4.1.1 String-Variable

Bezeichnung in der Metasprache: <S-VAR>

<S-VAR> ::= <BU> [<ZI>] $ [(<N-EXPR> [,<N-EXPR>])]

Definition: Dient zur Bezeichnung von Strings bzw. beliebiger Sub-Strings :

- kein Index
- 1 Index
- 2 Indizes

3.4.1.2 Numerische Variable

Bezeichnung in der Metasprache: <N-VAR>

<N-VAR> ::= <BU> [<ZI>] [(<N-EXPR> [,<N-EXPR>])]

Definition: Dient zur Bezeichnung von einfachen numerischen Variablen sowie zur Bezeichnung eines einzelnen Elements eines Vektors oder einer Matrix.

Kein Index = einfache numerische Var.
Ein Index = Vektorelement
Zwei Indizes = Matrixelement

	Die Syntax

3.4.1.3 Matrix-Variable

Bezeichnung in der Metasprache: <M-VAR>

<M-VAR> ::= <BU> [<ZI>]

Definition: Dient zur Bezeichnung von Matrizen in ihrer Gesamtheit (siehe MAT-Anweisungen).

3.4.2 Expressions (Ausdrücke)

3.4.2.1 String-Expressions

Bezeichnung in der Metasprache: <S-EXPR>

<S-EXPR> ::= { <S-VAR> / <S-LIT> } [, { <S-VAR> / <S-LIT> }] (n, 1)

Definition: Bezeichnet die Verkettung beliebig vieler String-Variable und/oder String-Literale.

3.4.2.2 Numerische Expressions

Bezeichnung in der Metasprache: <N-EXPR>

<N-EXPR> ::= [+ / -] { (<N-EXPR>) / <N-VAR> / <N-FUNC> / <N-LIT> } [{ + / - / * / / } { (<N.EXPR>) / <N-VAR> / <N-FUNC> / <N-LIT> }] (n, 1)

Definition: Bezeichnung von arithmetischen Ausdrücken. Sie setzen sich aus arithm. Operanden zusammen, die durch arithmetische Operatoren und Klammern miteinander verbunden sind.

3.4.3 Literale

3.4.3.1 String-Literal

Bezeichnung in der Metasprache: <S-LIT>

<S-LIT> ::=

Definition: Eine Folge von Oktal-Codes und/oder beliebigen darstellbaren Zeichen mit Ausnahme von <- und ". Diese Zeichenfolge wird in Doppelapostrophe eingeschlossen.

3.4.3.2 Numerisches Literal

Bezeichnung in der Metasprache: <N-LIT>

<N-LIT> ::=

Definition: Beliebiger vorzeichenloser Wert im Bereich von:

0 bis 99999999999999E+49 in Gleitkomma-Darstellung

oder bis zu 63 Ziffern in Dezimal-Darstellung.

Die Syntax

3.4.4 Funktionen

Bezeichnung in der Metasprache: <N-FUNC>

<N-FUNC> ::= { [<U-FUNC> / <F-NAME>] (<N-EXPR>) | LEN (<S-VAR>) | TAB (<N-EXPR> [,<N-EXPR>]) }

Definition: Eine Funktion, außer den E/A-unterstützenden TAB-Funktionen, liefert einen numerischen Funktionswert. Es werden Anwenderfunktionen und Standard-Basic-Funktionen unterschieden.

3.4.4.1 Anwender-Funktion

Bezeichnung in der Metasprache: <U-FUNC>

<U-FUNC> := FN<BU>(<N-EXPR>)

Definition: Eine durch den Anwender beliebig zu definierende Funktion (siehe auch DEF-Anweisung).

3.4.4.2 Funktions-Namen

Bezeichnung in der Metasprache: <F-NAME>

<F-Name> ::= SIN | COS | TAN | ATN | SQR | LOG | EXP
ABS | SGN | INT | FRA | RND | NOT | MAN
CHR | IXR | SPC | DET

Definition: Vom System zur Verfügung gestellte Funktionen.

3.4.5 Sonstige Elemente

3.4.5.1 Oktal-Codes

Bezeichnung in der Metasprache: <OCT-C>

<OCT-C> ::= <- <OZI> <- 3 3

Definition: Drei Oktalziffern, die zusammen einen Wert von 200 bis 377 (oktal) annehmen können. Oktal-Codes können nur in String-Literalen auftreten.

3.4.5.2 Zeilen-Nummer

Bezeichnung in der Metasprache: <ZL-NR>

<ZL-NR> ::= <ZI> 4 1

Definition: Eine ganze Zahl im Bereich von 1 bis 9999.

3.4.5.3 Vergleichs-Operanden

Bezeichnung in der Metasprache: <V-OP>

<V-OP> ::= | = | <> | > | < | >= | <=

Definition: Zeichen, die eine Vergleichsoperation zwischen zwei Operanden auslösen.

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

	Die Syntax

3.4.5.4 Display-Funktionen

Bezeichnung in der Metasprache: <DIS-F>

<DIS-F> ::= 'BEL' | 'BP' | 'BS' | 'CF' | 'CS' | 'LD'
'LI' | 'MP' | 'SB' | 'SF' | 'TB' | 'CR'
'CH' | 'CFF'

Definition: In Basic vordefinierte Funktionen zur Bildschirm-Programmierung (siehe auch Pkt. 7.2.2)

3.4.6 Liste der Syntax-Elemente

BEZEICHNUNG	META-BEZ.	DEFINITION
STRING-VARIABLE	<S-VAR>	<BU> [<ZI>] $ [(<N-EXPR> [,<N-EXPR>])]
NUMERISCHE VARIABLE	<N-VAR>	<BU> [<ZI>] [(<N-EXPR> [,<N-EXPR>])]
MATRIX-VARIABLE	<M-VAR>	<BU> [<ZI>]
EXPRESSION	<S-EXPR>	{ <S-VAR> / <S-LIT> } [, { <S-VAR> / <S-LIT> }]$_1^n$
NUM. EXPRESSION	<N-EXPR	[{ + / - }] { (<N-EXPR>) / <N-VAR> / <N-FUNC> / <N-LIT> } [{ + / - / * / / } { (<N-EXPR>) / <N-VAR> / <N-FUNC> / <N-LIT> }]$_1^n$

Die Syntax

BEZEICHNUNG	META-BEZ.	DEFINITION
STRING-LITERAL	<S-LIT>	" [<OKT-C> / <ZCHN> ausser "]$_{1}^{n}$ "
NUMERISCHES LITERAL	<N-LIT>	[.{<ZI>}$_{1}^{n}$ [E [+/-] {<ZI>}$_{1}^{2}$] / {<ZI>}$_{1}^{n}$ [.{<ZI>}$_{1}^{n}$] [E [+/-] {<ZI>}$_{1}^{2}$]]
FUNKTION	<N-FUNC>	{ <U-FUNC> / <F-NAME> } (<N-EXPR>) / LEN (>S-VAR>) / TAB (<N-EXPR> [,<N-EXPR>])
ANWENDER-FUNKTION	<U-FUNC>	FN<BU>(<N-EXPR>)
FUNKTIONS-NAME	<F-NAME>	SIN COS TAN ATN SQR LOG EXP ABS SGN INT FRA RND NOT MAN CHR IXR SPC DET

Die Syntax

BEZEICHNUNG	META-BEZ.	DEFINITION
OKTAL-CODES	<OCT-C>	<OZI> (3 / 3)
ZEILEN-NUMMER	<ZL-NR>	<ZI> (4 / 1)
VERGLEICHS-OPERANDEN	<V-OP>	= | <> | > | < | >= | <=
DISPLAY-FUNKTIONEN	<DIS-F>	'BEL' 'BP' 'BS' 'CF' 'CS' 'LD' 'LI' 'MP' 'SB' 'SF' 'CR' 'TB' 'CH' 'CFF'

Die Syntax

3.5 Der Zeichenvorrat

<BU>	::=	Buchstaben (ABC.....Z)
<ZI>	::=	Ziffern (0,1,2.....9)
<OZI>	::=	OKTAL-Ziffern (0,1,2....7)
<ZCHN>	::=	Die Gesamtheit aller darstellbaren Zeichen.

Datendarstellung/Datenformate

4

4. Datendarstellung/Datenformate

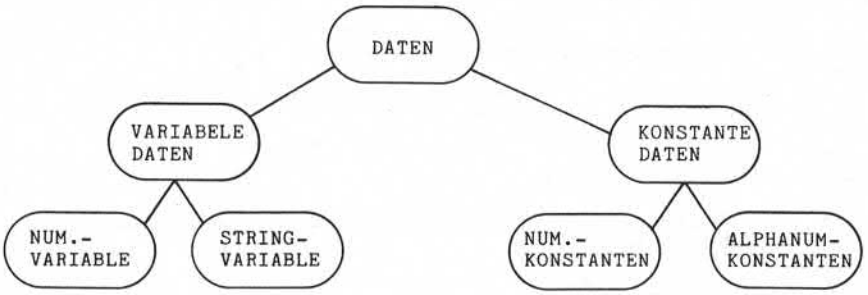
Daten sind von unterschiedlicher Struktur und verschiedenem Volumen.
Sie bestehen sowohl aus einzelnen Zeichen (Ziffern, Buchstaben und Sonderzeichen), Zahlen und Strings (Zeichenketten), als auch aus dem Inhalt ganzer Dateien.

Datenformate: In "Business Basic" wird unterschieden zwischen:

- variablen Daten und
- konstanten Daten.

Art der internen Datendarstellung:

- ASCII-Code
- Integer (BCD)
- Gleitkommazahl



Datendarstellung/Datenformate

4.1 Allgemeines zur Datenverwaltung

Die Verwaltung der Daten eines "Business-Basic" Programmes erfolgt durch den Basic-Interpreter "RUN". Der Programmierer hat keinen Einfluß auf die Lage der Daten innerhalb des Hauptspeichers (Partition). Die Zuweisung von Speicherplatz für eine Variable durch "RUN" erfolgt erst zum Zeitpunkt der Ausführung der Deklaration.

4.2 Variable

Unter dem Begriff Variable werden im folgenden die Bereiche (Felder) im Datenbereich, in die Daten eingesetzt oder in denen sie bearbeitet werden, also nicht die Werte selbst, verstanden.

Es wird unterschieden zwischen:

- numerischen Variablen und
- String-Variablen.

4.2.1 Einfache numerische Variable

Definition: Datenfeld zur Aufnahme von numerischen Werten.

4.2.1.1 Datendarstellung

In numerischen Variablen ermöglicht "Business-Basic" die Bearbeitung von

- BCD - Integer und
- Gleitkommazahlen.

Die Darstellung der Daten und die Größe der numerischen Variablen ist vom Aufbau des Hauptspeichers beeinflußt. Der Speicher des Systems 8870/1 hat Wort-Struktur.

1 Wort = 2 Byte = 16 Bit

Eine numerische Variable belegt im Speicher 1, 2, 3 oder maximal 4 Worte.
In 1-Wort-Variablen werden Werte grundsätzlich als BCD-Integer (BCD-Ganzzahl) mit Vorzeichen abgestellt.
In 2 bis 4-Wort-Variablen werden Werte grundsätzlich als Gleitkommazahl dargestellt.

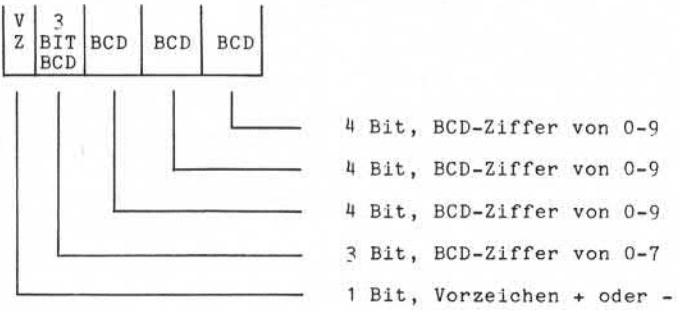
Datendarstellung/Datenformate

4

BCD-Integer

BCD-Integer = BCD-Ganzzahl mit Vorzeichen. Ein Integer belegt immer 1 Wort.

Darstellung:



Der maximale darstellbare Wert ist +/- 7999
Die Darstellung eines Dezimalzeichens (Komma oder Punkt) ist in 1-Wort-Variablen nicht möglich.

Gleitkommazahlen

Gleitkomma oder Gleitpunkt (floating point) ist eine halblogaritmische Zahlendarstellung.
Die Zahlen werden in Form:

a * b ↑ n

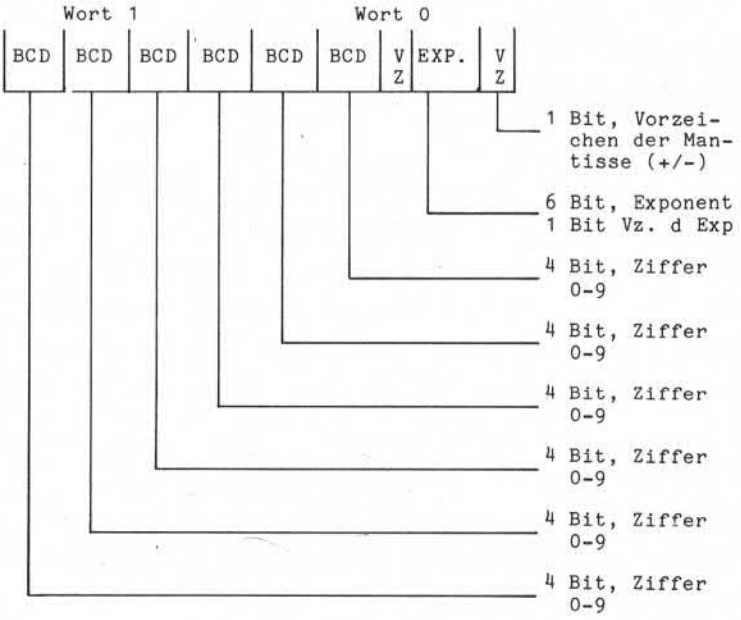
dargestellt.

a = Mantisse
b = Basis
n = Exponent (oder Charakteristik)

Variable zur Darstellung von Gleitkommazahlen belegen mindestens 2 und höchstens 4 Worte.

Datendarstellung/Datenformate

Darstellung:



Dieses Beispiel zeigt den Aufbau einer 2-Wort Variablen.
Der Auf bau der 3- und 4-Wort-Variablen ist entsprechend, nur sind 4 bzw. 8 BCD-Ziffern mehr darstellbar.
Die Mantisse (a) wird von den BCD-Ziffern gebildet.
Die Basis (b) ist nicht angegeben, da grundsätzlich 10 angenommen wird.
Der Exponent (oder Charakteristik) ist als Binärzahl verschlüsselt.Er setzt sich aus dem Exponent und dessen Vorzeichen zusammen.
Der maximal darstellbar Wert ist:

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

Datendarstellung/Datenformate

4

4.2.1.2 Formate

Bei numerischen Variablen wird zwischen vier Formaten unterschieden.
Diese Formate unterscheiden sich hinsichtlich ihrer

- Größe,
- Genauigkeit und
- Art der Darstellung.

Übersicht über die Formate von numerischen Variablen:

Format	Genauigkeit	Dezimalstellen	Darstellbarer Wert
1 Wort BCD-Integer	1	4	+/-7999
2 Worte GK-Zahl	2	6	+/- 0.999999E.+/- 63
3 Worte GK-Zahl	3	10	+/- 0.9999999999E.+/- 63
4 Worte GK-Zahl	4	14	+/- 0.99999999999999E.+/- 63

Erläuterung:

Format: = BCD-Integer oder Gleitkommazahl und Größe der Variablen in Worten.

Genauigkeit = Angabe, die bei der Deklaration von mumerischen Variablen (siehe auch Pkt. 4.2.4.1) erforderlich ist und die Größe der Variablen und damit auch die Rechengenauigkeit (bis maximal 14 signifikante Ziffern) festlegt.

Dezimalstellen = Anzahl der signifikanten Ziffern, die dargestellt werden können.

Darstellbar = Maximaler Wert, der dargestellt werden kann.

Datendarstellung/Datenformate

4.2.1.3 Arithmetischer Überlauf

Entsteht während der Verarbeitung ein Wert, der größer ist als in der aufnehmenden Variablen dargestellt werden kann, wird Basic-Fehler # 15

- ARITHMETIC OVERFLOW (SUCH AS DIVISION BY ZERO)

gemeldet.
Dieser Fehler kann mit der Anweisung IF ERR 0 maskiert werden.

Nach Auftreten dieses Fehlers enthält die Variable den maximal darstellbaren Wert (siehe Pkt. 4.2.1.2).

Entsteht in einer 2 bis 4-Wort Variablen ein Wert, der aus mehr signifikanten Ziffern besteht, als die Variable aufnehmen kann, werden die überlaufenden Ziffern rechts abgeschnitten. Der Exponent wird um die Anzahl der übergelaufenen Ziffern erhöht.

4.2.2 Vektoren/Matrizen

Vektoren:

Vektoren sind lineare Datenfelder, die aus 1 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:

1 bis n Reihen und
1 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.

Datendarstellung/Datenformate

4

Anmerkung: Soweit in der Syntax der Begriff <N-VAR> benutzt wird, bezeichnet er:

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

Matrix-Variable <M-VAR> bezeichnen Matrizen in ihrer Gesamtheit (siehe MAT-Anweisung).

4.2.3 String-Variable

Definition:

Datenfeld zur Aufnahme von 1 bis n beliebiger Zeichen.

4.2.3.1 Zeichendarstellung

Die Darstellung von alphanumerischen Daten, wie z.B. Texten, erfolgt in einem modifizierten ASCII-Code (siehe Codetabelle).
Jedes Zeichen belegt 1 Byte. Dabei muß beachtet werden, daß in jedem Byte zusätzlich zu dem Code des dargestellten Zeichens das Bit 7 immer "1" ist.

Beispiel: Alphanumerische Darstellung der Buchstaben"AB".

A							B							
L	L	0	0	0	0	0	L	L	L	0	0	0	0	L	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Bit

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 Anhang dieser Dokumentation behandelt.

Datendarstellung/Datenformate

4.2.3.2 Formate

Im Gegensatz zu numerischen Variablen bestehen für Strings keine eng begrenzten Formatvorschriften. Die Größe eines Strings kann vom Programmierer frei definiert werden und kann 1 bis n Byte betragen. Die maximale Länge eines Strings ist nur durch die Größe der Partition eingeschränkt.

4.2.4 Deklaration (Dimensionierung) von Variablen

Die Dimensionierung (Zuordnung/Reservierung) von Speicherplatz für eine Variable erfolgt zur Programmausführungszeit:

- bei einfachen numerischen Variablen, Vektoren und Matrizen durch die Anweisung "DIM" oder durch erstes Ansprechen der Variablen in einer beliebigen Anweisung.
- bei String-Variablen ausschließlich durch die Anweisung "DIM".

Bei der Dimensionierung werden im einzelnen folgende Spezifikationen festgelegt:

- bei numerischen Variablen:
  - Name der Variablen
  - Format und Genauigkeit
- bei Vektoren:
  - Name des Vektors
  - Anzahl Elemente
  - Format und Genauigkeit der Elemente
- bei Matrizen:
  - Name der Matrix
  - Anzahl Reihen
  - Anzahl Spalten
  - Format und Genauigkeit der Elemente
- bei String-Variablen:
  - Name der Variablen
  - Größe des Strings in Byte

Zum Zeitpunkt der Deklaration wird der jeweilige Variablen-Name von "RUN" in eine Variablen-Liste eingetragen.

4

Diese Liste kann die Namen von maximal 93 Variablen aufnehmen.

Nach durchgeführter Dimensionierung enthalten numerische Variablen sowie die Elemente von Vektoren und Matrizen den Wert "0". Strings sind mit Grenzzeichen gefüllt (binär 0).

Eine Variable darf in einem Programm nur einmal dimensioniert werden. Redimensionierung ist nicht möglich!

Davon ausgenommen sind:

Matrizen und Vektoren, wenn sich die Anzahl ihrer Elemente dadurch nicht vergrößert.

4.2.4.1 Einfache numerische Variable

Zur Dimensionierung von numerischen Variablen stehen zwei Möglichkeiten zur Verfügung:

a) explizite Dimensionierung:
Dimensionierung von Variablen mit der "DIM"-Anweisung.

b) implizite Dimensionierung:
Dimensionierung durch Angeben von Variablennamen in Anweisungen außer "DIM" wie z.B.:

"LET", "FOR", "IF" usw.

Die Größe der numerischen Variablen wird gesteuert durch einen 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 numerische Variable der Platz im Speicher reserviert.
Der Standard-Zustand dieses Schalters ist "2".
Eine Veränderung ist nur mit der "DIM"-Anweisung möglich.
Wird in einer "DIM"-Anweisung eine Formatangabe (1 bis 4) gemacht, nimmt der Schalter den entsprechenden Zustand an. Eine Formatangabe bleibt solange für folgende Dimensionierungen gültig, bis sie durch eine neue Formatangabe aufgehoben oder geändert wird.

Datendarstellung/Datenformate

Anmerkung: Die Schalterstellung kann durch dynamischen Programmablauf beeinflußt werden!

Explizite Dimensionierung:

Der Vorteil der Dimensionierung mit der "DIM"-Anweisung liegt darin, daß das gewünschte Format (1 bis 4 Worte) angegeben werden kann.
In der "DIM"-Anweisung sind die folgenden Angaben zu machen:

Format = Diese Angabe besteht aus einer Ziffer von 1 bis 4 und dem %-Zeichen.
Sie definiert die Größe der numerischen Variablen in Worten.

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

Diese Formatangabe gilt für alle im Anschluß angegebenen Variablen bis sie durch eine erneute Formatangabe aufgehoben/geändert wird.
Ist keine Formatangabe gemacht, wird das zuletzt angegebene Format bzw. das Standardformat (2%) eingesetzt.

Variablenname = Der Name, mit dem die Variable im Programm adressiert wird.
Der Name einer numerischen Variablen entspricht dem syntaktischen Element

Er besteht aus einem Buchstaben (A bis Z), an den wahlweise eine Ziffer angefügt werden kann.
ZB.: A, B1, Z, Z9, etc.

Beispiel: Es werden die Variablen mit den Namen "A" und "B" als 1-Wort, die Variablen "A1" und "B1" als 3-Wort und die Variablen "A2" und "B2" als 2-Wort Variablen dimensioniert.

```
DIM 1%,A,B,3%,A1,B1,2%,A2,B2
```

Die zuletzt angegebene Formatangabe bleibt solange gültig, bis sie entweder aufgehoben oder geändert wird.

Implizite Dimensionierung

Eine numerische Variable, die nicht explizit dimensioniert ist, wird vom Interpreter dimensioniert, wenn sie bei der Programmausführung zum ersten Mal auftritt. Das Format dieser Variablen wird durch die zur Zeit gültige Formatangabe bestimmt.

Beispiel: Die Anweisungsfolge

```
DIM 1%,A,B,C,3%
.
.
.
LET X = A*B
```

reserviert Platz für die 1-Wort Variablen A, B und C. Die Variable X, die nicht in einer "DIM"-Anweisung dimensioniert wurde, wird als 3-Wort Variable dimensioniert, da die Formatangabe 3% aktuell ist.

Datendarstellung/Datenformate

4.2.4.2 Vektoren/Matrizen

Wie bei numerischen Variablen ist es auch bei Matrizen und Vektoren möglich, sie implizit und explizit zu dimensionieren.
Für das Format der Elemente von Vektoren und Matrizen gilt die gleiche Regel wie für einfache numerische Variablen (siehe auch Pkt. 4.2.4.1).

Vektoren: Bei der Dimensionierung von Vektoren ist zusätzlich zur Formatangabe auch die Angabe der Anzahl der Elemente, die der Vektor haben soll, erforderlich, was durch die Angabe der Nummer des letzten Vektorelementes erreicht wird. Die Elemente eines Vektors werden von 0 bis n durchnumeriert. Das bedeutet, daß die Anzahl der Elemente eines Vektors der Nummer des letzten Elements + 1 entspricht.

Matrizen: Bei der Dimensionierung einer Matrix ist zusätzlich zur Formatangabe auch die Angabe der Anzahl der Reihen und der Spalten erforderlich.
Dies wird durch die Nummern der letzten Reihe und der letzten Spalte der Matrix erreicht. Die Numerierung der Reihen und Spalten läuft von 0 bis n. Das bedeutet, daß die Anzahl Spalten/Reihen der angegebenen Nummer + 1 entsprechen.
Die Anzahl der Elemente einer Matrix errechnet sich aus

— Anzahl Reihen * Anzahl Spalten. —

Explizite Dimensionierung

In der "DIM"-Anweisung sind folgende Angaben erforderlich:

Format Entspricht der Angabe für numerische Variablen. Das Format gilt für alle Elemente des Vektors/der Matrix.

Variablenname Entspricht der Namensbildung von einfachen numerischen Variablen.

Datendarstellung/Datenformate

4

Nummer des letzten Elements

Nur bei Vektoren. Es wird die Nummer des letzten Elements in "()" hinter dem Vektornamen angegeben.

Die maximale Anzahl Elemente ist nur durch die Partitiongröße eingeschränkt.

Nummer der letzten Reihe und der letzten Spalte

Nur bei Matrizen. Es wird die Nummer der letzten Reihe und der letzten Spalte in "()" hinter dem Namen der Matrix angegeben.

(R,S)

- S: Nummer der letzten Spalte
- R: Nummer der letzten Reihe

Die maximale Anzahl Elemente wird nur durch die Partitiongröße eingeschränkt.

Beispiele:

a) Dimensionierung eines Vektors mit 10 Elementen (0 bis 9) mit dem Namen V1. Jedes ELement soll 2 Worte groß sein.

```
DIM 3%,A,B,2%,X,V1(9)
```

Die Formatangabe 2% ist für die numerische Variable X und sämtliche Elemente des Vektors V1(9) gültig.

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

```
DIM A,B,4%,M1(5,5)
```

Die numerische Variable A und B werden so dimensioniert, wie der aktuelle Zustand des "Format-Schalters" angibt.
Die Formatangabe 4% gilt für alle Elemente der Matrix M1 und für alle folgenden Dimensionierungen bis zum Auftreten einer neuen Formatangabe.

Datendarstellung/Datenformate

Implizite Dimensionierung

Vektoren und Matrizen, die in einer beliebigen Anweisung außer "DIM" angesprochen werden und noch nicht in einer "DIM"-Anweisung dimensioniert wurden, werden von "RUN" zum Zeitpunkt ihres ersten Auftretens dimensioniert.

Vektor: Ein implizit dimensionierter Vektor besteht immer aus 11 Elementen (0 bis 10).

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

Das Format der Elemente wird durch die zur Zeit gültige Fomatangabe bestimmt.

Beispiele: a) Die Anweisungsfolge

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

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

b) Die Anweisungsfolge

```
DIM A,B,C,1%

.
.
.
LET M1(3,8) = C
```

dimensioniert die Matrix M1 mit 11 Reihen und 11 Spalten (jeweils 0 bis 10). Alle Elemente haben 1-Wort-Format.

Achtung: Es ist unbedingt zu beachten, daß bei der impliziten Dimensionierung von Vektoren ein Element > 0 und bei Matrizen zumindest eine der beiden Angaben (Zeile/Spalte) > 0 angesprochen wird: andernfalls wird eine einfache numerische Variable dimensioniert.

4

4.2.4.3 String-Variable

Die Dimensionierung von String-Variablen ist nur mit der "DIM"-Anweisung möglich.

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

In der "DIM"-Anweisung sind die folgenden Angaben zu machen:

Variablenname: Der Name, mit dem der String im Programm adressiert wird. Der Name einer String-Variablen besteht aus einem Buchstaben und dem $-Zeichen (z.B. A$).
Wahlweise kann an den Buchstaben eine Ziffer (z.B. A1$) angefügt werden.

Länge in Byte: Die Anzahl Zeichen, die der String maximal aufnehmen kann. Diese Angabe wird in "()" hinter dem Namen der String-Variablen gemacht, z.B. A1$(100).

Beispiel: Dimensionierung von zwei String-Variablen

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

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

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

Datendarstellung/Datenformate

4.2.5 Adressierung von Variablen

4.2.5.1 Einfache numerische Variable

Die Adressierung von einfachen numerischen Variablen erfolgt durch die Angabe des Variablennamens. Durch die in Kapitel 9 (Funktionen von Basic) beschriebenen

- mathematischen Funktionen und
- Zahlenmanipulations-Funktionen

ist es möglich, gezielt auf einzelne Bestandteile einer Variablen (z.B. auf den Exponenten) zuzugreifen.

4.2.5.2 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.
Darüberhinaus werden Matrizen von Matrix-Anweisungen (MAT) immer als ganze Einheit angesprochen. Dies erfolgt, indem der Name ohne Indizes angegeben wird (<M-VAR>).

Beispiel: Adressierung des Elements

Reihe 2
Spalte 4

in einer Matrix mit 5 Reihen und 5 Spalten mit einer "LET"-Anweisung:

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

Datendarstellung/Datenformate

4

Aufbau der Matrix

Spalte 0 - 4

Reihe 0 - 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
4	4,0	4,1	4,2	4,3	4,4

Datendarstellung/Datenformate

4.2.5.3 String-Variable

Bei der Bearbeitung von Strings sind folgende Regeln zu beachten:

- Daten in Strings werden immer linksbündig abgestellt und von links nach rechts bearbeitet.
- Die Bytes in einem String werden, beginnend mit 1, von links nach rechts durchnumeriert.
- Informationen in Strings werden durch Grenzzeichen begrenzt.

Grenzzeichenverarbeitung

Für einen String wird bei der Dimensionierung festgelegt, wieviele Zeichen (Byte) er maximal aufnehmen kann. Intern wird noch ein zusätzliches Byte (Null-Byte) für das Grenzzeichen reserviert.
Da ein String nicht in seiner ganzen dimensionierten Länge mit Daten belegt sein muß, stellt "RUN" hinter dem letzten Datenzeichen ein Grenzzeichen ab.

"RUN" stellt Grenzzeichen ab:

- Bei der Dimensionierung in allen Bytes.
- Nach einer "INPUT"-Anweisung hinter dem letzten eingegebenen Zeichen.
- Nach einer "LET"-Anweisung hinter dem letzten übertragenen Zeichen.

Ein String kann mehrere Grenzzeichen enthalten. Vorsicht bei Zugriffen auf Strings, da die Grenzzeichen hierbei nicht berücksichtigt werden und daher Daten zerstört werden können.
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.

Z.B.: PRINT A$

- Adressierung eines Strings ab einer anzugebenden Byte-Adresse bis zum nächsten Grenzzeichen (ab X-Index).

  Z.B.: IF A$(19) = X$ .....

- Adressierung eines Teilstrings innerhalb anzugebender Byte-Adressen ((X,Y)-Indizes).

  Z.B.: LET A$(3,7) = X$

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 (X,Y)-Indizes (im Quell- und/oder Ziel-String), wenn die Anzahl zu übertragender Zeichen der Anzahl aufnehmender Bytes entspricht.

Wenn der Ziel-String mit (X,Y)-Indizes adressiert wird, und die Anzahl zu übertragender Zeichen kleiner als der aufnehmende Bereich im Ziel-String ist, tritt der folgende Effekt auf:

- Der adressierte Bereich des Quell-Strings wird übertragen.
- Hinter dem letzten übertragenen Zeichen wird ein Grenzzeichen abgestellt.
- Alle rechts vom Y-Index stehenden Zeichen bis inclusive des ersten Grenzzeichens werden um die Anzahl Bytes nach links übertragen, die sich aus der Differenz zwischen den übertragenen Zeichen und dem angegebenen Zielbereich ergibt.
- Das neu eingesetzte Grenzzeichen wird dabei überschrieben.

Dieses Verhalten tritt auch bei der "INPUT"-Anweisung mit (X,Y)-Indizes auf.

Datendarstellung/Datenformate

Achtung: Wird eine Byte-Adresse (Index) angegeben, die außerhalb des adressierbaren Bereiches liegt:

- Index <= 0 oder
- Index > dimensionierte Länge,

wird Basic-Fehler #28 gemeldet.

Die LEN-Funktion

Diese Funktion dient zum Ermitteln von Grenzzeichenpositionen in einem String und übergibt die jeweils aktuelle String-Länge. "LEN" kann direkt ausgewertet bzw. der Wert von "LEN" in eine beliebige Variable übertragen werden.

Beispiel: Ermitteln von Grenzzeichenpositionen im String A$ und Ausgabe der Positionen am Bildschirm.

Inhalt A$:

X	X	X	X	GZ	Y	Y	Y	GZ	GZ

```
10 IF ERR 0 GOTO 60   /* FEHLERVERZWEIGUNG
20 LET A=A+1          /* X-INDEX
30 LET A=A+LEN A$(A) /* GRENZZEICHENPOS.
                        NACH A
40 PRINT A            /* GRENZZEICHENPOS.
                        AUSGEBEN
50 GOTO 20            /* NOCH KEIN STRING-
                        ENDE
60 IF SPC 8 <> 28 STOP /* FEHLER <> # 28
70 ......             /* STRING-ENDE ER-
                        REICHT
```

Am Bildschirm werden die Grenzzeichenpositionen:

5
9
10

ausgegeben.

Datendarstellung/Datenformate

4

Beispiele zur Adressierung von Strings:

a) Transport des Inhalts von A$ nach B$
Zustand vorher:

A$= | X | X | X | X | X | GZ |

LEN A$ = 5

B$ = | Y | Y | Y | Y | Y | Y | Y | Y | Y |

LEN B$ = 9

Anweisung: LET B$=A$

Zustand hinterher:

B$= | X | X | X | X | X | GZ | Y | Y | Y |

LEN B$ = 5
LEN B$(7) = 3

b) Transport einer Konstanten nach A$ ab Byte-Adresse 3:
Zustand vorher:

A$= | X | X | X | GZ | | | | | |

LEN A$ = 3

Anweisung: LET A$(3) ="YYYYY"

Zustand hinterher:

A$= | X | X | Y | Y | Y | Y | Y | GZ | |

LEN A$ = 7

Datendarstellung/Datenformate

c) Transport des Inhalts von A$ nach B$ von Byte 3 - 8

Zustand vorher:

A$= | X | X | GZ | | | |

LEN A$ = 2

B$= | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | GZ | |

LEN B$ = 8

Anweisung: LET B$(3,6) = A$

Zustand hinterher:

B$= | 1 | 2 | X | X | 7 | 8 | GZ | 8 | GZ | |

LEN B$ = 6
LEN B$(8) = 1

Bei diesem Beispiel tritt der Effekt auf, daß ein Teil des Zielstring-Inhaltes nach links übertragen wird. Soll dies vermieden werden, muß die Anweisung folgendermaßen aussehen:

```
LET B$(3,3+LEN A$-1) = A$
```

4.3 Konstante

Als Konstante werden die Daten bezeichnet, die bei der Programmerstellung definiert und während der Programmausführung nicht verändert werden.

Man unterscheidet:

- numerische Konstanten und
- alphanumerische Konstanten.

4.3.1 Numerische Konstanten

Eine numerische Konstante ist ein beliebiger Wert im Bereich von

0 bis 99999999999999E49 in Gleitkommadarstellung
oder
maximal 63 Ziffern in dezimaler Darstellung.

Stellt die Konstante einen negativen Wert dar, ist das Vorzeichen "-" unbedingt anzugeben.
Ist kein Vorzeichen angegeben, ist die Konstante positiv.

Beispiele: a) Vergleich, ob der Inhalt der numerischen Variablen "X" im Bereich von 1 bis 10 liegt.

```
IF X >= 1 IF X <=10 GOTO .....
```

b) Durchführen einer arithmetischen Operation mit einer numerischen Konstanten.

```
LET I = I+10
```

Datendarstellung/Datenformate

4.3.2 Alphanumerische Konstante

Eine alphanumerische Konstante ist eine Folge beliebiger Zeichen die durch "" begrenzt sind.

Beispiele: a) Eingabe und Vergleich dieser Eingabe, ob Y oder N eingegeben wurde.

```
INPUT TAB(X,Y),"FÜHRUNGSTEXT" A$
IF A$="Y" GOTO ....
IF A$="N" GOTO ....
GOTO ....
```

b) Abstellen der alphanumerischen Konstanten ABCD und dem BA-Funktionscode für BELL in einem String.

```
DIM X$(10)
.
.
.
LET X$ = "ABCD ←207← "
```

Nach der Durchführung der "LET"-Anweisung steht das Grenzzeichen in X$ auf der Position 6.
Die Abfrage der "LEN"-Funktion ergibt 5.

Anmerkung: Soll eines der Zeichen " oder ← in einer alphanumerischen Konstanten angegeben werden, muß dies in Oktal-Code geschehen.

Beispiel

```
PRINT "ABC ←337←←242←"

          ←337←  =  ←
          ←242←  =  "
```

Datendarstellung/Datenformate

4

4.4 Masken

Masken sind String-Literale und/oder String-Variablen die zur Aufbereitung des Inhaltes oder des Wertes von:

- numerischen Literalen
- numerischen Ausdrücken
- numerischen Variablen

dienen.

Die Aufbereitung mittels Masken ist in den Anweisungen

- LET
- PRINT
- PRINT #

möglich.

Während bei den Anweisungen PRINT und PRINT # die Aufbereitung jeweils direkt in den Ausgabepuffer erfolgt, erfordert die Anweisung LET die Angabe eines Ziel-Strings.
Zur Steuerung der Aufbereitung der Masken dienen die Steuerzeichen:

\# / + / - / $ / * / , / .

die jeweils eine bestimmte Aufbereitungsfunktion auslösen. Alle anderen Zeichen ungleich der Steuerzeichen werden 1:1 übernommen, dürfen allerdings nur am Anfang oder am Ende der Maske angegeben werden.

Abarbeiten von Masken

Masken werden von links nach rechts abgearbeitet. Die Anzahl der übertragenen Zeichen ist abhängig von der Anzahl der Masken-Steuerzeichen.
Ist die Anzahl signifikanter Ziffern im Quell-String größer als die Anzahl der Masken-Steuerzeichen die eine Ersetzung bewirken, werden nur "*" in den Ausgabepuffer, bzw. den Ziel-String übertragen.

Adressierung von Masken

Sind Masken in String-Variablen abgestellt, kann die Aufbereitung ab einer bestimmten Stelle in der Maske

Datendarstellung/Datenformate

begonnen werden. Die Maske wird immer bis zum Auftreten des Grenzzeichens bearbeitet. Eine Adressierung mit zwei Indizes ist nicht möglich.

Beispiel:

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

Datendarstellung/Datenformate

4

Steuerzeichen

Zeichen	Funktion
#	Übertragen von Ziffern mit Unterdrücken von führenden Nullen. Führende Nullen werden bei der Ausgabe durch Leerzeichen ersetzt. Das Steuerzeichen bleibt wirkungslos: - Bei Stellen hinter dem Dezimalkomma (Dezimalpunkt). - Wenn der Vorkommawert (Punkt) gleich Null ist.. Z.B. "#####"
+	Festes Vorzeichen +. Es muß, wenn es angegeben wird, unbedingt das erste Steuerzeichen in der Maske sein. Es bewirkt: Vorzeichen "+" bei der Aufbereitung eines positiven, und "-" bei der Aufbereitung eines negativen Wertes im Ausgabe-String. Z.B. "+###.##"
-	Festes Vorzeichen -. Es muß, wenn es angegeben wird, unbedingt das erste Steuerzeichen in der Maske sein. Es bewirkt die Übertragung eines: " " (Blank) bei der Aufbereitung eines positiven, und ein "-" bei der Aufbereitung eines negativen Wertes im Ausgabe-String. Z.B. "-###.##"
$	Festes Zeichen $ Es muß, wenn es angegeben wird, als erstes Steuerzeichen in der Maske stehen. Es bewirkt die Übertragung eines $-Zeichens an der entsprechenden Stelle im Ausgabe-String. Z.B. "$###"

Datendarstellung/Datenformate

Zeichen	Funktion
++ oder --	Gleitendes Vorzeichen + oder - Es bewirkt die Ausgabe eines Vorzeichens entsprechend der Regeln der festen Vorzeichen, bündig vor den übertragenenen Ziffern im Ausgabe-String. Führende Nullen werden durch Leerzeichen ersetzt. Wenn mehrere Steuerzeichen verwendet werden, muß dieses an erster Stelle stehen. Z.B. "++++.++"
$$	Gleitendes Zeichen $ Werden diese Zeichen verwendet, müssen sie ab der zweiten Position der Masken-Steuerzeichen angegeben werden. Sie bewirken die Ausgabe eines $-Zeichens bündig vor den ersten übertragenen signifikanten Ziffern im Ausgabe-String.Führende Nullen werden durch Leerzeichen ersetzt. Z.B. "BETRAG+$$$$.$$" Ist das zweite Maskensteuerzeichen ein "$-Zeichen muß das erste Steuerzeichen ein festes Vorzeichen (+/-) sein.
*	Schutzstern. Ersetzen aller führenden Nullen durch Sicherungssterne. Die Angabe der Schutzsterne muß ab der ersten oder zweiten Position der Maskensteuerzeichen erfolgen. Z.B. "$**,***.**" Steht das "*-Zeichen" innerhalb der Maskenangabe an zweiter Position, muß an erster Position entweder ein "$" oder ein festes Vorzeichen auftreten.
.	Einfügen des Deimalkommas oder -Punkt "." bzw. "," im Ausgabe-String. Das gewünschte Zeichen kann mit dem Dienstprogramm "SYSMOD" konfiguriert werden. Beispiel: "###.##"
,	Einfügen des Trennungssymbols "," oder "." in den Ausgabe-String. Das gewünscht Zeichen kann mit "SYSMOD" konfiguriert werden. Beispiel: "##,###,###.##" Ist, bevor das Trennungssymbol auftrat, noch keine signifikante Ziffer übertragen worden, wird in den Ausgabe-String anstelle des Trennungssymbols das aktuelle Erstzungszeichen ("␣" oder "*") übertragen.

Datendarstellung/Datenformate

Anmerkung:

Das Ersetzungs-Zeichen "#" ersetzt, bzw. überträgt in den Ausgabe-String soviele Zeichen wie in der Maske "=" angegeben sind.
Die Maske "###" bewirkt die Ausgabe von 3 Zeichen im Ausgabe-String. Enthält der aufzubereitende Wert mehr als 3 signifikante Ziffern, werden drei Schutzsterne in den Ausgabe-String übertragen.

Ist das erste Zeichen ein: "+" "-" "$" "*"
bewirkt das, daß dieses Zeichen einmal in den Ausgabe-String übertragen wird, und zwar:
- als festes Zeichen, wenn ein anderes Masken-Steuerzeichen (außer "," oder ".") folgt.

- als gleitendes Zeichen, wenn die folgenden Steuerzeichen gleich dem ersten sind.

Beispiel: LET A$=A USING "+++"

Es werden maximal ein "+" und zwei signifikante Ziffern in den Ausgabe-String übertragen. Enthält die Variable A mehr als zwei Ziffern, wird ein "+" und zwei Sicherungssterne übertragen.

Beispiele:

Anweisung: LET A$=A USING B$

Maske (B$)	Variable A	A$ nach LET
Datum: **,**,**	241277	Datum: 24.12.77
	010178	Datum: 1.01.78
	10	Datum: 10
	1234567	Datum: **.**.**
+$$$$$.$$	50	+ $50,00
	0,25	+ $,25
	12345,00	+12345.00
+$####	-50,50	+$ 50
	1000	+$1000
	10000	+10000

Programmstruktur

5. Programmstruktur

5.1 Aufbau eines Programmes

Ein zum Ablauf in die Partition geladenes Basic-Programm ist wie folgt aufgebaut:

Stacks = Arbeitsbereich für den Basic-Interpreter "RUN" für:

- Variablen-Liste
- Unterprogramm-Keller
- FOR-NEXT-Keller
  usw.

Tabelle der die Zeilennummern (in aufsteigender Folge) und einem Verweis auf die zugehörige Anweisung.

Die Anweisungen des Programmes in komprimierter Form. Teilweise Zwischencodes.

Datenbereich. Der Bereich wird erst zur Laufzeit des Programmes und beim Durchlaufen der "DIM"-Anweisung, bzw. bei impliziter Deklaration angelegt.

Programmstruktur

5.2 Abarbeiten eines Programmes

Ein Basic-Programm, das zum Ablauf kommen soll, muß in die Partition des jeweiligen Teilnehmers geladen sein. Das wird jeweils vom Betriebssystem durchgeführt, sobald ein Programm vom Bediener gestartet ist.

Hierzu bestehen zwei Möglichkeiten:

- Das Programm wurde soeben eingegeben oder geändert und wird per "RUN"-Kommando gestartet.
  In diesem Falle bfindet sich das Programm bereits in der Partition und das Laden entfällt.

- Ein gesichertes Programm wird aus der Programmdatei in die Active-File geladen.

Die Bearbeitung eines Programmes erfolgt über die Tabelle der Zeilennummern. Diese Tabelle wird sequentiell abgearbeitet. Der Zugriff auf die Anweisungen erfolgt über die in der Tabelle der Zeilennummern abgestellten Verweise. Die sequentielle Bearbeitung dieser Tabelle wird durch die Anweisungen:

- GOSUB
- GOTO
- NEXT
- RETURN
- END
- STOP

und unter bestimmten Umständen auch durch "FOR" (siehe Befehlsbeschreibung Pkt. 10.3.7) unterbrochen.

Diese Anweisungen führen zu einer Programmverzweigung. In diesem Falle wird die adressierte Zeilennummer nach der Methode des binären Suchens in der Tabelle der Zeilennummern gesucht. Die adressierte Zeilennummer ist entweder in der Anweisung (GOTO, GOSUB) vorhanden oder wird aus den entsprechenden Zwischenspeichern (FOR-NEXT-Keller, Unterprogramm-Keller) geholt.

Programmstruktur

Eine Besonderheit bezüglich der Programmabarbeitung stellt die Anweisung

IF ERR 0

dar. Mit dieser Anweisung wird festgelegt, wie im Programm beim Auftreten von Basic-Fehlern reagiert werden soll. Dies erfolgt durch Angabe einer Bedingung wie z.B.:

IF ERR 0 GOSUB 9000

Sobald im Programm ein Basic-Fehler auftritt, wird anstelle der verursachenden Anweisung, die in IF ERR 0 angegebene Anweisung ausgeführt. Im Beispiel oben würde ein Unterprogrammsprung zur Zeilennummer 9000 ausgeführt.

"RUN" kann nur eine IF ERR 0 Bedingung speichern. Das heißt wenn mehrere abgesetzt werden, ist jeweils die Bedingung der zuletzt durchlaufenen IF ERROR 0 -Anweisung aktuell.

Es ist zu beachten, daß zum Zeitpunkt der Fehlererkennung die verursachende Anweisung bereits teilweise abgearbeitet sein kann, was besonders bei den Anweisungen zu Problemen führen kann, in denen mehrere Operanden enthalten sind. Insbesondere sind dies:

- OPEN #
- CLOSE #
- KILL

Beispiel: CLOSE #2,#3,#4

Programmstruktur

5.3 Abarbeiten von Anweisungen

Die Anweisungen von Business-Basic werden von links nach rechts in der Reihenfolge:

- Schlüsselwort
- Operanden

interpretiert.
Während der Interpretation werden verschiedene Prüfungen durchgeführt und unter Umständen Aktionen ausgeführt, die abhängig vom derzeitigen Programm-Status sind.
Durch das Interpreter-Konzept zeigen viele Anweisungen ein dynamisches Verhalten. Daß dieselbe Anweisung völlig unterschiedliche Interpreter-Routinen auslösen kann, ist aus dem folgenden Beispiel zu ersehen:

```
10 LET A=A+1
20 GOTO 10
```

Erstes Durchlaufen von "LET A=A+1":

- Prüfung, ob die Variable A bereits dimensioniert ist.
- Dimensionierung der Variablen A
- Durchführung der Operation A=A+1

Weitere Durchläufe von "LET A=A+1":

- Prüfung, ob die Variable A bereits dimensioniert ist.
- Durchführung der Operation A=A+1.

Dieses dynamische Verhalten ist darin begründet, daß sämtliche Datenbereiche erst zur Laufzeit initialisiert bzw. angelegt werden. Bei übersetzten Programmen liegen die Datenfelder bereits von vornherein fest. Eine variable Dimensionierung aufgrund bestimmter datenabhängiger Ereignisse kann nur mittels Interpretation ermöglicht werden.

Arithmetische Ausdrücke werden nach den algebraischen Regeln der Mathematik bearbeitet und die jeweiligen Zwischenergebnisse abgespeichert.

Beispiel: LET A=B*(B+10)-5

1. Ausrechnen des Klammerausdruckes und das Ergebnis zwischenspeichern.
2. Das Zwischenergebnis mit dem Inhalt von B multiplizieren.
3. Vom Produkt danach 5 subtrahieren.

Wenn B = 5, entsteht als Ergebnis 70.

5.4 Schnittstellen zum Maschinencode

Die Schnittstelle zum Maschinencode bildet die Anweisung "CALL".
Sie dient zum Aufrufen und anschließender Ausführung von Maschinencode-Unterprogrammen.
Solche Unterprogramme erlauben die Durchführung von Funktionen, die im Sprachumfang von Business-Basic nicht vorhanden sind oder nur mit aufwendigen Basic-Routinen lösbar wären.
Eine Beschreibung aller verfügbaren Unterprogramme ist im Anhang dieser Dokumentation, Pkt. 11 vorhanden.

Programmstruktur

5.5 Segmentierung/Verkettung von Programmen

Mit der Anweisung "CHAIN" können beliebige Basic-Programme geladen und gestartet werden. Dadurch besteht die Möglichkeit, Programme, die als ganzes den verfügbaren Speicherbereich überschreiten würden, in Segmente zu zerlegen und nacheinander ablaufen zu lassen.

Eine weitere Anwendungsmöglichkeit ist das Festlegen von Zwangsabläufen. Das heißt, daß in Abhängigkeit von Programmzuständen bestimmte Programme nachgeladen werden, ohne daß der Bediener darauf Einfluß nehmen muß.

Bei der Durchführung einer "CHAIN"-Anweisung wird das angegebene Basic-Programm von der Platte in die Partition des Teilnehmers geladen. Das bedeutet, daß alle derzeitigen Daten in der Partition überschrieben werden.
Die Datenübergabe von Segment zu Segment muß also vom Programmierer vorgenommen werden.

Hierzu bestehen die folgenden Möglichkeiten:

- Übergabe der Daten im Common-Bereich.
  Für jeden Teilnehmer besteht ein Common-Bereich in der Größe von 512 Byte, der mittels "CALL" angesprochen werden.
  Es können also maximal 512 Bytes im Common-Bereich übergeben werden.
- Datenübergabe in Plattendatein. Diese Möglichkeit sollte nur in Ausnahmefällen angewandt werden, da sie relativ zeitaufwendig ist.
- Die Verwendung der Anweisungen "LINK" und "DEALLO" (s. Kap. 1.1.4 und 10.3.12).

Ein-/Ausgabekonzept

6 Ein-/Ausgabekonzept

6.1 Die Schnittstellen

Die Programmiersprache "Business-Basic" unter dem Betriebssystem "NIROS" bietet dem Programmierer in Abhängigkeit von der anzusprechenden Peripherie folgende Schnittstellen:

- Anweisungen fÜr interaktive Ein-/Ausgabe an den angeschlossenen Bildschirm-Arbeitsplätzen.
- Dateiverarbeitung mit Hilfe eines logischen IOCS (Input Output Control System) fuer:
  - Magnetplatte
  - Drucker
  - Lochkartenleser
- Physikalische Ein-/Ausgabe mit der Basic-Anweisung "CALL" für Magnetband und Magnetbandcassette.

6.1.1 Interaktive Ein-/Ausgabe

Ein-/Ausgabe für den Dialog zwischen dem Bediener am Arbeitsplatz und dem Programm.

Ein Arbeitsplatz besteht aus einem Bildschirm und einer Tastatur, eine Kombination für einen optimalen Dialog zwischen Programm und Bediener.

Dem Programm steht der Bildschirm zur Ausgabe von:

- Daten
- Führungsmasken
- Meldungen
- Anweisungen

zur Verfügung.

Die angeschlossene Tastatur ermöglicht dem Bediener die Eingabe von:

- Daten und
- Anweisungen.

Ein-/Ausgabekonzept

Dieser Dialog muß vom jeweils laufenden Basic-Programm gesteuert werden.

Die Programmierung selbst erfolgt mit den Basic-Anweisungen:

- PRINT (Ausgabe) und
- INPUT (Eingabe).

die durch 15 Bildschirm-Funktionen zur Steuerung der Ein-/Ausgabemöglichkeiten erleichtert werden können.

Im Gegensatz zu den anderen Peripheriegeräten sind die Ein-/Ausgabedaten des BA nicht als Dateien organisiert.

Ein BA kann nur von dem Programm angesprochen werden, das von ihm aufgerufen und gestartet wurde (Ausnahme CALL 4, siehe auch Pkt.: 8.3).

6.1.2 Dateikonzept unter dem IOCS

Für die Peripheriegeräte:

- Magnetplatte,
- Drucker und
- Lochkartenleser

bildet ein logisches IOCS (Input Output Control System) die Schnittstelle zwischen der Sprache Business-Basic und dem jeweiligen Peripheriegerät.
Das IOCS übernimmt die Verwaltung der Datenbestände. Die Daten werden vom IOCS im Rahmen von Dateien organisiert. Dem Programmierer wird der Zugriff auf die Daten ohne Rücksicht auf ihre physikalische Anordnung auf dem Datenträger nach rein logischen Gesichtspunkten ermöglicht.

Ein-/Ausgabekonzept

Zur Bearbeitung von Dateien stellt Business-Basic folgende Anweisungen zur Verfügung:

Anweisung	Magnetplatte	Drucker	Lochkartenl.
BUILD #	x	-	-
CLOSE #	x	x	x
KILL	x	-	-
MAT READ #	x	-	-
MAT WRITE #	x	-	-
OPEN #	x	x	x
PRINT #	x	x	-
READ #	x	-	x
SEARCH #	x	-	-
WRITE #	x	x	-

Ein-/Ausgabekonzept

6.1.2.1 Dateiorganisation

Die Organisation einer Datei bestimmt, wie die Datei auf einem Datenträger angelegt wird und welche Möglichkeiten des Zugriffs auf die Datensätze bestehen.
Das IOCS unterstützt folgende Arten der Dateiorganisationen:

- sequentiell
- formatiert
- relativ
- Index
- Text

Die physikalische Ein/Ausgabe-Schnittstelle (CALL) ermöglicht nur sequentielle Dateiorganisation, wobei die gesamte Dateiverwaltung vom Anwenderprogramm übernommen wird.

Übersicht der Möglichkeiten von Dateiorganisation auf den verschiedenen Datenträgern:

Datenträger

Org. art	Magnetpl.	Drucker	LK-Leser	M.-Band	M.-Cass.
Seq.	x	x	x	x	x
Form.	x	-	-	-	-
Relat.	x	-	-	-	-
Index	x	-	-	-	-
Text	x	-	-	-	-

Ein-/Ausgabekonzept

6.1.2.2 Record Pointer

Für Magnetplatten- und Druckerdateien wird vom IOCS ein Satzzeiger (Record-Pointer) geführt, sobald eine Datei eröffnet ist. Greifen mehrere Teilnehmer auf dieselbe Datei gleichzeitig zu, wird der Zeiger für jeden Teilnehmer individuell geführt.
Der Record-Pointer repräsentiert die augenblickliche Zugriffsposition in der Datei, d.h. die relative Satznummer des in der Datei zuletzt bearbeiteten Datensatzes. Der Inhalt des Record-Pointers kann mit der Basic-Funktion "CHF" abgefragt werden.

Bei Textdateien wird die Satznummer des aktuellen Daten-Blocks (Sektors) und die Byte-Position innerhalb dieses Blocks, die auf das zuletzt bearbeitete Zeichen folgt, geführt.

Ein-/Ausgabekonzept

6.1.2.3 Das Kanalkonzept

Die Bearbeitung der Dateien wird über Datenkanäle gesteuert. Jedem Programm stehen dazu max. 14 Kanäle (0 bis 13) zur Bearbeitung zur Verfügung.

Beim Einsatz von TAMOS sind die Kanäle 0 und 1 belegt.

Pro Kanal kann jeweils nur eine Datei eröffnet werden.
Plattendateien können auf verschiedenen Kanälen gleichzeitig eröffnet werden (File-sharing).
Dies gilt unabhängig davon, ob eine Datei von mehreren Programmen (Teilnehmern) oder innerhalb eines Programmes mehrfach auf verschiedenen Kanälen eröffnet wird.

Dagegen ist ein Eröffnen von Drucker- und Lochkartenleser-Dateien nur einmal möglich, da diese Peripheriegeräte nur von jeweils einem Programm benutzt werden können.

Bildschirmarbeitsplätze belegen keine Kanäle.

Ein Bildschirmarbeitsplatz bleibt immer dem Programm zugeordnet, das er gestartet hat.

Ein-/Ausgabekonzept

6.1.2.4 Definition von Dateien

Dateidefinitionen (Satzlänge, Organisationsarten, etc.) sind bei der Programmierung mit Business-Basic nicht erforderlich.

- Bei Plattendateien werden die zur Bearbeitung notwendigen Daten vom IOCS aus den Dateikennsätzen übernommen.
- Bei Lochkartendateien werden grundsätzlich Standardwerte eingesetzt.
- Bei Druckerdateien werden die erforderlichen Parameter in der OPEN # - Anweisung an das IOCS übergeben.

6.1.2.5 Der Dateiname

Es wird unterschieden zwischen:

- Namen für Plattendateien und
- Namen für Peripheriedateien

Der Dateiname ist bei der Verarbeitung einer Datei nur einmal, und zwar bei der Dateieröffnung, anzugeben.

Beispiel: OPEN # 2, "FAKTUR"

Diese Anweisung eröffnet auf Kanal 2 eine Plattendatei mit dem Namen FAKTUR.

Für alle weiteren Anweisungen, die der Bearbeitung der Datei dienen, ist nur noch die Angabe der Kanalnummer erforderlich.

Der Dateiname kann in einer String-Variablen oder als alphanumerische Konstante in einer Basic-Anweisung angegeben werden.

Ein-/Ausgabekonzept

Name für Plattendateien

Bei Magnetplattendateien besteht der Dateiname aus 1 bis 14 Buchstaben und/oder Ziffern. Als einziges Sonderzeichen ist der "." zugelassen. Das erste Zeichen des Dateinamens muss jedoch unbedingt ein Buchstabe sein.

<DATEINAME> :: = <BU> { [<BU> / <ZI> / .] 13 / 1 }

Ein Dateiname ist frei wählbar, muß jedoch eindeutig sein.

Name für Peripheriedateien

Im Gegensatz zur Magnetplatte sind die Dateinamen für die

- Drucker- und
- Lochkartenleserdateien

vom System festgelegt, was darin begründet ist, daß auf diesen Peripheriegeräten nur jeweils eine Datei eröffnet sein kann.
Als Dateiname ist der Name des jeweiligen Kanalprogrammes anzugeben:

- $CRD = Lochkartenleser (500 K/min)
- $CRD1 = Lochkartenleser (90 K/min)
- $LPT = Nadeldrucker, linker Vorschub
- $LPTR = Nadeldrucker, rechter Vorschub
- $LPT1 = Zeilendrucker
- $LPTS = 2. Nadeldrucker, linker Vorschub
- $LPTRS = 2. Nadeldrucker, rechter Vorschub
- $RPLn = ND an Kanal n, linker Vorschub
- $RPRn = ND an Kanal n, rechter Vorschub
- $RPFn = ND an Kanal n, Formulareinzug

Ein-/Ausgabekonzept

6.1.2.6 Das Konzept der logischen Einheiten

Bei der Bearbeitung von Dateien auf Magnetplatten, speziell beim Anschluß mehrerer Laufwerke und der Bearbeitung mehrerer Problemkreise gleichzeitig, ist es vorteilhaft, wenn die Plattenpacks nicht auf dem gleichen physikalischen Laufwerk liegen. Dies wird durch die Vergabe von logischen Einheiten-Nummern (LU-Nummern) erreicht. Die LU-Nummern werden mit dem "INSTALL"-Processor zugewiesen. Gleichzeitig wird vom Betriebssystem vermerkt, welche physikalische Einheit dieser logischen Einheit entspricht. Die Systemplatte wird immer mit der logischen Einheiten-Nummer 0 angesprochen.

In Basic-Programmen ist die LU-Nummer beim Eröffnen einer Datei in der Anweisung OPEN # , und zwar in der Form:

LU/

bündig vor dem Dateinamen anzugeben.

Beispiel: Eröffnung der Datei "FAKTUR" auf Kanal 2. Die Datei liegt auf LU 3.

OPEN # 2,"3/FAKTUR"

Wird eine Datei auf der Systemplatte (LU=0) angesprochen, kann diese Angabe entfallen.

6.1.3 Physikalische Ein-/Ausgabe mit CALL

Die physikalische Ein-/Ausgabe erlaubt die Programmierung von Ein-/Ausgaben für Peripheriegeräte, die nicht vom IOCS unterstützt werden. Die Programmierung erfolgt mit der Anweisung "CALL".

Die gesamte Datenträger- und Datenverwaltung muß vom Basic-Programm übernommen werden.

Ein-/Ausgabe-Programmierung

7 Ein-/Ausgabe-Programmierung

7.1 Die Anweisungen zur Ein-/Ausgabe

Zur Programmierung der Ein-/Ausgabe dienen die Anweisungen:

- INPUT = Eingabe über Tastatur
- PRINT = Bildschirm-Ausgabe
- OPEN # = Datei-Eröffnung
- READ # = Datensatz lesen
- WRITE # = Datensatz schreiben
- PRINT # = Zeilen/Datensatz schreiben
- MAT READ # = Daten in Felder einlesen
- MAT WRITE # = Daten aus Feldern ausgeben
- CLOSE # = Datei/Kanal schliessen
- BUILD # = Datei erstellen
- SEARCH # = Bearbeitung der Indexbereiche von Index-Dateien
- CALL = Physikalische Ein-/Ausgabe
- KILL = Löschen von Dateien

Die Beschreibungen der Anweisungen:

- BUILD #
- SEARCH #
- CALL
- KILL

erfolgt gesondert in den Kapiteln:

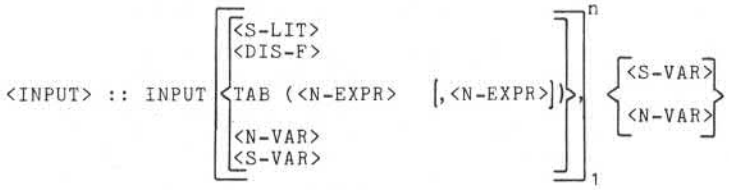
- BUILD # Pkt. 7.3.4 (Erstellen von Plattendateien)
- SEARCH # Pkt. 7.3.5.4 (Zugriff auf Index-Dateien)
- CALL Pkt. 7.6.1 (Funktionen des CALL 70)
  Pkt. 11.21 (CALL 80)
- KILL Pkt. 7.3.6 (Löschen von Dateien)

Ein-/Ausgabe-Programmierung

7.1.1 Die Anweisung "INPUT"

Eingabe von Daten über die Tastatur in beliebige Variablen. Zusätzlich können beliebige Führungstexte und/oder beliebige Bildschirmfunktionen ausgegeben werden, d.h, die eingegebenen Daten erscheinen ab der aktuellen Cursorposition am Bildschirm.

Syntax



Funktion:

INPUT = Anweisung.

<S-LIT> = Beliebiges String-Literal.
Dient zur Ausgabe von Führungstexten.
<S-LIT> wird ab der aktuellen Cursorposition am Bildschirm angezeigt.

<DIS-F> = Beliebige Display-Funktion.
Es ist die Angabe der von Pkt. 7.2.2.2 bis 7.2.2.15 beschriebenen Funktionen möglich.

TAB = Tabulation des Cursors auf eine beliebige Bildschirmposition. Es kann:

- Eine Spalte in der aktuellen Zeile angegeben werden:
  TAB (<N-EXPR>)
- Eine Spalte und Zeile angegeben werden:
  TAB (<N-EXPR>,<N-EXPR>)

Die Spaltenangabe muß in dem Bereich von 0 bis 79, die Zeilenangabe im Bereich von 0 bis 24 liegen, andernfalls ist das Ergebnis undefiniert.

Wird nur die Position innerhalb der Zeile

angegeben, ist zu beachten, daß diese größer als die aktuelle Position sein muß.

<N-VAR> = Variable, in die eingegeben werden soll.
oder
<S-VAR> Es können in einer "INPUT"-Anweisung mehrere Eingabevariable angegeben werden.

Sämtliche Operanden sind durch Komma zu trennen. Sind mehrere Eingabevariablen angegeben, muß jede Eingabe mit der Taste "CR" ausgelöst werden.
Als letzter Operand muss eine Eingabevariable angegeben sein!
Beim Auftreten einer "INPUT" - Anweisung wird das Programm so lange auf "INPUT - mode" geschaltet, bis die Eingabe beendet ist, d.h, daß das Programm in Wartezustand versetzt wird und bis zum Ende der Eingabe keine Rechnerzeit mehr erhält.

Jedem Bildschirmarbeitsplatz ist im System ein Eingabepuffer in der Größe von 256 Byte zugeordnet, wovon 12 Byte als Voreingabepuffer reserviert sind. Von diesen 12 Byte dienen 11 zur Pufferung von eingegebenen Daten und ein Byte als Zeiger im Voreingabepuffer.
Das ermöglicht, ausserhalb des "INPUT - mode" bis zu 11 Zeichen einzugeben.
Werden mehr als 11 Zeichen eingegeben, ertönt der akustische Signalgeber und die zuviel eingegebenen Zeichen werden nicht übernommen.
Beim Betätigen einer der Auslösetasten außerhalb des "Input-mode" ertönt ebenfalls der akustische Signalgeber und die Eingabe wird 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, wird das durch den akustischen Signalgeber gemeldet und die Zeichen nicht übernommen.
Steht der Cursor in einem Hintergrundfeld, können keine Zeichen eingegeben werden.
Ist ein Eingabefeld auf dem Bildschirm durch Hintergrundzeichen begrenzt und man versucht zu viele Zeichen einzugeben, wird die gesamte Eingabe gelöscht.

"INPUT" bietet zusätzlich die Möglichkeit, Führungstexte und BA - Funktionen vor der eigentlichen Eingabe auszugeben. Das bedeutet, daß die "INPUT" - Anweisung in beschränktem Maße auch Ausgabefunktionen übernehmen kann.

Ein-/Ausgabe-Programmierung

Führungstexte können allerdings nicht in Variablen stehen, sondern müssen als alphanumerische Konstante angegeben werden.

Beispiel:
1) Ausgabe des Führungstextes "RABATT" ab Spalte 0 in Zeile 10.
2) Ausgabe von 2 Punkten zur Anzeige der maximal einzugebenden Anzahl Stellen ab Position 30.
3) Ausgabe eines Hintergrund-Blanks zur Begrenzung des Eingabefeldes.
4) Umschalten auf Vordergrund.
5) Tabulation auf das erste Zeichen des Vordergrundfeldes.
6) Eingabe in die Variable "R".

```
INPUT TAB(0,10),"RABATT",TAB(30),"..",'SB'," ",'SF',TAB(30,10)R
      1                  2           3    4   5               6
```

Ein-/Ausgabe-Programmierung

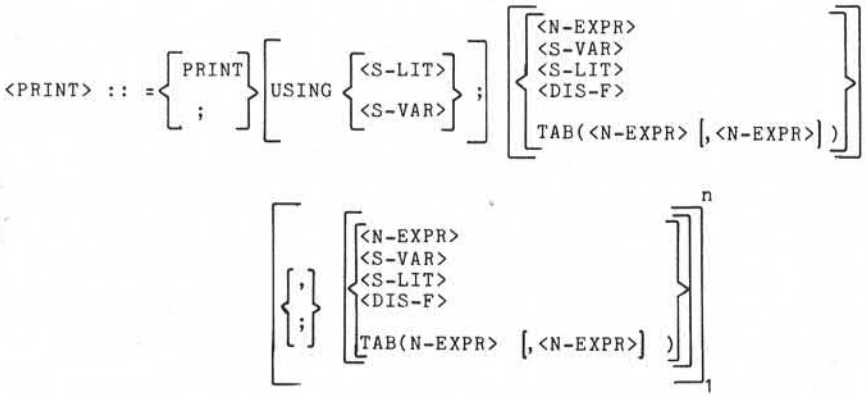
7.1.2 Die Anweisung "PRINT"

Print ermöglicht die Ausgabe von:

- String-Variablen
- numerischer Variablen
- Konstanten
- BA - Funktionen

in beliebiger Reihenfolge am Bildschirm.

Syntax:



Funktion:

PRINT = Anweisung.
; Bei der Eingabe der Anweisung kann statt "PRINT" auch ein Semikolon ";" angegeben werden.

USING <S-LIT> / <S-VAR> = Mit dieser Angabe wird eine Aufbereitungsmaske vorgegeben, die die Aufbereitung aller mit der Anweisung auszugebenden numerischen Variablen steuert.

Die Maske selbst, wird in <S-LIT> oder <S-VAR> vorgegeben.
"USING" darf in einer "PRINT"-Anweinur einmal angegeben werden.

Der Aufbau von Masken und die Funk-

tionen der zur Verfügung stehenden Masken - Steuerzeichen sind unter Pkt. 4.4 beschrieben.

Die "USING" - Angabe muss vom Folgeoperanden unbedingt durch ein Semikolon ";" getrennt sein.

<N-EXPR> = Eine beliebige Folge von:
<S-VAR> - String - Variablen
<S-LIT> - numerischen Ausdrücken
<DIS-F> - Konstanten
TAB - BA-Funktion

Die Ausgabe von Gleitkommazahlen bzw. BCD-Ganzzahlen, die nicht über Masken gesteuert werden, erfolgt ab der aktuellen Cursorposition in der Form:

- Vorzeichen (1 Zeichen), für "+" wird Blank "␣" und für "-" wird Minus "-" ausgegeben.
- Im Anschluß an das Vorzeichen die signifikanten Ziffern. Führende Nullen werden unterdrückt.
- Im Anschluß an die zuletzt ausgegebene Ziffer wird ein Blank " " ausgegeben.

Enthält die auszugebende Variable keine signifikanten Ziffern, wird eine Null "0" ausgegeben.
Bei der Ausgabe von Gleitkommazahlen ist besonders zu beachten, daß die Ausgabe ohne Maskenangabe ("PRINT USING") im Gleitkommaformat erfolgt, wenn:

- die Anzahl der signifikanten Ziffern 14 überschreitet.
- ein Wert kleiner 0,1 entsteht.

Beispiel: Der Wert 0,01 wird als
1E -2
ausgegeben.

Der Wert 999999999999999 wird als
9,9999999999999E +14
ausgegeben.

Anmerkung:

Formulartrennzeichen für die Druckausgabe, außer "CR", bewirken keine Ausgabe am Bildschirm.

Ein-/Ausgabe-Programmierung

Als Trennzeichen zwischen den Operanden dienen "," (Komma) und ";" (Semikolon).

Trennzeichen "," (Komma)

Nach der Bearbeitung des vorhergehenden Operanden wird der Cursor auf die nächsthöhere, ohne Rest durch 15 teilbare Spaltenposition tabuliert. Ist dieser Wert größer 79, wird ein Zeilenvorschub durchgeführt.

Trennzeichen ";" (Semikolon)

Nach Bearbeitung des vorhergehenden Operanden bleibt der Cursor auf der aktuellen Position stehen.

Nach Abarbeitung des letzten Operanden in der "PRINT" - Anweisung wird ein Zeilenvorschub durchgeführt. Soll das verhindert werden, der Cursor also auf der aktuellen Position stehenbleiben, ist hinter dem letzten Operanden ein ";" anzugeben.

Jedem BA ist im System ein Ausgabepuffer in Größe von 256 Byte zugeordnet.
Ausgaben am Bildschirm erfolgen erst, wenn:

- ein Task-Wechsel durchgeführt wird, also ein gleichzeitig laufendes Programm seine Zeitscheibe erhält oder
- der Ausgabepuffer voll ist oder
- eine "SIGNAL 3" - Anweisung,
- eine "INPUT" - Anweisung,
- eine "CHAIN" - Anweisung,
- eine "LINK" - Anweisung auftritt.

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.

Anmerkung:

Alle Ausgaben erfolgen grundsätzlich ab der aktuellen Cursorposition. Der Cursor wird intern geführt und nur durch die "INPUT" - Anweisung sichtbar gemacht.

Ein-/Ausgabe-Programmierung

7.1.3 Die Anweisung "OPEN #"

Zum Eröffnen und Bearbeiten von Dateien auf beliebigen Kanälen.

Syntax:

<OPEN #> :: = OPEN #<N-EXPR> [{ ;<S-LIT1> / ;<S-VAR1> } , <S-LIT2> / <S-VAR2>]

[[, #<N-EXPR> { ;<S-LIT1> / ;<S-VAR1> }] , { <S-LIT2> / <S-VAR2> }] n 1

Funktion:

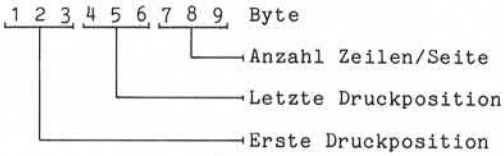
OPEN # = Anweisung.

<N-EXPR> = Angabe der Nummer des Kanals, auf dem die Datei eröffnet werden soll. Standardmäßig stehen 14 Kanäle (0-13) pro Teilnehmer zur Verfügung. Beim Einsatz von TAMOS sind die Kanäle 0 und 1 belegt.

<S-VAR1>
<S-LIT1> = Soll mit dieser Anweisung eine Druckerdatei eröffnet werden, sind hier die Parameter zur Formularsteuerung zu definieren. Der String ist in diesem Fall für mindestens 9 Byte zu dimensionieren.

Ein-/Ausgabe-Programmierung

Aufbau:



Beim Einsatz eines Zeilendruckers werden die erste und die letzte Druckposition nicht ausgewertet.

Die letzte Druckposition muß immer größer sein als die erste Druckposition.

Bei Nadeldruckern mit zwei Vorschub-aggregaten dürfen sich die Druckbereiche nicht überlappen.

Werden keine Angaben zur Formularsteuerung gemacht, gelten folgende Standardwerte:

Zeilendrucker: Erste Druckposition = 000
Letzte Druckposition = 131
Formularhöhe = 048

Nadeldrucker linker Vorschub:
Erste Druckposition = 000
Letzte Druckposition = 176
Formularhöhe = 048

Nadeldrucker rechter Vorschub:
Erste Druckposition = 177
Letzte Druckposition = 177
Formularhöhe = 048

<S-LIT1>
<S-VAR1>

Zum Eröffnen von beliebigen Dateien (außer Druckerdateien) können hier Parameter zur Bearbeitung der jeweils folgenden Datei festgelegt werden.

Ein-/Ausgabe-Programmierung

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 die Fehlermeldung "ERROR # 50".

"R" als erstes Zeichen im String verhindert, daß auf einen gesperrten Satz in der betreffenden Datei zugegriffen werden kann, d.h., greift ein Teilnehmer auf einen gesperrten Satz zu, wird im Programm dieses Teilnehmers "ERROR # 123" gemeldet.

Lochkartenleser:

In den ersten beiden Zeichen im String kann die Anzahl der Spalten (1-80) der zu verarbeitenden Lochkarte angegeben werden.

<S-LIT2> = Name der Datei, die eröffnet werden soll.
<S-VAR2>

Die Angabe der Kanalnummer ist nur für die erste mit einer "OPEN #"-Anweisung zu eröffnenden Datei zwingend erforderlich.

Sollen mehrere Dateien mit einer "OPEN #" - Anweisung auf aufeinanderfolgenden Kanälen eröffnet werden, ist nur die Kanalnummer für die erste Datei erforderlich, z.B.:

```
OPEN #2,"$LPT",A$,B$,C$
```

Die Dateien ohne führende Kanalnummer-Angabe werden in aufeinanderfolgenden Kanälen eröffnet (Kanalnummer der letzten mit dieser Anweisung eröffneten Datei + 1).

Der Nachteil beim Eröffnen von mehreren Dateien mit einer "OPEN #" - Anweisung ist eine unzureichende Fehlerauswertung, d.h., tritt bei der Ausführung einer solchen Anweisung ein Fehler auf, kann nicht festgestellt werden, welche Dateieröffnung den Fehler verursacht hat.

Ein-/Ausgabe-Programmierung

7.1.4 Die Anweisung "READ #"

Lesen von Daten aus Dateien und Übertragen der gelesenen Daten in beliebige Variablen.
Diese Anweisung ist nur für Magnetplattendateien und Lochkartenleser-Dateien zulässig.

Syntax:

<READ #> :: = READ #<N-EXPR1> [,<N-EXPR2> [,<N-EXPR3>]];

{<N-VAR> / <S-VAR>} [, {<N-VAR> / <S-VAR>}]$_{1}^{n}$ [;]

Funktion:

READ = = Anweisung.

<N-EXPR1> = Nummer des Kanals, auf dem die Datei eröffnet ist.

<N-EXPR2> = Satznummer des zu lesenden Satzes relativ zum Dateianfang (Satz # 0).

Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben.

Ist keine Satznummer angegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet, ebenso, wenn <N-EXPR2> = -1 ist.
Ist <N-EXPR2> = -2, wird der Satz gelesen, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde. Textdateien sind davon ausgenommen.

Bei der Bearbeitung von Lochkartenleser-Dateien ist die Angabe <N-EXPR2> bedeutungslos.

<N-EXPR3> = Vorgabe der Anfangsadresse innerhalb eines Datensatzes.
Diese Angabe ist nur dann erlaubt, wenn

<N-EXPR2> codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld # 0) angegeben, ab dem die Datenübertragung beginnen soll.

Bei allen anderen Magnetplattendateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0) anzugeben.

Bei der Bearbeitung von Lochkartenleser-Dateien ist die Angabe <N-EXPR3> ohne Bedeutung.

<N-VAR> oder <S-VAR> = Beliebig viele numerische und/oder String-Variable, Elemente von Vektoren und/oder Matrizen. In diesen Variablen wird der Inhalt des adressierten Datensatzes übertragen. (Bei Textdateien dürfen keine <N-VAR> angegeben werden.) Eine Überprüfung des Formats der Daten, die übertragen werden sollen, erfolgt nur bei formatierten Dateien.
Bei relativen Dateien erfolgt die Übertragung ohne Rücksicht auf Typ/Format der Ziel-Variablen.
Die Anzahl Zeichen, die in die jeweiligen Ziel-Variablen zu übertragen sind, werden in Abhängigkeit von dem Dateityp, gesteuert durch:

- das dimensionierte Format (1-4 Worte) bei numerischen Variablen.
- das Auftreten eines Grenzzeichens oder das Erreichen der dimensionierten bzw. durch Indizes angegebenen Länge bei String-Variablen.
- das Auftreten des "CR" - Codes oder das Erreichen der dimensionierten bzw. durch Indizes angegebenen Länge bei String-Variablen.

Besonderheiten, die bei den verschiedenen Dateitypen zu beachten sind, werden unter Pkt.: 7.3.5.2 bis 7.3.5.5 beschrieben.

= Kennzeichen für Satzsperre.

Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung gelesene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt.
Ist dagegen ein Semikolon codiert, erfolgt keine Satzsperre (siehe auch Pkt.: 7.3.3, File-sharing).

READ ♯ auf Lochkartenleser-Dateien bewirkt keine Satzsperre.

Die Angaben zur Adressierung der zu lesenden Daten (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon ";" von den Namen der "Ziel-Variablen" getrennt sein.

Für Lochkartenleser-Dateien ist als Ziel-Variable nur eine String-Variable zugelassen, da nur alphanumerische Daten (ASCII-Code) angeliefert werden.

Beispiele: 1) Anweisung zum sequentiellen Lesen.
Die Kanalnummer wird in der Variablen "C" vorgegeben. Die Daten werden in die Variablen A, B, C und A$ übertragen.
Nach der Durchführung dieser Anweisung ist der gelesene Datensatz gesperrt.

```
1000 READ #C;A,B,C,A$
```

2) 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 im adressierten Satz.
Der Satz wird nicht gesperrt!

```
1000 READ #C,R,50;A$;
```

Ein-/Ausgabe-Programmierung

7.1.5 Die Anweisung "WRITE #"

Übertragen von Daten aus beliebigen Variablen in Dateien. Diese Anweisung ist für Magnetplattendateien und für Druckerdateien zugelassen.

Syntax:

<WRITE #> :: = WRITE #<N-EXPR1> [,<N-EXPR2> [,<N-EXPR3>]] ;

{<N-EXPR4> / <S-VAR> / <S-LIT>} [, {<N-EXPR4> / <S-VAR> / <S-LIT>}] (1 bis n) [;]

Funktion:

WRITE # = Anweisung.

<N-EXPR1> = Die Nummer des Kanals auf dem die Datei eröffnet ist.

<N-EXPR2> = 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.

Ist keine Satznummer angegeben (N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet. Desgleichen wenn <N-EXPR2> = -1 ist.

Ist <N-EXPR2> = -2, wird der Satz geschrieben, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde. Z.B.: den zuletzt gelesenen und geänderten Satz zurückschreiben.

Für Textdateien gilt diese Konvention nicht.

Bei der Bearbeitung von Druckerdateien ist die Angabe <N-EXPR2> ohne Bedeutung.

<N-EXPR3> = Vorgabe der Anfangsadresse innerhalb eines Datensatzes.
Diese Angabe ist nur dann erlaubt, wenn <N-EXPR2> codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld ≠ 0) angegeben, ab dem die Daten in den adressierten Satz übertragen werden sollen.

Bei allen anderen Magnetplattendateien ist die Adresse des ersten zu übertragenden Bytes, relativ zum Anfang des Datensatzes (Byte # 0) anzugeben.

Bei der Bearbeitung von Druckerdateien ist die Angabe <N-EXPR3> ohne Bedeutung.

<N-EXPR4>
<S-VAR>
<S-LIT> = Beliebige Anzahl von
- Numerischen Ausdrücken,
- String-Variablen und
- String-Literalen.

in beliebiger Reihenfolge.
Der Inhalt dieser Variablen wird in den adressierten Datensatz übertragen.

Bei Drucker- und Textdateien ist die Angabe <N-EXPR> unzulässig.
Eine Überprüfung des Formats der Daten, die übertragen werden sollen, erfolgt nur bei formatierten Dateien. Bei relativen Dateien erfolgt die Übertragung 1:1.

"WRITE #" stellt hinter geschriebenen String-Variablen grundsätzlich ein Grenzzeichen ab, wenn der Quell-String in der dimensionierten Länge ausgegeben wird. Bei Sub-Strings wird kein Grenzzeichen ausgegeben, wogegen dies bei String-Literalen immer der Fall ist.

Die Anzahl Zeichen, die in den Datensatz zu übertragen sind, werden in Abhängigkeit von dem Dateityp gesteuert durch:

- das dimensionierte Format (1-4 Worte) bei numerischen Variablen, 4 Worte bei numerischen Ausdrücken.
- das Auftreten eines Grenzzeichens oder wenn die dimensionierte bzw. durch Indizes angegebene Länge bei String-Variablen erreicht ist.
- die Ausgabe des letzten Zeichens bei String-Literals.

Besonderheiten, die bei den verschiedenen Dateitypen zu beachten sind, werden unter Pkt. 7.3.5.2 bis 7.3.5.5 beschrieben.

; = Kennzeichen für Satzsperre. Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung geschriebene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt, andernfalls erfolgt keine Satzsperre (siehe auch Pkt.: 7.3.3, File-Sharing).

Bei der Bearbeitung von Druckerdateien ist diese Angabe ohne Bedeutung.

Die Angaben zur Adressierung der zu schreibenden Daten (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon ";" von den Namen der zu schreibenden Variablen getrennt sein!

Ein-/Ausgabe-Programmierung

Beispiele:

1) Direktes Schreiben des Inhalts der Variablen "A" in Feld #5 einer formatierten Datei.
Die Kanalnummer wird in der Variablen "C" vorgegeben.
Die Satznummer wird aus dem Inhalt der Variablen "R" + "S" gebildet.
Der Satz wird nicht gesperrt!

```
1000 WRITE #C,R+S,5;A;
```

2) Sequentielles Lesen, Verändern und Zurückschreiben in einer relativen Datei.
Die Kanalnummer wird in der Variablen "C" vorgegeben.
Die Variablen A, B, C und A$ werden bearbeitet.
Nach dem Lesen wird der betreffende Satz gesperrt und erst nach dem Schreiben wieder freigegeben.

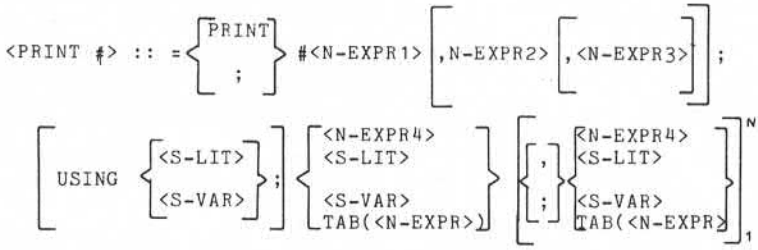
```
1000 READ  #C;A,B,C,A$
1010 GOSUB 2000            /* DATEN VERÄNDERN
1020 WRITE #C,-2;A,B,C,A$;
1030 GOTO 1000
```

Ein-/Ausgabe-Programmierung

7.1.6 Die Anweisung "PRINT #"

Übertragen von Daten aus beliebigen Variablen in Dateien mit gleichzeitiger Umwandlung des Inhalts von numerischen Variablen in alphanumerische (druckbare ASCII) Daten. Diese Anweisung gilt für Magnetplatten- und Druckerdateien.

Syntax:



Funktion:

PRINT =
;
= Anweisung.
Bei der Eingabe der Anweisung kann statt "PRINT" auch ein Semikolon ";" eingegeben werden.

<N-EXPR1> = Die Nummer des Kanals, auf dem die Datei eröffnet ist.
Ist der angegebene Kanal nicht eröffnet, erfolgen sämtliche Ausgaben ohne Fehlermeldung auf dem Bildschirm.

<N-EXPR2> = 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.

Ist keine Satznummer vorgegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet.
Ebenso, wenn <N-EXPR2> = -1 ist.

Ist <N-EXPR2> = -2, wird der Satz

geschrieben, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde (ausgenommen Textdateien).

Bei der Bearbeitung von Druckerdateien ist die Angabe <N-EXPR2> ohne Bedeutung.

<N-EXPR3> = Vorgabe der Anfangsadresse innerhalb eines Datensatzes.
Diese Angabe ist nur dann erlaubt, wenn <N-EXPR2> codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld # 0) angegeben, ab dem die Übertragung der Daten beginnen soll.
Es können in formatierten Dateien nur Felder bearbeitet werden, die nicht numerisch sind.

Bei allen anderen Dateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0) anzugeben. Es ist zu beachten, daß mit "PRINT #" nur ASCII-Code ausgegeben wird, da der Inhalt von numerischen Variablen vor der Ausgabe in ASCII-Code umgewandelt wird (automatisches Editieren).

Bei der Bearbeitung von Druckerdateien ist die Angabe <N-EXPR3> ohne Bedeutung.

USING { <S-LIT> / <S-VAR> } = Mit dieser Angabe wird eine Aufbereitungsmaske vorgegeben, welche die Aufbereitung aller mit dieser Anweisung auszugebenden numerischen Ausdrücke und Variablen steuert.
Die Maske selbst wird in <S-LIT> oder <S-VAR> angegeben.
Die Angaben "USING" darf in einer "PRINT #" - Anweisung nur einmal vorkommen.
Der Aufbau der Masken und die Funktionen der zur Verfügung stehenden Masken-Steuerzeichen ist unter Pkt.: 4.4 beschreiben.

Die "USING"-Angabe muß vom Folgeoperanden unbedingt durch ein Semikolon ";" getrennt sein.

<N-EXPR4> = Beliebige Anzahl von
<S-LIT> - numerischen Ausdrücken
<S-VAR> - String-Variablen
TAB (<N-EXPR>) - String-Literalen
- TAB-Funktionen
in beliebiger Reihenfolge

; = Zeilenvorschub-Steuerung.
Ist im Anschluß an den letzten Operanden der "PRINT #" - Anweisung kein Semikolon ";" codiert, wird zusätzlich zu den Anwenderinformationen ein Zeilenvorschub-Code (215 oktal) ausgegeben. Ist ein Semikolon codiert, entfällt die Ausgabe des zusätzlichen "CR".

Die Ausgabe der Inhalte von numerischen Variablen und numerischen Ausdrücken erfolgt nach ihrer Umwandlung in ASCII-Code. Die Ausgabe von Gleitkommazahlen bzw. BCD-Ganzzahlen, die nicht über Masken aufbereitet werden, erfolgen in der Form:

- Vorzeichen (1 Zeichen), für "+" wird Blank = "␣" und für "-" wird Minus = "-" ausgegeben.
- Im Anschluss an das Vorzeichen werden die signifikanten Ziffern ausgegeben. Führende Nullen werden unterdrückt.
- Im Anschluß an die zuletzt ausgegebene Ziffer wird ein Blank = "␣" ausgegeben.

Enthält die auszugebende Variable den Wert 0, wird "0" ausgegeben.

Bei der Ausgabe von Gleitkommazahlen ist besonders zu beachten, daß ohne Maskenangabe ("USING") die Ausgabe im Gleitkommaformat erfolgt, wenn:

- Die Anzahl der signifikanten Ziffern 14 überschreitet.

- Ein Wert kleiner 0,1 entsteht.

Ein-/Ausgabe-Programmierung

Beispiele: 1) Der Wert 0,01 wird als
1E -2
ausgegeben.

2) Der Wert 999999999999999 wird als
9,9999999999999E +14
ausgegeben.

Das System führt intern (pro Dateikanal) einen "TAB-Zeiger" der die Anzahl ausgegebener Zeichen seit dem letzten:

"CR" - Code (oktal 215)

bzw. seit der Eröffnung der Ausgabedatei beinhaltet. Die Funktion TAB (<N-EXPR>) gibt soviel Blanks aus, wie die Differenz zwischen dem aktuellen TAB-Zeiger und der mit <N-EXPR> angegebenen TAB-Position beträgt.

Achtung! Die angegebene TAB-Position (<N-EXPR>) muß größer als der aktuelle TAB-Zeiger sein. Ist dies nicht der Fall, ist das Ergebnis der Funktion undefiniert.

Der TAB-Zeiger wird auf 0 gesetzt, wenn:

- ein "CR"-Code auftritt.
- die Anweisung komplett abgearbeitet und hinter dem letzten Operanden kein Semikolon codiert ist (dann erzeugt PRINT # "CR"-Code).

Die Angaben zur Adressierung der zu schreibenden Daten (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon ";" von dem Folgeoperanden ("USING" oder die Namen der zu schreibenden Variablen) getrennt sein. Die Namen der Variablen, deren Inhalt ausgegeben werden sollen, müssen durch Komma (",") oder Semikolon (";") voneinander getrennt sein.

Ein-/Ausgabe-Programmierung

Trennzeichen "," (Komma)

Nach Ausgabe des Inhalts der vorhergehenden Variablen werden soviele Blanks ausgegeben, bis die aktuelle TAB-Position ohne Rest durch 15 teilbar ist.

Trennzeichen ";" (Semikolon)

Die Ausgabe des Inhalts der auf ";" folgenden Variablen erfolgt bündig zur letzten Ausgabe.

Die Anweisung "PRINT #" für Magnetplattendateien setzt immer eine Satzsperre für den geschriebenen Datensatz und ein Grenzzeichen hinter das letzte übertragene Zeichen.

Beispiel: 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 ####.

```
PRINT #C USING "####";TAB(10);K$;TAB(70);"SEITE: ";A
```

Im Anschluß daran wird ein "CR"-Code (215 oktal) ausgegeben. Die Ausgabe erfolgt in die Datei, die auf dem in der Variablen C angegebenen Kanal eröffnet ist.

Ein-/Ausgabe-Programmierung

7.1.7 Die Anweisung "MAT READ #"

Lesen von Daten aus Magnetplattendateien und Übertragen der gelesenen Daten in eine:

- String-Variable,
- Numerische Variable,
- Matrix oder einen
- Vektor.

Syntax:

<MAT READ #> ::= MAT READ #<N-EXPR1> [,<N-EXPR2>[,<N-EXPR3>]] ; {<M-VAR> / <S-VAR>} [;]

Funktion:

MAT READ # = Anweisung.

<N-EXPR1> = Nummer des Kanals, auf dem die Datei eröffnet ist.

<N-EXPR2> = Satznummer des zu lesenden Datensatzes relativ zum Dateianfang (Satz 0). Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben. Ist keine Satznummer angegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet. Ebenso wenn <N-EXPR2> = -1 ist.

Ist <N-EXPR2> = -2, wird der Satz gelesen, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde.

<N-EXPR3> = Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0). Die Übertragung erfolgt wortorientiert. Aus diesem Grund sollte in <N-EXPR3> eine geradzahlige Byte-Adresse vorgegeben werden.

Bei einer ungeraden Byte-Adresse beginnt die Übertragung bei der nächsthöheren geradzahligen Adresse.

<M-VAR>
<S-VAR> = Numerische Variable, Matrix, Vektor, oder String-Variable.

Es darf nur ein Variablenname vorgegeben werden! In die codierte Variable wird der Inhalt des adressierten Datensatzes übertragen.
Die Übertragung der Daten erfolgt bis:

- das Ende der aufnehmenden Variablen oder
- das Ende der Datei erreicht ist.

Bei der Bearbeitung eines Vektors oder einer Matrix wird die Variable in ihrer Gesamtheit übertragen, inclusive Reihe und Spalte 0.

; = Kennzeichen für Satzsperre.
Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung gelesene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt.
Ist ein Semikolon codiert, erfolgt keine Satzsperre (siehe auch Pkt.: 7.3.3, File-sharing).

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 (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon von den Namen der Ziel-Variablen getrennt sein.

Beispiele: 1) Lesen der Matrix "A" in ihrer Gesamtheit aus dem Satz, dessen Nummer in der Variablen "R" vorgegeben ist. Die Nummer des Kanals, auf dem die Datei eröffnet ist, steht in der Variablen "C".

```
DIM A(3,3)
.
MAT READ #C,R;A;
```

2) 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 ist, steht in der Variablen "C".

```
DIM A$(512)
.
.
MAT READ #C;A$;
```

7.1.8 Die Anweisung "MAT WRITE #"

Übertragen von Daten aus einer String-Variablen, einer numerischen Variablen, einem Vektor oder einer Matrix in eine Magnetplattendatei.

Syntax:

<MAT WRITE #> ::= MAT WRITE #<N-EXPR1> [,<N-EXPR2> [,<N-EXPR3>]] ;

{<M-VAR> / <S-VAR>} [;]

Funktion:

MAT WRITE # = Anweisung.

<N-EXPR1> = Nummer des Kanals, auf dem die Datei eröffnet ist.

<N-EXPR2> = Satznummer des zu schreibenden Datensatzes relativ zum Dateianfang (Satz # 0).

Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben.

Ist keine Satznummer angegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet, ebenso, wenn <N-EXPR2> = -1 ist.
Ist <N-EXPR2> = -2, wird der Satz geschrieben, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde, z.B.: den zuletzt gelesenen und geänderten Satz zurückschreiben (updating).

<N-EXPR3> = Die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0). Die Übertragung erfolgt wortorientiert. Aus diesem Grund sollte unter <N-EXPR3> eine geradzahlige Byte-Adresse angegeben werden.

Bei einer ungeraden Byte-Adresse beginnt die Übertragung bei der nächsthöheren geradzahligen Adresse.

<M-VAR>
<S-VAR> = Numerische Variable, Matrix, Vektor oder String-Variable.

Es darf nur ein Variablenname angegeben werden. Der Inhalt der codierten Variablen wird ab der mit <N-EXPR2> und <N-EXPR3> definierten Adresse in die Datei übertragen.

Die Übertragung der Daten erfolgt inclusive eventueller Grenzzeichen bis:

- die Variable in ihrer dimensionierten Länge ausgegeben ist.
- das Dateiende erreicht ist (nur bei relativen Dateien).
- nicht genügend Plattenkapazität verfügbar ist.

Ist <S-VAR> angegeben, werden Grenzzeichen mit übertragen.

Bei der Bearbeitung eines Vektors oder einer Matrix wird auch das Element 0 bzw. Reihe und Spalte 0 übertragen.
Es können mehrere Sätze mit einer Anweisung geschrieben werden.

; = Kennzeichen für Satzsperre.
Wird hinter dem letzten Operanden kein Semikolon codiert, ist der mit dieser Anweisung geschriebene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt. Ist ein Semikolon codiert, erfolgt keine Satzsperre (siehe auch Pkt.: 7.3.3, File-Sharing).
Die Satzsperre wird nur für den jeweils adressierten Satz wirksam, auch dann, wenn mit einer Anweisung mehrere Datensätze geschrieben werden.

Ein-/Ausgabe-Programmierung

Beispiele: 1) 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" vorgegeben.

```
.
DIM A$(512)
.
.
MAT WRITE #C,-2;A$;
```

2) Schreiben der Matrix "A" in ihrer Gesamtheit ab der in der Variablen "X" angegebenen Byte-Nummer relativ zu Byte # 0 in Satz 0. Die Nummer des Kanals, auf dem die Datei eröffnet ist, ist in der Variablen "C" vorgegeben.

```
.
DIM A(3,3)
.
.
MAT WRITE #C,0,X;A;
```

7

Ein-/Ausgabe-Programmierung

7.1.9 Die Anweisung "CLOSE #"

Schließen von eröffneten Dateien.

Syntax:

<CLOSE #> ::= CLOSE # <N-EXPR> [,#<N-EXPR>]$_{1}^{n}$

Funktion:

CLOSE # = Anweisung.

<N-EXPR> = Die Nummer des Kanals, auf dem die zu schließende Datei eröffnet ist. Mit einer "CLOSE #" - Anweisung können beliebig viele Dateien geschlossen werden. Ist unter der angegebenen Kanalnummer keine Datei eröffnet, wird Basic-Fehler # 49 gemeldet.

Der Nachteil beim Schließen mehrerer Dateien (Kanäle) mit einer "CLOSE #" - Anweisung ist eine unzureichende Fehlerauswertung, d.h. tritt bei der Anweisung ein Fehler auf, kann nicht festgestellt werden, welche Datei den Fehler verursacht hat.

Beispiel: Routine zum Schließen der Kanäle 2 bis 13. Die von TAMOS belegten Kanäle 0 und 1 werden nicht geschlossen.

```
.
1000 FOR I=2 TO 13
1010 IF ERR 0 GOTO..... /* FEHLERAUSW. 49
1020 CLOSE #I
1030 IF ERR 0 GOSUB..... /* STANDARD
                            FEHLERBEHANDL.
1040 NEXT I
```

Ein-/Ausgabe-Programmierung

7.2 Der Bildschirm-Arbeitsplatz (BA)

Der BA ist ein rechnergesteuerter Bildschirm-Tastatur-Arbeitsplatz, der über Leitungsanschluß mit der Zentraleinheit des Systems 8870/1 verbunden ist.

Der BA übernimmt folgende Funktionen:

- An- und Abmelden von Programmen/Tasks
- Dialog zwischen Bediener und Programm
- Daten Ein- und Ausgabe.

Jeder am System angeschlossene BA repräsentiert einen autonomen Teilnehmer.
Die Bildschirm-Arbeitsplätze unterliegen nicht dem Kanalkonzept. Jeder BA ist immer dem Programm zugeordnet, das er aufgerufen und gestartet hat.

7.2.1 Aufbau von Bildschirm und Tastatur

Auf dem Bildschirm können 2000 Zeichen in 25 Zeilen zu je 80 Spalten dargestellt werden.
Die Anzeige der Zeichen kann in zwei Intensitätsstufen erfolgen:

- Vordergrund = volle Leuchtkraft
- Hintergrund = verminderte Leuchtkraft.

Die Tastatur ist aufgeteilt in:

- die alphanumerische Tastatur und
- die numerische Tastatur.

Außer Tasten, die der Eingabe von Datenzeichen dienen, enthält sie eine Reihe von Funktionstasten, die zur

- Unterstützung der Eingabe dienen und/oder
- Systemfunktionen auslösen.

7.2.2 Die Funktion des BA

Zur Unterstützung von Ein-/Ausgaben am BA stellt Basic 15 Funktionen zur Verfügung, die symbolisch in den Anweisungen:

- PRINT und
- INPUT

angegeben werden.

Diese Funktionen sind:

TAB (X,Y)	=	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 Keybord Bell
'TB'	=	Tab to next Foregroundfield
'CFF'	=	Clear Foregroundfield
'CH'	=	Cursor home

und können auch oktal codiert in alphanumerischen Konstanten und String-Variablen auftreten.

Bei der oktalen Codierung ist unbedingt zu beachten, daß vor den meisten Funktionscodes ein "Lead-in" Code (376 oktal) angegeben werden muß. Dieser Code zeigt dem Steuerprogramm im BA an, daß das folgende Zeichen ein Funktionscode ist (siehe auch Pkt. 7.2.2.1 bis 7.2.2.15).

Anmerkung: Bei oktaler Angabe der Funktion TAB, müssen die X,Y - Koordination ebenfalls oktal codiert vorgegeben werden (siehe Pkt. 7.2.2.1).

7.2.2.1 Tabulation

Die Bildschirm-Funktion "TAB" ermöglicht die beliebige Cursorpositionierung auf dem Bildschirm.

Die Funktion ist folgendermaßen aufgebaut:

TAB (X)

"(X)" repräsentiert einen beliebigen numerischen Ausdruck. Der Cursor wird auf die mit "(X)" bezeichnete Spalte der aktuellen Zeile positioniert. Die neue Cursorposition muß gleich oder größer als die aktuelle Position sein. Andernfalls ist das Ergebnis undefiniert.

TAB (X,Y)

"(X,Y)" repräsentieren beliebige numerische Ausdrücke. Der Cursor wird auf die mit "X" bezeichnete Spalte in der mit "Y" bezeichneten Zeile positioniert.

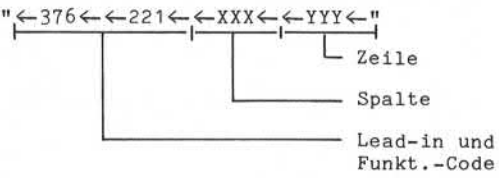
Anmerkung: Die Spaltenangabe muß im Bereich von 0 bis 79, die Zeilenangabe im Bereich von 0 bis 24 liegen. Andernfalls ist das Ergebnis der Funktion undefiniert. Aus syntaktischen Gründen müssen die Koordinaten in Klammern angegeben werden.

Beispiel: Positionierung des Cursor auf Spalte 10 in der Zeile 2.

```
PRINT TAB(10,2);
```

Funktions-Symbol: TAB

Oktal-Code



Ein-/Ausgabe-Programmierung

7.2.2.2 Backspace Cursor

Positionierung des Cursors ab der aktuellen Position um eine Spalte nach links. Ein auf der neuen Position stehendes Zeichen wird gelöscht.
Steht der Cursor auf Spalte 0 einer beliebigen Zeile, erfolgt eine Positionierung auf die Spalte 79 der darüberliegenden Zeile.
Steht der Cursor auf Spalte 0 der Zeile 0 bleibt die Funktion ohne Auswirkung.

Der Wechsel aus einem Vordergrundfeld in ein Hintergrundfeld ist nicht möglich. Steht der Cursor unmittelbar hinter einem Hintergrundfeld, bleibt die Funktion ohne Auswirkung. Der Cursor bleibt also auf der aktuellen Position und der akustische Signalgeber wird gesetzt.

Funktions-Symbol : 'BS'

Oktal-Code : "←210←"

Beispiel : PRINT 'BS';

7.2.2.3 Mark Cursorposition

Die aktuelle Position des Cursors (Spalte und Zeile) wird gespeichert. Die gespeicherten Koordinaten werden nur durch erneutes Aufrufen dieser Funktion geändert, oder durch Ausschalten des BA gelöscht.

Funktions-Symbol : 'MP'

Oktal-Code : "←376←←211←"

Beispiel : PRINT 'MP';

7.2.2.4 Back to marked Cursorposition

Der Cursor wird auf die Bildschirmposition gesetzt, die beim letzten Aufruf der Funktion 'MP' (Mark Cursorposition) gespeichert wurde.
Nach Neueinschalten des BA wird beim Aufruf dieser Funktion der Cursor auf Spalte 0 in Zeile 0 gesetzt.

Funktions-Symbol : 'BP'

Oktal-Code : "←376←←212←"

Beispiel : Eingabe und Positionierung auf die gemerkte Bildschirmposition.

```
INPUT 'BP' X
```

7.2.2.5 Carriage Return

Positionieren des Cursors auf Spalte 0 der nächsten Zeile (Zeilenschaltung). Der Bildschirminhalt wird nicht verändert. Befindet sich der Cursor 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. Anschließend steht der Cursor auf Spalte 0 in Zeile 24.

Funktions-Symbol : 'CR'

Oktal-Code : "←215←"

Beispiel : Ausgabe der alphanumerischen Konstanten "FELD 1" und "FELD 2" in den Spalten 0 der Zeilen 5 und 6.

```
PRINT TAB(0,5);"FELD 1";'CR';"FELD 2";
```

Ein-/Ausgabe-Programmierung

7.2.2.6 Line Deletion

Alle Zeilen unterhalb der Zeile, in der der Cursor steht, werden um eine Zeile nach oben geschoben.
Die Zeile in der der Cursor steht wird überschrieben und Zeile 24 wird frei. Steht der Cursor in Zeile 24, wird nur diese eine Zeile gelöscht.

Funktions-Symbol : 'LD'

Oktal-Code : "←376←←223←"

Beispiel : Normieren der Statuszeile (Zeile 24) durch Löschen und die Ausgabe des Textes "STATUS" ab der Spalte 0.

```
PRINT TAB(0,24);'LD';"STATUS :";
```

7.2.2.7 Line Insertion

Einfügen einer Leerzeile (Vordergrund) ab der aktuellen Cursorposition.
Die Zeile, in der der Cursor steht, sowie alle darunterliegenden werden um eine Zeile nach unten geschoben. Die Informationen in Zeile 24 gehen verloren, und der Cursor steht anschließend auf Spalte 0 der eingefügten Zeile.

Funktions-Symbol : 'LI'

Oktal-Code : "←376←←232←"

Beispiel : Verschieben des gesamten Bildschirminhaltes um eine Zeile nach unten.

```
PRINT TAB(0,0);'LI';
```

7.2.2.8 Start Background

Alle folgenden Zeichen erscheinen auf dem Bildschirm als Hintergrund-Zeichen mit verminderter Leuchtkraft.

Funktions-Symbol : 'SB'

Oktal-Code : "←376←←231←"

Beispiel : PRINT 'SB';

Ein-/Ausgabe-Programmierung

7.2.2.9 Start Foreground

Alle folgenden Zeichen erscheinen auf dem Bildschirm als Vordergrund-Zeichen mit voller Leuchtkraft. Diese Funktion ist nur dann notwendig, wenn zuvor "Start Background" abgesetzt wurde, weil standardmäßig immer Vordergrundzeichen erscheinen.

Funktions-Symbol : 'SF'

Oktal-Code : "←376←←237←"

Beispiel : Nach dem Text "RABATT" (erscheint im Hintergrund) Umschalten in den Vordergrund und Eingabe in "A".

```
INPUT 'SB',"RABATT : ",'SF'A
```

7.2.2.10 Clear Screen

Der gesamte Bildschirm wird gelöscht. Der Cursor steht anschließend auf Spalte 0 in Zeile 0.

Funktions-Symbol : 'CS'

Oktal-Code : "←376←←234←"

Beispiel : PRINT 'CS';

7.2.2.11 Clear Foreground

Alle Vordergrundfelder werden gelöscht, und der Cursor steht anschließend auf dem ersten Zeichen des ersten Vordergrundfeldes. Ist kein Vordergrundfeld vorhanden, steht der Cursor auf Spalte 79 in Zeile 24.

Funktions-Symbol : 'CF'

Oktal-Code : "←376←←235←"

Beispiel : PRINT 'CF';

7.2.2.12 Ring Keybord Bell

Starten des akustischen Signalgebers der Tastatur. Die Funktion hat keinen Einfluß auf den Bildschirminhalt. Das Signal wird nach kurzer Zeit hardwaremäßig wieder abgeschaltet.

Funktions-Symbol : 'BEL'

Oktal-Code : "←207←"

Beispiel : Ausgabe einer String-Variablen ab Spalte 25 in Zeile 24 und setzen des akustischen Signalgebers.

```
PRINT TAB(25,24);A$;'BEL';
```

7.2.2.13 Tab to next Foregroundfield

Cursorpositionierung auf das erste Zeichen des folgenden Vordergrundfeldes.
Folgt kein Vordergrundfeld mehr, bleibt der Cursor auf der aktuellen Position stehen.

Funktions-Symbol : 'TB'

Oktal-Code : "←211←"

Beispiel : Tabulation auf das nächste Vordergrundfeld und Eingabe in das n-te Element eines Vektors.

```
INPUT 'TB',X(N)
```

Ein-/Ausgabe-Programmierung

7.2.2.14 Clear Foregroundfield

Löschen eines Vordergrundfeldes.
Diese Funktion löscht das Vordergrundfeld in dem sich der Cursor befindet. Nachdem die Funktion ausgeführt ist, steht der Cursor am Zeilenanfang der betreffenden Zeile oder auf der ersten Position des entsprechenden Vordergrundfeldes.

Funktionssymbol : 'CFF'

Oktal-Code : "←230←"

7.2.2.15 Cursor home

Mit dieser Funktion wird der Cursor, per Programm, in der Zeile und Spalte "0" positioniert, sofern diese ein Vordergrundfeld ist. Gehört diese Position zu einem Hintergrundfeld, erfolgt die Positionierung auf die erste Stelle des ersten Vordergrundfeldes.

Folgt kein Vordergrundfeld, bleibt der Cursor auf der aktuellen Position.

Funktions-Symbol : 'CH'

Oktal-Code : "←376←←222←"

Ein-/Ausgabe-Programmierung

7.2.3 Übersicht der verfügbaren Funktionen

Symbol	Funktion	Lead-in	Oktal-Code
TAB (X,Y)	Cursorpositionierung X=Spalte / Y=Zeile	X	221 + X,Y Koordinaten
'BS'	Cursor um eine Stelle nach links setzen.	-	210
'MP'	Speichern der aktuellen Cursorposition	X	211
'BP'	Cursor auf die gesp. Position stellen.	X	212
'CR'	Zeilenschaltung	-	215
'LD'	Eine Zeile löschen	X	223
'LI'	Eine Zeile einfügen	X	232
'SB'	Start Hintergrundausgabe	X	231
'SF'	Start Vordergrundausgabe	X	237
'CS'	Bildschirm löschen	X	234
'CF'	Vordergrundfelder löschen	X	235
'BEL'	Akustisches Signal ausgeben	-	207
'TB'	Tabulation auf das nächste Vordergrundfeld	-	211
'CFF'	Löschen des aktuellen Vordergrundfeldes		230
'CH'	Cursor in Zeile 0 und Spalte 0	X	222

Ein-/Ausgabe-Programmierung

7.2.4 Fehlermeldungen des BA

Vom BA-Steuerprogramm ist eine Fehlerzeile (Zeile 24) eingerichtet, die bei Bedarf sichtbar gemacht werden kann.
Sie hat folgenden Aufbau:

ERROR: XX		Zeile 24
0 Bereich für Fehlermeldungen	47 Freier Bereich 79	Spalte

Erkennt das Steuerprogramm einen Fehler, wird das dem Bediener durch die rote Lampe an der Tastatur angezeigt. Durch Drücken der Taste "ERR" kann sich der Bediener die Fehlernummer in Zeile 24 anzeigen lassen. Die Anwenderinformation der Zeile 24 wird zwischengespeichert.
Nach erneuter Betätigung der Fehlerschlüsseltaste erscheint wieder die Anwenderinformation in der Zeile 24, und die rote Lampe erlischt.

Anmerkung: Die Fehlernummer sollte immer nur für kurze Zeit angezeigt werden, da die Ausführungszeit des BA beeinflußt wird!

Der einzige Fehler, der durch das Anwenderprogramm verursacht werden kann, ist

ERROR: 01

Dieser Fehler bedeutet, daß vom BA ein "Lead-in" - Code ohne folgenden Funktionscode empfangen wurde.

Eine Übersicht aller möglichen Fehler ist im Anhang "Fehlermeldungen" vorhanden.

Ein-/Ausgabe-Programmierung

7.3 Die Magnetplatte

Alle auf einer Magnetplatte gespeicherten Daten sind in Dateien organisiert.
Neben Programmdateien für Anwenderprogramme und Datendateien für Anwenderdaten befinden sich auf der Systemplatte (LU-Nummer 0) die Dateien zur Aufnahme des Betriebssystems (NIROS) und der zur Systemverwaltung und Systemsteuerung erforderlichen Daten.

7.3.1 Aufbau von Magnetplattendateien

Jede Plattendatei besteht aus

- einem oder mehreren Blöcken (Sektoren),
- einem Datei-Kennsatzblock (Header) und
- 0 bis n Datenblöcken.

Bei formatierten Dateien und bei Textdateien können zusätzlich zum Datei-Kennsatz noch Kennsatz-Erweiterungsblöcke vorhanden sein. Anwenderdaten sind in jeder von Basic unterstützten Darstellungsart:

- ASCII-Code,
- Integer und
- Gleitkomma-Zahl

in Datendateien mit Ausnahme von Text-Dateien abzustellen. Textdateien erlauben nur Darstellung in ASCII-Code.

Ein-/Ausgabe-Programmierung

7.3.1.1 Formatierte Dateien

Eine formatierte Datei besteht aus einem Datei-Kennsatz und der entsprechenden Anzahl Datenblöcke, die erforderlich sind, alle aktuellen Datensätze aufzunehmen.
Sie belegt auf der Platte keinen zusammenhängenden Bereich. Erweiterungen werden, wenn sie erforderlich sind, über eine Liste der freien Plattenblöcke vorgenommen.

Bei Datensätzen mit Satznummern >128*(256/Satzl.), werden zusätzlich zum Dateikennsatz 1 bis N , maximal jedoch 128 , Kennsatzerweiterungsblöcke angelegt.
Ein Kennsatzerweiterungsblock ermöglicht die Verwaltung von max. 256 Datenblöcken.

Alle Datensätze einer formatierten Datei haben die gleiche Struktur hinsichtlich Anordnung und Art der einzelnen Datenfelder. Ein Datensatz kann bis zu 64 verschiedene Datenfelder aufnehmen. Die Formatbeschreibung wird im Datei-Kennsatz abgelegt.

Die maximale Anzahl von Datensätzen für eine formatierte Datei ist abhängig von der Satzlänge.
Die Berechnung der maximalen Anzahl Datensätze erfolgt mit der Formel:

```
128 * 256 * INT (256 / Satzlänge in Worten)
```

128	= Maximale Anzahl von Kennsatz-Erweiterungsblöcken.
256	= Maximale Anzahl von Verweisen auf Datenblöcke pro Kennsatz-Erweiterungsblock
INT(256/ Satzlänge)	= Anzahl Datensätze pro Datenblocklänge in Worten.9

Minimale Satzlänge: 1 Wort
Maximale Satzlänge: 256 Worte

Formatierte Dateien ermöglichen sowohl sequentiellen als auch direkten Zugriff auf die Datensätze.
Ein direkter Zugriff erfolgt durch die Vorgabe der Satznummer relativ zum Dateianfang.
Durch die Vorgabe der Feldnummer relativ zum Satzanfang (Feld 0) kann gezielt auf einzelne Felder innerhalb eines Datensatzes zugegriffen werden.

7.3.1.2 Relative Dateien

Eine relative Datei besteht aus dem Datei-Kennsatz und der entsprechenden Anzahl Datenblöcken, die erforderlich sind, die maximal erwartete Anzahl von Datensätzen aufzunehmen.

Eine relative Datei belegt immer einen zusammenhängenden Bereich auf der Platte. Die Größe dieses Bereiches wird durch die maximal erwartete Satzanzahl und die Satzlänge bestimmt.

Berechnung des Platzbedarfes:

$$\frac{\text{Satzlänge in Worten x Anzahl Datensätze}}{256} + 1$$

Das aufgerundete Ergebnis ist die Anzahl Plattenblöcke, die zur Aufnahme der Datei erforderlich sind. Diese Plattenblöcke müssen einen zusammenhängenden Bereich bilden.

Die maximale Anzahl von Datensätzen ist bei relativen Dateien auf 1000000 begrenzt.

Minimale Satzlänge: 1 Wort
Maximale Satzlänge: 1000000 Worte (theoretischer Wert)

Relative Dateien ermöglichen sowohl sequentiellen als auch direkten Zugriff auf die Datensätze.

Ein-/Ausgabe-Programmierung

7.3.1.3 Index-Dateien

Eine Index-Datei besteht aus dem Datei-Kennsatz, 1 bis 15 Schlüsselverzeichnissen und dem Bereich zur Aufnahme der Datensätze.

Eine Indexdatei belegt immer einen zusammenhängenden Bereich auf einer Magnetplatte. Die Größe des Bereiches wird durch die maximal erwartete Anzahl Sätze, die Satzlänge und die Länge der Ordnungsbegriffe bestimmt.

Berechnung des Plattenbedarfes:

- SB = 254 / (L + 1)
- DB = AD * SL / 256
- GF = AD * 2 / SB
- GC = GF / SB
- GM = GC / SB
- GS = GF + GC + GM
- GI = GS + DB

Abkürzungen:

DB = Größe des Datenbereichs

SL = Satzlänge

SB = Anzahl Schlüssel pro Block

L = Schlüssellänge in Worten

GF = Größe der Fine Ebene in Blöcken

AD = Anzahl der Datenblöcke

GC = Größe dr Coarse Ebene in Blöcken

GM = Größe der Master Ebene in Blöcken

GI = Größe der Index Datei in Blöcken

GS = Größe des Verzeichnisses in Blöcken

Ein-/Ausgabe-Programmierung

Ist

GI * 256 / SL > 65534

muß die Berechnung wiederholt werden. Bei GF muß der Wert für SB lauten:

SB = 254 / (L + 2)

DB, GF, GC und GM sind jeweils aufzurunden, SB ist abzurunden. Der Gesamtbedarf an Plattenblöcken für eine Index Datei Beträgt

GI + 1. Die Größe einer Index-Datei bzw. die maximale Anzahl der Datensätze, die eine Index-Datei aufnehmen kann, ist von der Länge der Ordnungsbegriffe, der Satzlänge und der Anzahl Verzeichnisse abhängig.

Minimale Satzlänge:	1 Wort
Maximale Satzlänge:	1000000 Worte (theoretischer Wert)
Minimale OB-Länge:	1 Wort
Maximale OB-Länge:	15 Worte

Index-Dateien ermöglichen sowohl sequentiellen als auch direkten Zugriff auf die Datensätze.

Dabei bestehen die folgenden Möglichkeiten:

- Sequentielles Lesen des Datenbereiches.
- Sequentielles Lesen des Indexbereiches und direkter Zugriff auf die Datensätze.
- Direkter Zugriff im Indexbereich durch Vorgabe eines Ordnungsbegriffs und direkter Zugriff auf den zugehörigen Datensatz.

7

Ein-/Ausgabe-Programmierung

7.3.1.4 Text-Dateien

Eine Text-Datei besteht aus dem Datei-Kennsatz und der entsprechenden Anzahl Datenblöcken, die erforderlich sind, um die aktuelle Datenmenge aufzunehmen.
Belegen die Daten mehr als 128 Plattenblöcke, sind zusätzlich zum Datei-Kennsatz 1 bis n Erweiterungsblöcke vorhanden. Ein Kennsatz-Erweiterungsblock ermöglicht die Verwaltung von 256 Datenblöcken.

Eine Text-Datei belegt keinen zusammenhängenden Bereich auf der Magnetplatte. Erweiterungen werden über eine Liste der freien Plattenblöcke vorgenommen.
Der Inhalt einer Text-Datei besteht aus einer einzigen Zeichenkette, deren maximale Länge theoretisch

16777216 Byte

beträgt.

In diesem Fall werden 128 Kennsatz-Erweiterungsblöcke angelegt.

Diese Zeichenkette wird in Seiten und Zeilen aufgeteilt. Zeilen werden durch "Return" - Codes (215 oktal) und Seiten durch "Top of Form" - Codes (214 oktal) voneinander getrennt.
Das Ende der Zeichenkette, zugleich auch das Ende der Text-Datei, wird durch ein "NULL"- Byte angezeigt.

Text-Dateien ermöglichen sowohl sequentiellen als auch direkten Zugriff.
Ein direkter Zugriff auf bestimmte Zeilen bzw. Seiten ist nicht möglich. Es kann nur auf ganze Plattenblöcke zugegriffen werden, was durch die Vorgabe der Blocknummer relativ zum Dateianfang erfolgt.
Durch die Angabe einer Byte-Adresse relativ zum Anfang des adressierten Blockes kann gezielt auf Daten in einem Block zugegriffen werden.

Ein-/Ausgabe-Programmierung

7.3.2 Die Zugriffsmöglichkeiten

Alle Organisationsformen von Magnetplattendateien ermöglichen sowohl sequentiellen als auch direkten Zugriff.

Übersicht über die Zugriffsmöglichkeiten bei den verschiedenen Organisationsformen und Anweisungen:

	Organisation			
Anweisung	Formatiert	Relativ	Index	Text
MAT READ #	R,S	R,S	R,S	R,S
MAT WRITE #	R,S	R,S	R,S	R,S
PRINT #	R,S	R,S	R,S	S
READ #	R,S	R,S	R,S	R,S
SEARCH #	-	-	R,S	-
WRITE #	R,S	R,S	R,S	R,S

R = Random (direkt, wahlfrei)
S = Sequentiell

Anmerkung: Die Anweisung "SEARCH #" dient nur zum Zugriff auf die Indexbereiche von Indexdateien, nicht zum Zugriff auf einzelne Datensätze.

Ein-/Ausgabe-Programmierung

7.3.3 File-Sharing

NIROS erlaubt das Bearbeiten von Magnetplatten-Dateien durch mehrere Teilnehmer, auch wenn gleichzeitig auf denselben Datensatz zugegriffen wird.
Dies erfordert das Vorhandensein von Schutzfunktionen gegen:

- Löschen von Dateien, während mehrere Teilnehmer diese benutzen.
- Unkontrolliertes, gleichzeitiges Verändern (updaten) von Datensätzen und damit verbundene Datenverfälschung.

Der Schutz gegen Löschen von Dateien, während andere Teilnehmer darauf zugreifen, ist unter Punkt 7.3.6 (Löschen von Dateien) beschrieben.

Der Schutz vor unkontrolliertem, gleichzeitigen Verändern ist durch die Möglichkeit des gezielten Sperrens von Datensätzen gegeben.

Sperren von Datensätzen

Ein Datensatz wird gesperrt, wenn eine der Anweisungen

- MAT READ #
- MAT WRITE #
- READ #
- WRITE #

ausgeführt wird, und nach dem letzten Operanden kein ";" folgt.
Die Anweisung "PRINT #" setzt grundsätzlich Satzsperre auf!

Die Satzsperre ist wirksam:

- Bei sequentiellem Zugriff für den aktuellen Datensatz, der mit dieser Anweisung gelesen wird.

- Bei direktem Zugriff für den Satz, dessen relative Satznummer in der Anweisung angegeben ist.

Werden mit einer der Anweisungen "MAT READ #", "MAT WRITE #", READ # oder WRITE #,mehrere Datensätze gleichzeitig gelesen oder geschrieben, ist die Satzsperre nur für den ersten gelesenen/geschriebenen Satz wirksam.
Ist in einer Anweisung ein Displacement angegeben, das größer als die Satzlänge des adressierten Satzes ist,

also Daten in einem Folgesatz adressiert, gilt die Sperre immer nur für den aktuellen Datensatz, nicht für den, der über das Displacement adressiert wird.
Pro Datenkanal kann jeweils nur ein Satz gesperrt werden! Sollen in einer Datei gleichzeitig mehrere Sätze gesperrt werden, muß die Datei auf mehreren Kanälen gleichzeitig eröffnet und pro Kanal ein Satz gesperrt werden.
In Text-Dateien können keine Sätze gesperrt werden!

Wirkung der Satzsperre

Ein gesperrter Datensatz kann von anderen Teilnehmern nicht bearbeitet (gelesen oder geschrieben) werden. Die Adressierung eines gesperrten Satzes bewirkt einen Task-Wechsel. Bei jeder Aktivierung des Programmes wird versucht den Satz zu lesen. Er wird erst dann zur Verfügung gestellt, wenn er vom sperrenden Programm wieder freigegeben ist. Es erfolgt keine Statusmeldung.

Aufheben der Satzsperre

Ein Aufheben der Satzsperre kann nur durch das Programm erfolgen, das den Satz gesperrt hat.
Dies ist der Fall, wenn:

- der gesperrte Satz auf dem Kanal, auf dem er gesperrt wurde, erneut gelesen oder geschrieben wird und in der ausführenden Anweisung:
  - MAT READ # - MAT WRITE # - READ # - WRITE #
  auf den letzten Operanden ein ";" folgt.

- ein anderer Datensatz auf dem gleichen Kanal gelesen oder geschrieben wird.

- der Kanal auf dem gesperrt wurde, geschlossen wird.

- das Programm durch eine der Bedingungen:
  - Auftreten einer Anweisung "END"

  - Auftreten einer Anweisung "CHAIN" die einen Processor aufruft (z.B. CHAIN "RUN")

  - durch [ESC] und [CTL][C] oder [CTL][Y] , [ESC] und [CTL][C]

beendet wird.

Ein-/Ausgabe-Programmierung

Dead lock

Beim Arbeiten mit Satzsperre ist es möglich, daß sich zwei oder gar mehrere Programme gegenseitig blockieren. Bei der Organisation und auch bei der Programmierung sollte darauf geachtet werden, daß keine Konstellation von Zugriffen entsteht, die einen "dead lock" verursachen kann.

Beispiel: Programm A und Programm B arbeiten beide mit den Dateien X und Y.
Beide Programme sperren die Datensätze beim Lesen und geben sie beim Zurückschreiben, bzw. beim Lesen des nächsten Satzes wieder frei.

Ablauf:

Zeitscheibe	Programm	
1	A	Lesen und sperren von Satz # 10 in der Datei X.
2	B	Lesen und sperren von Satz # 100 in der Datei Y.
3	A	Auswertung der gelesenen Daten und Lesen von Satz # 100 in der Datei Y.
4	B	Auswerten der gelesenen Daten und Lesen von Satz # 10 in der Datei X.

Beide Programme versuchen, den vom jeweils anderen Programm gesperrten Datensatz zu lesen. In diesem Fall muß zumindest eines der beiden Programme abgebrochen werden, um dem anderen das Weiterarbeiten zu ermöglichen.

7.3.4 Erstellen von Dateien

Zum Erstellen neuer Dateien bzw. zum Ersetzen von bereits vorhandenen Dateien stellt Business Basic die Anweisung

BUILD #

zur Verfügung. BUILD # erstellt neue Dateien bzw. ersetzt vorhandene Dateien. Es wird ein Bereich auf der Magnetplatte reserviert, der Dateiname ins Inhaltsverzeichnis (Datei INDEX) eingetragen und der Datei-Kennsatz angelegt.
Die benötigten Plattenblöcke werden in der Plattenbelegungsliste als belegt eingetragen.

Aufbau:

<BUILD #> ::= BUILD #<N-EXPR>, [+] { <S-VAR> / <S-LIT> }

[[,#<N-EXPR>] , [+] { <S-VAR> / <S-LIT> }] n 1

Funktion:

BUILD #	=	Anweisung.
<N-EXPR>	=	Die Nummer des Kanals, auf dem die Datei erstellt werden soll.
+	=	"+" kennzeichnet die zu erstellende Datei als Text-Datei.
<S-VAR> oder <S-LIT>	=	Bezeichnet die zu erstellende Datei. Die gesamten Angaben können in einer String-Variablen (<S-VAR>) oder als String-Literal (<S-LIT>) in der Anweisung übergeben werden.

Ein-/Ausgabe-Programmierung

<S-VAR> und <S-LIT> haben folgenden Aufbau:

[<PP>] [COST] [(SA:SL)] [LU/] DATEINAME [!]

<PP> = Schutzstufen-Zuordnung (Protection). Die beiden spitzen Klammern müssen codiert werden ! Für Text-Dateien ist diese Angabe nicht zulässig.

<PP>
- Benutzer derselben Privilegebene
- Benutzer niedriger Privilegebenen

Ein Schutz gegen Benutzer höherer Privilegebenen besteht nicht.

Es gelten folgende Schutzkennzeichen: stehen:

P = 0 = kein Schutz

1 = Kopierschutz

2 = Schreibschutz

3 = Kopier- und Schreibschutz

4 = Leseschutz

5 = Lese- und Kopierschutz

6 = Lese- und Schreibschutz

7 = Lese, Schreib- und Kopierschutz

Ist keine Schutzstufe angegeben, wird automatisch "77" eingetragen.

Ein-/Ausgabe-Programmierung

COST = Der Betrag, der dem Konto eines anderen Benutzers für den Zugriff auf diese Datei berechnet wird.
Für Text-Dateien ist diese Angabe nicht zulässig.

Für die Kostenangabe gilt die Form:

$DDD.PP

$ muß codiert sein.
DDD - Vorkommabetrag
PP - Nachkommabetrag

Der eingegebene Nachkomma-Betrag wird immer auf einen vollen 10er-Wert abgerundet.

Ist "COST" nicht angegeben, wird "0" eingetragen.

(SA:SL) = Anzahl Datensätze und Satzlänge

SA = Anzahl Datensätze, für die ein Plattenbereich reserviert werden soll.

SL = Satzlänge in Worten
(1 Wort = 2 Byte)

Diese beiden Angaben stehen in Klammern und sind durch ":" voneinander getrennt Ist diese Angabe in einer "BUILD #"-Anweisung vorhanden, wird Platz für eine relative Datei reserviert.

LU/ = LU-Nummer der Platte, auf der die Datei erstellt werden soll.
Die Lu-Nr. ist durch "/" vom Dateinamen zu trennen.
Für die log. Einheit 0 kann diese Angabe entfallen.

DATEINAME = Der Name, unter dem die Datei auf der Platte erstellt werden soll, bzw. der Name, der Datei die ersetzt werden soll

! = Muß dann angegeben werden, wenn eine bereits bestehende Datei ersetzt werden soll.

Ein-/Ausgabe-Programmierung

Anmerkung:
Soll eine Datei ersetzt werden, ist zu beachten, daß diese erst dann gelöscht wird, wenn die neue Datei (die sie ersetzt) vorher angelegt wurde.

"BUILD #" ist eine Anweisung zum Erstellen mehrerer Dateien.

Wird diese Möglichkeit genutzt 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 (Kanalnummer der letzten mit dieser Anweisung erstellten Datei + 1).

- Mit "BUILD #" erstellte Dateien sind gleichzeitig für die Verarbeitung eröffnet.
- Mit "BUILD #" erstellte Dateien sind, bevor sie durch das erstellende Programm mit der Anweisung "CLOSE #" geschlossen werden, vor dem Zugriff anderer Teilnehmer geschützt.
- Eine mit "BUILD #" angelegte Datei wird gelöscht, wenn das Anwenderprogramm beendet wird und der Kanal auf dem die Datei erstellt wurde, nicht vorher durch die Anweisung "CLOSE #" geschlossen wurde bzw. das Programm mit der Anweisung "END" beendet wurde.

Beispiele

A) Erstellen einer relativen Datei auf Kanal 2. Alle Parameter sind als <S-LIT> in der Anweisung angegeben.

- Schutzstufe = 33
- Kosten = 0
- Anzahl Sätze = 100
- Satzlänge = 50 (Worte)
- LU-Nummer = 9
- Dateiname = TEST
- Neuanlage

```
BUILD #2,"<33>(100:50)9/TEST"
```

Ein-/Ausgabe-Programmierung

B) Erstellen mehrerer Dateien mit einer "BUILD #"-Anweisung:

```
BUILD #2,+A$,"<00>$10.00(100:50)TEST!",B$,#6,C$
```

Erläuterung:

- Erstellen/Ersetzen einer Text-Datei auf Kanal 2, deren Bezeichnung in der Variable A$ abgestellt ist.
- Ersetzen der relativen Datei TEST auf Kanal 3. Die Dateibezeichnung ist als <S-LIT> 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.

Anmerkung:
Der Nachteil beim Erstellen mehrerer Dateien mit einer "BUILD #"-Anweisung ist eine unzureichende Fehlerauswertung.
Tritt bei der Ausführung ein Fehler auf, kann nicht festgestellt werden, bei welcher Datei dieser Fehler entstanden ist.

Ein-/Ausgabe-Programmierung

7.3.4.1 Erstellen formatierter Dateien

Beim Erstellen einer formatierten Datei wird nur der Datei-Kennsatz angelegt.
Das Anlegen der Formatbeschreibungsliste im Kennsatz erfolgt, indem Satz # 0 mit der Anweisung

```
WRITE #
```

direkt (<N-EXPR2> # 0) geschrieben wird.
Alle Datensätze dieser angelegten Datei haben das mit "WRITE #" für Satz 0 festgelegte Format.

Beispiel: Anlegen einer formatierten Datei mit dem Namen TEST auf Kanal 2.

- Schutzstufe = 03
- Kosten = 00
- LU-Nummer = 0

Format: Feld 1 = String, 20 Byte
Feld 2 = Numerische Variable 2%
Feld 3 = Numerische Variable 2%
Feld 4 = Numerische Variable 2%
Feld 5 = String, 10 Byte

```
.
.
.
DIM A$(20),B$(10)
.
BUILD #2,"<03>TEST"
WRITE #2,0;A$,A,B,C,B$;
.
.
.
```

7.3.4.2 Erstellen relativer Dateien

Da relative Dateien einen zusammenhängenden Bereich auf einer Magnetplatte belegen, muß der gesamte Bereich zur Aufnahme der Datei bereits beim Erstellen reserviert werden. Um dieses zu ermöglichen, muß in der "BUILD #"-Anweisung die Anzahl der Datensätze und die Satzlänge angegeben werden.

Die Angaben

- Anzahl Datensätze und
- Satzlänge in Worten

werden in Klammern gesetzt und durch ":" voneinander getrennt.

Beispiel: Anlegen einer relativen Datei mit dem Namen TEST auf Kanal 2:

- Schutzstufe = 0
- Kosten = 0
- LU-Nummer = 1
- Anzahl Sätze = 1500
- Satzlänge = 50 Worte (100 Byte)

```
BUILD #2,"(1500:50)1/TEST"
```

Anmerkung:

Beim Berechnen der Satzlänge ist zu beachten, daß die Grenzzeichen von Strings Bestandteile der der Datensätze sind.

Beim Berechnen der Satzlänge sind die Grenzzeichen mit zu berücksichtigen.
Numerische Variable beginnen innerhalb der Datensätze immer in einem neuen Wort.
Das kann, wenn String-Variable und numerische Variable im Datensatz aneinander grenzen, dazu führen, daß jeweils 1 Byte ungenutzt bleibt.
Die Länge eines Datensatzes kann also auch durch die Lage der Variablen im Datensatz beeinflußt werden.

Ein-/Ausgabe-Programmierung

Platzbedarf der verschiedenen Variablen:

Variable	Platzbedarf in Worten
Numerisch 1%	1
Numerisch 2%	2
Numerisch 3%	3
Numerisch 4%	4
String	(Dimens. Anzahl Byte + 1) / 2

Beispiel: Berechnen der Satzlänge in Worten.
Folgende Variablen sind in der angegebenen Reihenfolge im Datensatz vorhanden:

Feld 1 = String, dimensioniert für 10 Byte

Feld 2 = Numerische Variable 3%

Feld 3 = String, dimensioniert für 16 Byte

Feld 4 = String, dimensioniert für 14 Byte

Feld 5 = Numerische Variable 2%

Berechnung:

Feld 1: (10 + 1) /2	=	5,5 Worte
Feld 2: 3% Variable	=	3,0 Worte
Feld 3: (16 + 1) /2	=	8,5 Worte
Feld 4: (14 + 1) /2	=	7,5 Worte
Feld 5: 2% Variable	=	2,0 Worte
Gesamt		26,5 Worte

Da Feld 1 keine ganze Anzahl Worte belegt (5,5) und das folgende Feld eine numerische Variable ist, bleibt Byte 1 ungenutzt bzw. Feld 1 belegt 6 Worte.
Die gesamte Länge beträgt also 27 Worte.

7.3.4.3 Erstellen von Index-Dateien

Zum Anlegen von Index-Dateien steht das Dienstprogramm "BUILDXF" zur Verfügung.

7.3.4.4 Erstellen von Text-Dateien

Beim Anlegen von Text-Dateien muß vor der Dateibezeichnung ein "+" angegeben werden.
Text-Dateien bestehen bei der Anlage zunächst nur aus einem Datei-Kennsatz.

Die Angaben:

- Schutzstufe
- Kosten
- Anzahl Sätze
- Satzlänge

sind beim Anlegen von Text-Dateien nicht zugelassen.

Beispiel:

Ersetzen einer Textdatei mit dem Namen TEST auf Kanal 2 und der LU-Nummer = 2.

```
BUILD #2,+"2/TEST"
```

Ein-/Ausgabe-Programmierung

7.3.5 Der Zugriff auf Magnetplatten-Dateien

Für den Zugriff auf Datensätze in Magnetplatten-Dateien stehen in Business-Basic die Anweisungen:

- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #

zur Verfügung.

Alle Magnetplatten-Dateien erlauben sowohl direkten als auch sequentiellen Zugriff.

Zum Bearbeiten des Indexbereiches von Index-Dateien steht die Anweisung:

- SEARCH #

zur Verfügung.

Das Eröffnen und Schliessen von Dateien erfolgt mit den Anweisungen:

- OPEN #
- CLOSE #

Ein-/Ausgabe-Programmierung

7.3.5.1 Die Dateieröffnung

Bevor auf Datensätze in einer Datei zugegriffen werden kann, muß die Datei mit der Anweisung

OPEN #

auf einem beliebigen freien Kanal eröffnet werden.

Eine Magnetplattendatei kann von einem Programm auf mehreren Kanälen gleichzeitig eröffnet werden. Pro Kanal kann jedoch nur jeweils eine Datei eröffnet werden.

Magnetplattendateien können von mehreren unabhängigen Programmen gleichzeitig eröffnet werden.

7.3.5.2 Zugriff auf formatierte Dateien

Der Zugriff auf Datensätze in formatierten Dateien erfolgt mit den Anweisungen:

- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #

Die Anweisungen MAT READ # und MAT WRITE # können nur benutzt werden, wenn Satz 0 mit MAT WRITE # geschrieben wurde, also das Satzformat entsprechend definiert wurde. In diesem Fall kann die Datei im weiteren nur noch mit MAT READ # und MAT WRITE # bearbeitet werden.
Mit der Anweisung PRINT # können in formatierten Dateien nur Strings bearbeitet werden.

Alle Anweisungen ermöglichen sowohl sequentiellen als auch direkten Zugriff auf Datensätze und Datenfelder.

Ein-/Ausgabe-Programmierung

Sequentieller Zugriff

Sequentieller Zugriff auf einen Datensatz erfolgt, wenn in der Anweisung keine Angabe der relativen Satznummer (<N-EXPR2> ist nicht angegeben) gemacht ist oder wenn als relative Satznummer -1 vorgegeben ist. Soll sequentiell auf Datensätze zugegriffen, aber nur bestimmte Felder der Datensätze übertragen werden, muß <N-EXPR2> = -1 sein.

Beispiel: Sequentielles Lesen und übertragen der Datenfelder ab Feldnummer 0 in die Variable A, B, C und X$.

```
READ #C;A,B,C,X$;
```

Sequentielles Schreiben und Übertragen der Variablen D, E, F und Y$ in den aktuellen Datensatz ab Feld # 3. Der Wert -1, der den sequentiellen Zugriff bewirkt, wird in der Variablen R vorgegeben.

```
WRITE #C,R,3;D,E,F,Y$;
```

Zugriff auf den zuletzt bearbeiteten Satz

Ist als relative Satznummer der Wert -2 vorgegeben, wird auf den Satz zugegriffen, der zuletzt gelesen oder geschrieben wurde. Wird direkt nach der Dateieröffnung -2 vorgegeben, führt das zu dem Basic-Fehler = 51.
Die Möglichkeit des Zugriffs auf den zuletzt bearbeiteten Datensatz erleichtert das Verändern des Inhaltes von Datensätzen.

Beispiel: Sequentielles Lesen in einer Datei, Verändern und Zurückschreiben der geänderten Datensätze.

```
READ #3;A,B,C        /* SEQUENTIELL LESEN
GOSUB 2000           /* UPDATEN DER VARIABLEN
WRITE #3,-2;A,B,C;   /* ZURÜCKSCHREIBEN
```

Nach dem Lesen ist der Datensatz vor dem Zugriff anderer Teilnehmer geschützt und wird nach dem Zurückschreiben wieder freigegeben.

Direkter Zugriff

Ist als relative Satznummer (<N-EXPR2>) ein Wert >= 0 vorgegeben, wird auf den adressierten Satz direkt zugegriffen.

Ein-/Ausgabe-Programmierung

Es besteht jederzeit die Möglichkeit, die Zugriffsart zu wechseln.

Beispiel: Direktes Lesen von Satz # 100, anschließend sequentiell weiterarbeiten.

```
READ #3,100;A$        /* RANDOM LESEN
GOSUB 2000            /* VERARBEITUNG
READ #3;A$            /* SEQUENTIELL LESEN
```

Feldverarbeitung

Alle Datensätze einer formatierten Datei haben hinsichtlich Anordnung und Art der einzelnen Datenfelder die gleiche Struktur. Bei jedem Zugriff wird geprüft, ob die Variablentypen in der jeweiligen Anweisung identisch mit dem zugehörigen Eintrag im Dateikennsatz sind.
Besteht keine Übereinstimmung, wird BASIC-Fehler # 54 gemeldet. Daten werden im Fehlerfall nicht übertragen.
Der gezielte Zugriff auf bestimmte Felder eines Datensatzes erfolgt durch Vorgabe der Feldnummer relativ zum Satzanfang (Feld 0).
Die relative Feldnummer wird unter <N-EXPR3> angegeben. Bedingung ist, daß <N-EXPR2> codiert ist.

Die Übertragung der Daten erfolgt feldweise.

- Bei numerischen Variablen durch logische Konvertierung. D.h. es wird nicht physikalisch übertragen, sondern die Werte werden wie bei der Anweisung "LET" konvertiert. Z.B. können 1%-Variable in Felder für 2%-Variable übertragen werden und umgekehrt.
  Bei Ausnutzung dieser Möglichkeit kann es zum Basic-Fehler # 15 kommen, wenn z.B. aus einem Datenfeld auf Magnetplatte, das für 2%-Variable reserviert ist, ein Wert > +- 7999 in eine 1%-Variable übertragen wird.

- Bei Strings wird die Übertragung beendet durch:
  - Auftreten eines Grenzzeichens im Quell-String bei READ #, WRITE # und PRINT #.
  - Erreichen von String-Ende im Quell- oder Ziel-String bei READ #, WRITE # und PRINT #.

Mit der Anweisung PRINT # darf kein String, der länger als sein Zielfeld ist, in eine formatierte Datei übertragen werden.

Die Anweisung MAT READ # bzw. MAT WRITE # bearbeitet die codierte Variable in ihrer Gesamtheit. Grenzzeichen beenden die Übertragung nicht, sondern nur das Erreichen von String-Ende im Quell- oder Ziel-String.

Ein-/Ausgabe-Programmierung

Dateiendebehandlung

Wird in einer Anweisung zum Lesen (READ # oder MAT READ #) ein Datensatz adressiert, der hinter dem höchsten geschriebenen Datensatz liegt, wird Basic-Fehler # 52 gemeldet.

Anmerkung: Bei formatierten Dateien ist die Bearbeitung mehrerer Datensätze mit einer Anweisung nicht möglich.

7.3.5.3 Zugriff auf relative Dateien

Der Zugriff auf Datensätze in relativen Dateien erfolgt mit den Anweisungen:

- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #

Alle Anweisungen ermöglichen sowohl sequentiellen als auch direkten Zugriff auf Datensätze.

Sequentieller Zugriff

Sequentieller Zugriff auf einen Datensatz erfolgt, wenn in der Anweisung keine Angabe der relativen Satznummer (<N-EXPR2> ist nicht angegeben) gemacht ist bzw. als relative Satznummer -1 vorgegeben ist. Soll sequentiell auf Datensätze zugegriffen werden, aber nur bestimmte Bereiche der Datensätze übertragen werden (die Übertragung beginnt nicht mit Byte 0), muß <N-EXPR2> codiert und = -1 sein.

Beispiel: Sequentielles Lesen und Übertragen ab Byte-Nummer 5 in die Variable A$. Der gelesene Datensatz wird gesperrt.

```
READ #3,-1,5;A$
```

Sequentielles Schreiben der Variablen A, B und A$ ab Byte Nummer 0. Die Daten werden als String auf Magnetplatte abgestellt. Die numerischen Variablen werden mit "USING" über

eine Maske aufbereitet.

```
PRINT #3;USING "#####";A,B,A$;
```

Zugriff auf den zuletzt bearbeiteten Satz

Ist als relative Satznummer der Wert -2 vorgegeben, wird auf den Satz zugegriffen, der zuletzt gelesen oder geschrieben wurde.
Wird direkt nach der Dateieröffnung -2 vorgegeben, führt das zu dem Basic-Fehler # 51.
Die Möglichkeit des Zugriffs auf den zuletzt bearbeiteten Datensatz erleichtert das Verändern des Inhaltes von Datensätzen.

Beispiel: Direktes Lesen eines Satzes, dessen relative Satznummer in der Variablen "R" vorgegeben ist. Der Inhalt des Satzes wird verändert und anschließend zurückgeschrieben.

```
.
READ #3,R;A,B,C      /* DIREKT LESEN
GOSUB 2000           /* SATZINHALT VERÄNDERN
WRITE #3,-2;A,B,C;   /* ZURÜCKSCHREIBEN
```

Nach dem Lesen ist der Datensatz vor dem Zugriff anderer Teilnehmer geschützt und wird erst nach dem Zurückschreiben wieder freigegeben.

Direkter Zugriff

Ist als relative Satznummer (<N-EXPR2>) ein Wert >= 0 vorgegeben, wird auf den adressierten Satz direkt zugegriffen.
Es besteht jederzeit die Möglichkeit, die Zugriffsart zu wechseln. Zum Beispiel kann nach direktem Zugriff auf einen Datensatz ab dem aktuellen Stand des Record-Pointers sequentiell weitergearbeitet werden.

Beispiel: Direktes Lesen eines Satzes, dessen relative Satznummer in der Variablen "R" vorgegeben ist. Anschließend sequentielle Verarbeitung.

```
.
READ #3,R;A$         /* DIREKT LESEN
GOSUB 2000
READ #3;A$           /* SEQUENTIELL LESEN
```

Ein-/Ausgabe-Programmierung

Feldverarbeitung

Bei der Bearbeitung von Daten in relativen Dateien wird durch das IOCS keine Überprüfung der Datentypen wie bei formatierten Dateien vorgenommen.
Die Daten werden 1:1 übertragen. Der Programmierer muß dafür sorgen, daß Ziel- und Quellbereich den gleichen Feldtyp haben. Zum Beispiel muß eine 2%-Variable von der Magnetplatte auch unbedingt in eine 2%-Variable übertragen werden und umgekehrt.

Durch die Möglichkeit der Vorgabe einer Byte-Adresse (Displacement) unter <N-EXPR3> relativ zum Anfang des adressierten Datensatzes kann ab einem beliebigen Byte im Bereich von 0 bis 65335 mit der Übertragung begonnen werden. Das mit <N-EXPR3> adressierte Byte kann außerhalb des adressierten Datensatzes liegen.

Besteht ein Datensatz aus verschiedenen Feldtypen (Strings und numerische Variablen), muß bei der Ermittlung des Displacements berücksichtigt werden, daß:

- numerische Variable immer in einem neuen Wort (Wortgrenze) beginnen.
- Strings immer mit einem Grenzzeichen abgeschlossen werden (Ausnahmen siehe Besonderheiten).

Zur Ermittlung der Anfangsadressen der einzelnen Datenfelder relativ zum Satzanfang werden die folgenden Formeln verwendet:

```
Numerische Variable: Dimensionierte Länge in Worten * 2
String - Variable  : Dimensionierte Länge in Byte + 1
String - Literale  : Anzahl Zeichen/Literal + 1
```

Besonderheiten:

- Die Anweisung Write # stellt Grenzzeichen ab:
  - Nach Übertragung von Literalen und
  - Nach Übertragung von Strings in dimensionierter Länge.

- Die Anweisung READ # erwartet nach der Übertragung eines Strings hinter dem letzten übertragenen Zeichen unbedingt ein Grenzzeichen, dh., das dem letzten übertragenen Zeichen eines Strings folgende Zeichen wird übersprungen.

Es kann also zu Problemen führen, wenn mit WRITE # Sub-Strings geschrieben werden, da die entsprechende Anweisung READ # diese Daten nicht korrekt übertragen kann.

Beispiel: Die mit der Anweisung

```
WRITE #3;A$(5),B$;
```

geschriebenen Daten können mit der Anweisung

```
READ #3;A$(5),B$;
```

nicht korrekt gelesen werden.

Begründung:

```
Dimensionen = A$(10),B$(10)
Inhalt      = A$ = "ABCDEFGHIJ"
              B$ = "1234567890"
```

Nach der Durchführung der Anweisung

```
WRITE #3;A$(5),B$;
```

hat der Datensatz den Inhalt:

```
EFGHIJ1234567890
```

Es wurde kein Grenzzeichen abgestellt:
Nach Durchführung der Anweisung

```
READ # 3;A$(5),B$
```

haben die Zielvariablen den Inhalt:

```
A$ (5) = "EFGHIJ"
B$     = "234567890"
```

Die "1" wurde nicht übertragen, da READ #, nachdem das Ende von A$ erreicht war, ein Grenzzeichen im Datensatz erwartet hat und es übersprungen hat.

Der Inhalt dieses Datensatzes kann nur dann korrekt übertragen werden, wenn mit einer Anweisung READ # oder MAT READ # in eine String-Variable gelesen wird, die groß genug ist, sämtliche Daten aufzunehmen.

Es sollte also mit zwei Anweisungen:

```
READ #C;A$(5);
READ #C,-2,6;B$
```

ausgeführt werden.

Dateiendebehandlung:

Wird ein Datensatz adressiert, der hinter dem durch BUILD # oder FORMAT angelegten Dateiende liegt, wird Basic-Fehler # 51 gemeldet.

Ein-/Ausgabe-Programmierung

Anmerkung

Bei relativen Dateien ist die Bearbeitung mehrerer Datensätze mit einer Anweisung möglich.

Beispiel: Ein Datensatz besteht aus einem String in Länge von 80 Byte. Es sollen 5 Datensätze mit einer Anweisung gelesen werden.

```
.
DIM A$(80),B$(80),C$(80),D$(80),E$(80)
.
READ #3,R;A$,B$,C$,D$,E$;
```

oder:

```
.
DIM A$(400)
.
READ #3,R;A$(1,80),A$(81,160),A$(161,240)...
```

oder:

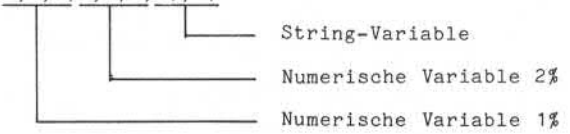
```
.
DIM A$(400)
.
MAT READ #3,R;A$
```

Es ist darauf zu achten, daß auch in solchen Fällen der Record-Pointer nur um "1" erhöht wird. Nach Durchführung einer dieser Anweisungen hätte der Record-Pointer den Inhalt R+1.

Tips für das Arbeiten mit relativen Dateien

- Datensätze, die aus verschiedenen Feldtypen bestehen, sollten so aufgebaut sein, daß die verschiedenen Feldtypen ganze Blöcke bilden.

Beispiel: A,B,C,D,E,F,A$,B$



Dadurch wird die Berechnung der Satzlänge und Displacements erleichtert. Zusätzlich besteht die Möglichkeit, daß Kapazität eingespart wird.

- Nach Möglichkeit keine Sub-Strings verarbeiten.

7.3.5.4 Zugriff auf Index-Dateien

Index-Dateien haben die gleiche Struktur wie relative Dateien. Der durch das Dienstprogramm "BUILDXF" oder durch Search Modus 0 auf der Magnetplatte reservierte Bereich unterteilt sich in:

- 1-15 Schlüsselverzeichnisse zur Aufnahme der Ordnungsbegriffe (Indexbereich)
  und
- den Bereich zur Aufnahme der Datensätze (Datenbereich).

Grundsätzliche Punkte, die bei der Bearbeitung von Index-Dateien unbedingt beachtet werden müssen, sind:

- Zugriffe auf Index-Dateien erfolgen in der Regel mit mehreren Basic-Anweisungen.
- Ein Zugriff auf eine Index-Datei kann über mehrere Zeitscheiben verteilt sein.
- Ein Datensatz kann mit mehreren Ordnungsbegriff-Verzeichnissen verkettet sein.

Diese Besonderheiten müssen bei der Organisation und Erstellung von Programmen, die Index-Dateien bearbeiten, berücksichtigt werden.

Der Zugriff auf den Indexbereich und den Datenbereich erfolgt mit unterschiedlichen Anweisungen. Der Zugriff auf Datensätze in Index-Dateien erfolgt mit den Anweisungen:

- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #

Für den Zugriff auf Datensätze gelten die gleichen Regeln wie bei relativen Dateien (siehe auch Pkt.: 7.3.5.3). Zusätzlich ist darauf zu achten, daß:

- die beiden ersten Bytes eines jeden freien Datensatzes den Verweis auf den nächsten freien Datensatz enthalten. Aus diesem Grund sollten die beiden ersten Byte eines jeden Datensatzes vom Anwenderprogramm nicht angesprochen werden (Arbeiten mit Displacement 2).
- der erste Datensatz im Datenbereich nicht Satz # 0 ist. Die Nummer des ersten Datensatzes relativ zum Dateianfang kann mit der Anweisung SEARCH # ermittelt werden.

Ein-/Ausgabe-Programmierung

Der Zugriff auf Datensätze erfolgt in der Regel durch die Vorgabe eines Ordnungsbegriffs über den Indexbereich.

Für die Bearbeitung der Indexbereiche steht die Anweisung:

- SEARCH #

zur Verfügung.

Die Anweisung SEARCH #

<SEARCH #> ::= SEARCH #<N-EXPR1>,<N-EXPR2>,<N-EXPR3>;
<S-VAR>,<N-VAR1>,<N-VAR2>

Funktion:

SEARCH # = Anweisung.

<N-EXPR1> = Nummer des Kanals, auf dem die Datei eröffnet ist.

<N-EXPR2> = Modus, der die auszuführende Funktion definiert.
Als Funktionen stehen zur Verfügung:

<N-EXPR2>	Bedeutung
0	Verzeichnis im Dateikennsatz eintragen. Im Dateikennsatz eingetragene Verzeichnisse initialisieren.
1	Übergabe von: - Ordnungsbegrifflänge, - Nummer des ersten Datensatzes, - Anzahl der freien Sätze. Durchführung von: - Übergabe der Nummer des ersten freien Datensatzes und Löschen dieses Satzes aus der Liste der freien Datensätze mit Sperren dieses Datensatzes.

<N-EXPR2>	Bedeutung
	- Eintragen einer vorgegebenen Satznummer in die Liste der freien Sätze in Abhängigkeit von <N-EXPR3> und <N-VAR2> unter Beachtung des Sperrkennzeichens.
2	Suchen eines Ordnungsbegriffes im angegebenen Verzeichnis. Satzsperre wird gesetzt und ausgewertet.
3	Im angegebenen Verzeichnis den gegenüber in <S-VAR> angegebenen nächst höheren OB suchen. Satzsperre wird gesetzt und ausgewertet.
4	Einfügen eines Ordnungsbegriffs in das angegebene Verzeichnis. Falls notwendig, wird das Verzeichnis reorganisiert.
5	Löschen eines Ordnungsbegriffs im angegebenen Verzeichnis unter Beachtung einer Satzsperre.
6	Löschen einer Indexdatei. Die Datensätze werden nicht gelöscht, sondern nur neu verkettet.
7	Diese Funktion entfällt.

<N-EXPR3> = Definiert die Nummer des zu bearbeitenden Schlüsselverzeichnisses (1-15). Die Angabe <N-EXPR3> = 0 ist dann zugelassen, wenn <N-EXPR2> = 0 oder 1 ist.

<S-VAR> = String-Variable, in der der Ordnungsbegriff vorgegeben bzw. vom IOCS übergeben wird. <S-VAR> muß für mindestens so viele Bytes dimensioniert sein, wie der Ordnungsbegriff lang ist.

<N-VAR1> = Numerische Variable, mindestens 2%. Diese Variable dient zur Übergabe der relativen Satznummer an das Anwenderprogramm bzw. an das IOCS.

<N-VAR2> = Numerische Variable zur Parameterübergabe an das Betriebssystem und zur Übergabe von Statusmeldungen an das Anwenderprogramm.

Die Funktionen (Modi) von SEARCH #

<N-EXPR2> = 0 und <N-EXPR3> <> 0
Das angegebene Verzeichnis wird mit der in <N-VAR1> vorgegebenen OB-Länge im Dateikennsatz eingetragen. Werden für eine Datei mehrere Verzeichnisse definiert, müssen die Verzeichnisnummern lückenlos aufsteigend mit 1 beginnend vorgegeben werden.

<N-EXPR2> = 0 und <N-EXPR3> = 0
Sämtliche Verzeichnisse, die mit Search Modus 0, <N-EXPR3> <> 0 angelegt wurden, werden initialisiert.
Folgt auf einen Search Modus 0, mit <N-EXPR3> = 0 abermals ein Search Modus 0, wird als Status der Wert 8 übergeben.

<N-EXPR2> = 1 und <N-EXPR3> <> 0
Diese Funktion liefert die Ordnungsbegrifflänge des unter <N-EXPR3> angegebenen Verzeichnisses. Die OB-Länge wird in <N-VAR2> übergeben. Ist das angegebene Verzeichnis nicht vorhanden, wird <N-VAR2> auf 0 gesetzt.

<N-EXPR2> = 1 und <N-EXPR3> = 0 und <N-VAR2> = 0
Diese Funktion übergibt in <N-VAR1> die relative Satznummer des ersten Datensatzes im Datenbereich.

<N-EXPR2> = 1 und <N-EXPR3> = 0 und <N-VAR2> = 1
Diese Funktion übergibt in <N-VAR1> die Anzahl freier Datensätze.

<N-EXPR2> = 1 und <N-EXPR2> = 0 und <N-VAR2> = 2
Diese Funktion übergibt in <N-VAR1> die relative Satznummer des ersten freien Datensatzes und löscht diesen Satz aus der Liste der freien Datensätze.
Gleichzeitig wird der übergebene Datensatz gesperrt.

<N-EXPR2> = 1 und <N-EXPR3> = 0 und <N-VAR2> = 3
Diese Funktion übernimmt aus <N-VAR1> eine relative Satznummer und trägt diesen Satz in die Liste der freien Datensätze ein. Der Datensatz wird erst dann zurückgegeben, wenn kein anderer Teilnehmer den Satz gesperrt hat.

<N-EXPR2> = 2 Diese Funktion sucht in dem angegebenen Verzeichnis den unter <S-VAR> angegebenen Ordnungsbegriff. Die Anzahl Zeichen, die verglichen werden, wird immer von <S-VAR> bestimmt.

Es wird verglichen, bis:
- das dimensionierte Ende von <S-VAR> erreicht ist oder
- ein Grenzzeichen in <S-VAR> auftritt.

Gleichheit wird erkannt, wenn alle verglichenen Zeichen des OB´s in <S-VAR> und im Verzeichnis übereinstimmen. Dieses Verhalten kann auch dann zur Gleichheit führen, wenn der in <S-VAR> vorgegebene OB kürzer ist als der im Verzeichnis stehende. Kommt es vor, daß die OBs, die in <S-VAR> vorgegeben werden, verschiedene Länge haben, sollte <S-VAR> nach rechts mit Blanks aufgefüllt werden.
Das Verfahren, nur bis zu dem ersten auftretenden Grenzzeichen zu vergleichen, bietet die Möglichkeit, Teilordnungsbegriffe vorzugeben.

Bei Gleichheit wird der gesamte OB aus dem Verzeichnis in <S-VAR> übergeben. In <N-VAR1> wird die relative Satznummer des dazugehörigen Datensatzes übergeben.

Wird kein dem in <S-VAR> entsprechender OB im Verzeichnis gefunden, bleiben <S-VAR> und <N-VAR1> unverändert. Als Status wird in <N-VAR2> der Wert 1 übergeben. Durch Search Modus 2 werden Satzsperren gesetzt und auch ausgewertet.

<N-EXPR2> = 3 Diese Funktion sucht im angegebenen Verzeichnis den nächstgrößeren OB gegenüber dem in <S-VAR> vorgegebenen. Ist ein größerer OB gefunden, wird dieser in <S-VAR> übergeben. In <N-VAR1> wird die relative Satznummer des dazugehörigen Datensatzes übergeben.

Ist kein größerer OB vorhanden, bleiben <S-VAR> und <N-VAR1> unverändert. Als Statusmeldung wird in <N-VAR2> der Wert 2 übergeben.
Durch Search Modus 3 werden Satzsperren gesetzt und auch ausgewertet.

<N-EXPR2> = 4 Diese Funktion fügt den in <S-VAR> vorgegebenen OB in das angegebene Verzeichnis ein.
Der OB darf in dem angegebenen Verzeichnis noch nicht vorhanden sein. Gleichheit wird erkannt, wenn <S-VAR> incl. Grenzzeichen identisch mit dem OB im Verzeichnis ist.
Die relative Satznummer des Datensatzes, auf den der OB verweisen soll, muß in <N-VAR1> vorgegeben werden, sodaß eine Verkettung zwischen OB und Datensatz vorgenommen werden kann.
Konnte der vorgegebene OB in das Verzeichnis eingefügt werden, wird als Statusmeldung in <N-VAR2> der Wert 0 übergeben.

Ist der OB im angegebenen Verzeichnis bereits vorhanden, wird in <N-VAR1> die relative Satznummer übergeben, auf den der vorhandene OB verweist. Als Statusmeldung wird in <N-VAR2> der Wert 1 übergeben.

Besteht beim Einfügen eines OBs die Notwendigkeit einer Reorganisation, wird diese automatisch durchgeführt.

<N-EXPR2> = 5 Diese Funktion löscht den in <S-VAR> vorgegebenen OB aus dem angegebenen Verzeichnis. Das Verfahren zum Vergleichen der OBs entspricht dem beim Suchen eines OB´s angewandten Verfahren. Ist der vorgegebene OB im Verzeichnis vorhanden, wird er aus dem Verzeichnis gelöscht. In <N-VAR1> wird die relative Satznummer des dazugehörigen Datensatzes übergeben, und <N-VAR2> auf 0 gesetzt.

Ist der vorgegebene OB nicht im Verzeichnis vorhanden, bleiben <S-VAR> und <N-VAR1> unverändert. Als Statusmeldung wird in <N-VAR2> der Wert 1 übergeben. Search Modus 5 nimmt eine Auswertung der Satzsperre vor.

<N-EXPR2> = 6 Dieser Search Modus ermöglicht das Löschen einer Indexdatei. Der Verzeichnisbereich hat nach der Ausführung des Modus 6 den gleichen Zustand wie nach der Neuanlage mit BUILDXF. Die Datensätze werden nicht gelöscht, sondern nur neu verkettet.
Diese Funktion gibt keine Zeit an andere Teilnehmer ab.
Ist die Datei von einem anderen Teilnehmer eröffnet, wird Status 15 gemeldet. Für diese Funktion wird nur <N-EXPR1> und <N-EXPR2> ausgewertet. In <N-VAR1> wird nach erfolgreicher Durchführung der Funktion die Anzahl der freien Datensätze übergeben.

<N-EXPR2> = 7 Durch die Erweiterung des Search Modus 4 ist Search Modus 7 überflüssig geworden. Ein Absetzen dieses Search Modus bleibt für die Datei ohne Auswirkung. Die Status-Variable wird auf 0 gesetzt.

Ein-/Ausgabe-Programmierung

Statusmeldungen von SEARCH #

Nach Durchführung einer jeden SEARCH # - Anweisung ist die Statusvariable <N-VAR2> unbedingt abzuprüfen. Folgende Statusmeldungen können nach Durchführung der verschiedenen Funktionen von SEARCH # in <N-VAR2> abgestellt sein:

<N-VAR2>	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 Index-Datei.
5	Unbestimmter Fehler. Tritt dieser Fehler auf, kann davon ausgegangen werden, daß die Datei zerstört ist. Ein Weiterarbeiten ist nicht ratsam.
6	Unzulässige Reihenfolge bei Angabe der Verzeichnisse (nur Modus 0).
7	Datei ist keine relative Datei (nur Modus 0).
8	Modus 0, <N-VAR1> = 0 wurde bereits ausgeführt (nur Modus 0).
10	Zu viele Verzeichnisse (max. 15 pro Datei).
12	Datei ist zu klein, um alle Verzeichnisse aufzunehmen (nur Modus 0, <N-VAR1> = 0).
15	Datei von anderem Teilnehmer eröffnet.

Regeln für die Index-Datei Programmierung

Bei der Bearbeitung von Index-Dateien sollten die im folgenden genannten Regeln berücksichtigt werden!

- Datensätze stets mit Displacement "2" schreiben, also die beiden ersten Bytes im Datensatz freilassen. Diese beiden Bytes müssen bereits beim Erstellen der Datei (BUILDXF) reserviert werden!

- Der Ordnungsbegriff sollte stets auch im Datensatz vorhanden sein und nach dem Lesen des Datensatzes mit dem in <S-VAR> vorgegebenen OB verglichen werden.

- Ist ein Datensatz mit OBs in mehreren Verzeichnissen verkettet, sind zuerst alle auf den Datensatz verweisenden OBs aus den Verzeichnissen zu löschen. Anschließend kann der Datensatz selbst in die Liste der freien Sätze aufgenommen werden.

- Nach Durchführung einer SEARCH # - Anweisung grundsätzlich den Status (<N-VAR2>) abfragen.

- Beim Bearbeiten der Datensätze müssen alle im Kapitel 7.3.5.3, Zugriff auf relative Dateien, angesprochenen Punkte beachtet werden.

Programmbeispiel:

Die Behandlung von Statusmeldungen ist hierbei nur angedeutet. Es ist selbstverständlich, daß in Anwenderprogrammen auf Statusmeldungen gezielt reagiert werden soll. In dem Beispiel werden folgende Variablennamen verwendet:

```
C   = Kanalnummer
D   = Verzeichnisnummer
V$  = Ordnungsbegriff
V1  = Relative Satznummer
V2  = Parameter und Statusvariable
```

Der Modus von SEARCH # wird in dem Beispiel als numerisches Literal angegeben.

Beispiel Direktes Lesen eines Datensatzes. Der gesuchte OB ist "AA".

```
.
9000 LET V$="AA"
9010 SEARCH #C,2,D;V$,V1,V2  /* OB SUCHEN
9020 IF V2 GOTO 9500         /* STATUS <> 0
9030 READ #C,V1,2;R$         /* DATENSATZ
                                LESEN
9040 IF V$<>R$(X,X+1) GOTO 9600
                            /* OB VERGLEICHEN
.
9500 IF V2=1 PRINT "SATZ NICHT VORHANDEN";
.
.
9600 PRINT "SATZ VON ANDEREM TEILNEHMER
     GELÖSCHT";
```

Ein-/Ausgabe-Programmierung

7.3.5.5 Zugriff auf Text-Dateien

Der Zugriff auf Daten in Textdateien ist mit den Anweisungen

- READ #
- WRITE #
- PRINT #

möglich.

In Textdateien dürfen nur String-Literale und/oder der Inhalt von String-Variablen geschrieben werden (Ausnahme PRINT #). Beim Lesen aus Textdateien sind als Ziel-Variable nur String-Variable zugelassen.

Als Besonderheiten sind zu beachten, daß:

- bei der Anweisung READ # die Übertragung durch Auftreten eines "CR"-Codes, eines Grenzzeichens oder des dimensionierten Endes im Ziel-String beendet wird.
- die Angabe von <N-EXPR2> = -2 nicht den Zugriff auf den zuletzt bearbeiteten Satz, sondern die Fortsetzung der Übertragung bei Auftreten eines "CR"-Codes bewirkt.
- in der WRITE # - Anweisung keine numerischen Variablen und keine numerischen Ausdrücke angegeben sein dürfen.
- die Funktion der Trennzeichen "," und ";"bei der Anweisung PRINT # unbedingt zu beachten sind.
- PRINT # hinter dem zuletzt übertragenen Zeichen grundsätzlich ein Grenzzeichen abstellt.

Der Record-Pointer für Textdateien besteht aus der aktuellen Blocknummer und der Adresse des dem letzten übertragenen Zeichen folgenden Bytes relativ zum Blockanfang.

Ein-/Ausgabe-Programmierung

Sequentieller Zugriff

Sequentieller Zugriff erfolgt, wenn in der Anweisung keine Angabe zur relativen Satznummer (<N-EXPR2> ist nicht angegeben) gemacht ist bzw. als relative Satznummer -1 oder -2 vorgegeben ist.

Anmerkung: Wird sequentiell zugegriffen, muß, falls codiert, <N-EXPR3> = 0 sein.

Beispiel: Sequentielles Lesen und Ausgeben der gelesenen Zeilen am Bildschirm. Die Anzahl auszugebender Zeilen ist in der Variablen I vorgegeben.

```
.
FOR I1=0 TO I-1
READ #C;R$;                 /*TEXTDATEI LESEN
PRINT R$                    /*BILDSCHIRMAUSGABE
NEXT I1
.
```

Zugriff auf den zuletzt bearbeiteten Satz

Diese Möglichkeit wird vom IOCS für Textdateien nicht unterstützt. Die Realisierung ist nur durch direkten Zugriff, verwaltet durch das Anwenderprogramm, möglich.

Direkter Zugriff

Da die Datensätze (Zeilen) in einer Textdatei verschieden lang sind, ist ein gezielter Zugriff auf Zeilen nicht möglich.
Um trotzdem auf ein bestimmtes Byte in einer Textdatei zuzugreifen, wird für den direkten Zugriff bei Vorgabe von <N-EXPR2> => 0, eine Satzlänge von 512 Byte angenommen. Die relative Satznummer ist bei Textdateien also die Nummer eines Blocks (Sektors) relativ zum Dateianfang.
Es besteht jederzeit die Möglichkeit, die Zugriffsart zu wechseln. Zum Beispiel kann nach dem Zugriff auf einen beliebigen Punkt innerhalb der Textdatei ab dem aktuellen Stand des Record-Pointers sequentiell weitergearbeitet werden.

Ein-/Ausgabe-Programmierung

Beispiel: Direktes Lesen ab Byte 0 im Block Nummer 3 und anschließender sequentieller Verarbeitung.

```
READ #C,3;R$            /* DIREKT LESEN
GOSUB 2000
READ #C;R$;             /* SEQUENTIELL LESEN
```

Anmerkung: Aufgrund der Tatsache, daß jede Übertragung in eine Textdatei mit einem Grenzzeichen abgeschlossen wird (außer: Schreiben von Sub-Strings mit WRITE #) und ein Grenzzeichen praktisch das Ende der Textdatei anzeigt, ist direktes Schreiben in Textdateien wenig sinnvoll.
Falls in einer Textdatei ein Grenzzeichen steht, muß, um Daten hinter dem Grenzzeichen zu adressieren, direkt auf dem Byte hinter dem Grenzzeichen aufgesetzt werden.

Beispiel:

```
.
9000 IF ERR 0 GOTO 9100   /*FEHLERBEHANDLUNG
9010 READ #C;R$           /*TEXTDATEI LESEN
9020 IF R$="" GOSUB 9200  /*LEERSTRING ??
9030 ....
9040 ....                 /*VERARBEITUNG
9050 ....
.
.
9100 IF SPC 8=52 PRINT "DATEIENDE"
9110 CHAIN ""
.
.
9200 READ #C,CHF103,CHF203+1;R$;
                          /*AUFSETZEN HINTER
                            DEM GRENZZEICHEN
9210 IF R$="" GOTO 9200   /*LEERSTRING
9220 RETURN
```

Feldverarbeitung

Bei Textdateien existieren keine Felder. Der Zugriff auf bestimmte Bytes innerhalb einer Textdatei ist durch die Angabe der relativen Nummer eines Bytes zum Anfang des adressierten Blocks möglich.
Der maximale Wert, der in <N-EXPR3> vorgegeben werden darf, ist 511. Diese Angabe darf nur bei direktem Zugriff <> 0 sein!

Ein-/Ausgabe-Programmierung

Beispiel: Lesen aus einer Textdatei ab einer in den Variablen X und Y vorgegebenen Adresse in die Variable R$.

```
.
READ #C,X,Y;R$;
.
```

Der Wert von Y darf nicht größer als 511 sein.

Dateiendebehandlung

Das Dateiende in Textdateien kann festgestellt werden durch:

- Lesen eines Grenzzeichens (Der Ziel-String ist "", oder kürzer als die dimensionierte Länge bei <N-EXPR2> = -2).
- Auftreten des Basic-Fehlers # 52. Dies ist allerdings nur dann der Fall, wenn ein Block direkt adressiert wird, in den noch kein Zeichen (incl. Grenzzeichen) geschrieben wurde.

7.3.5.6 Das Schließen von Dateien

Nach der Bearbeitung einer Datei ist der Kanal, auf dem diese Datei eröffnet wurde, mit der Anweisung:

- CLOSE #

zu schließen. Dadurch wird dieser Kanal für die Bearbeitung einer anderen Datei frei.

Kanäle werden auch dann geschlossen, wenn:

- eine "END" - Anweisung auftritt.
- mit einer "CHAIN" - Anweisung ein Processor aufgerufen wird.
- durch ESC und CTL | C oder CTL | Y, ESC und CTL | C ein Programm abgebrochen wird.

Anmerkung: Mit BUILD # eröffnete Kanäle werden nur durch die Anweisung "CLOSE #" und "END" geschlossen!

Mit einer "CLOSE #" - Anweisung können beliebig viele Dateien geschlossen werden. Basic-Fehler # 49 wird gemeldet, wenn ein Kanal geschlossen werden soll, auf dem keine Datei eröffnet ist. Mehrere Kanäle sollten wegen unzureichender Fehlerauswertung nicht mit einer Anweisung geschlossen werden (siehe auch Pkt.: 7.1.9).

7.3.6 Löschen von Dateien

Für das Löschen von Dateien stellt Basic die Anweisung

- KILL

zur Verfügung.
"KILL" dient zum Löschen von Dateien, die auf Magnetplatten gespeichert sind.
"KILL" löscht die ersten zwei Zeichen des Dateinamens im Inhaltsverzeichnis (Datei INDEX). 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 oder INSTALL.

Aufbau:

<KILL> ::=KILL { <S-LIT> / <S-VAR> } [, { <S-LIT> / <S-VAR> }]$_{1}^{n}$

Funktion:

KILL = Anweisung.

<S-LIT>
<S-VAR> = Beliebige Anzahl String-Variable und String-Literale. Bezeichnet die zu löschende(n) Datei(en).
<S-LIT> und <S-VAR> haben folgenden Aufbau:

[LU/] <Dateiname>

LU/ = LU-Nummer der Platte, auf der sich die zu löschende Datei befindet. Ist LU = 0, kann diese Angabe entfallen. Die LU-Nummer wird durch "/" vom Dateinamen getrennt.

<Dateiname> = Name der zu löschenden Datei.

Ein-/Ausgabe-Programmierung

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. Bei dieser Möglichkeit entsteht das Problem einer unzureichenden Fehlerauswertung, denn tritt bei der Ausführung 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.

```
.
KILL A$
```

Ein-/Ausgabe-Programmierung

7.3.7 Die Funktion CHF

Die Funktion CHF (<N-EXPR>) ermöglicht dem Programmierer, die aktuelle Größe einer Datei oder die augenblickliche Zugriffsposition innerhalb einer Datei zu ermitteln.
Der Wert in <N-EXPR> definiert die Kanalnummer und die durchzuführende Funktion.
Für die verschiedenen Arten von Plattendateien werden durch CHF verschiedene Angaben übergeben bzw. bestehen verschiedene Funktionsmöglichkeiten.

Formatierte Dateien

<N-EXPR>	Funktion
= Kanalnummer	Nummer des größten geschriebenen Datensatzes +1.
= Kanalnummer + 100	Nummer des Datensatzes, auf den zuletzt zugegriffen wurde.

Relative Dateien

<N-EXPR>	Funktion
= Kanalnummer	Anzahl Datensätze, die die Datei maximal aufnehmen kann.
= Kanalnummer + 100	Nummer des Datensatzes, auf den zuletzt zugegriffen wurde.

Index-Dateien

<N-EXPR>	Funktion
= Kanalnummer	Anzahl Datensätze, die durch BUILDXF für die Datei reserviert wurden. Diese Angabe beinhaltet auch den Bereich für Schlüsselverzeichnisse.
= Kanalnummer + 100	Nummer des Datensatzes, auf den zuletzt zugegriffen wurde.

Ein-/Ausgabe-Programmierung

Für Index-Dateien können zusätzliche Informationen über den Dateistatus mit SEARCH #, Modus 1, abgefragt werden.

Textdateien

<N-EXPR>	Funktion
= Kanalnummer	Nummer des höchsten geschriebenen Datenblocks relativ zu 0.
= Kanalnummer + 100	Nummer des aktuellen Datenblocks.
= Kanalnummer + 200	Nummer des Bytes, das hinter dem zuletzt übertragenen Byte liegt, relativ zum Anfang des aktuellen Blocks.

Beispiel: Eine Textdatei, die als Log-Datei benutzt wird, soll nicht größer als 50 Blöcke werden. Beim Erreichen von 40 Blöcken wird eine Warnung ausgegeben. Ist der 51. Block angelegt, wird ein Merker gesetzt und das aktuelle Programm bei der nächsten Schnittstelle abgebrochen, um die Log-Datei zu drucken.

```
.
9000 PRINT #C,CHF103,CHF203;M$; /*DIREKT SCHR
9010 IF CHF(103)>50 GOTO 9100   /*ABBRUCH?
9020 IF CHF(103)<40 RETURN      /*WARNUNG?
9030 PRINT TAB(0,24);'LD';"LOG-DATEI DRUCKEN!
9040 SIGNAL 3,20
9050 RETURN
9100 LET M=1                      /*MERKER FÜR
                                    ABBRUCH
9110 PRINT TAB(0,24);'LD';"LOG-DATEI WIRD
     GEDRUCKT";
9120 RETURN
```

Ein-/Ausgabe-Programmierung

7.3.8 Programmbeispiel

Ausgeben einer Textdatei am Drucker

Wird als Ausgabegerät ein Zeilendrucker benutzt, werden die Eingaben "ERSTE DRUCKPOSITION" und "LETZTE DRUCK-POSITION" ignoriert.

```
 10 DIM D$(20),P1$(9),P$(3)
 20 LET P1$="0",P1$
 30 IF ERR 0 GOSUB 350
 40 PRINT 'CS'; TAB (0,24);"TEXTDATEI AUSGEBEN"
 50 INPUT 'CR',"DATEINAME :", TAB (30),D$
 60 OPEN #2,D$
 70 INPUT 'CR',"MAX. SATZLÄNGE :",TAB (30)I
 80 IF FRA I IF I<0 IF I>999 GOTO 70
 90 DIM R$(I)
100 INPUT 'CR',"DRUCKER-DRIVER :", TAB (30)D$
110 INPUT 'CR',"ERSTE DRUCKPOSITION :", TAB (30)P$
120 CALL 62,P$
130 IF P$="" LET P$="000"
140 LET P1$(4- LEN P$,3)=P$
150 INPUT 'CR',"LETZTE DRUCKPOSITION :", TAB (30)P$
160 CALL 62,P$
170 IF P$="" LET P$="000"
180 LET P1$(7- LEN P$,6)=P$
190 IF P1$(1,3)>=P1$(4,6) GOTO 110
200 INPUT 'CR',"BLATTHÖHE :", TAB (30)P$
210 CALL 62,P$
220 IF LEN P$=1 GOTO 200
230 IF P$="" LET P$="072"
240 LET P1$(10- LEN P$,9)=P$
250 OPEN #3;P1$,D$
260 READ #2;R$;
270 IF R$="" GOTO 300
280 PRINT #3;R$;
290 GOTO 260
300 PRINT 'CR';"DATEIENDE"
310 CLOSE #2
320 CLOSE #3
330 PRINT "PROGRAMMENDE";
340 CHAIN ""
350 IF SPC 8>97 RETURN -1
360 IF SPC 8=38 RETURN -2
370 PRINT 'CR';"BASIC-FEHLER # "; SPC 8;"AUF ZEILE #";
    SPC 10
380 PRINT "PROGRAMMENDE";
390 CHAIN ""
```

Ein-/Ausgabe-Programmierung

7.4 Die Drucker

An das System 8870/1 können drei Druckertypen angeschlossen werden:

- Nadeldrucker 150 Z/Sec
- Nadeldrucker 100 Z/Sec
- Zeilendrucker 18000 Zl/h

Für den Programmierer bestehen je nach Druckeranschluß bestimmte Regeln bzw. Einschränkungen.

Die grundsätzlichen Unterschiede zwischen den Druckertypen sind:

- 132 Schreibstellen des Zeilendruckers gegenüber den 178 Schreibstellen des Nadeldruckers.
- Der Nadeldrucker kann mit zwei Formularführungen und einem Einzelformulareinzug ausgerüstet sein.
- Beim Nadeldrucker können die erste und letzte Druckposition per Programm festgelegt werden.

7.4.1 Zugriff auf Druckerdateien

Alle auf dem Drucker ausgegebenen Daten sind im Rahmen von Dateien organisiert. Die einzig mögliche Organisationsart ist sequentiell. Für die Ausgabe von Daten auf einem Drucker sind die gleichen Regeln verbindlich wie für Ausgaben in Textdateien.
Ein Zeilendrucker kann nicht gleichzeitig von mehreren Teilnehmern oder von einem Teilnehmer auf mehreren Kanälen eröffnet sein.
Bei Nadeldruckern mit doppeltem Vorschub können die beiden Vorschubaggregate und der Einzelformulareinzug von mehreren Teilnehmern gleichzeitig eröffnet werden, da jede Formularsteuerung als logisches unabhängiges Gerät betrieben wird. Der linke Leporello und der Einzelformulareinzug können aus Sicherheitsgründen nur von einem Benutzer zu einer Zeit eröffnet werden. Aus mechanischen Gründen muß der Einzelformulareinzug auf der linken Druckerseite betrieben werden.
Alle Drucker können als logisches Gerät über $LPT betrieben werden. Dies wurde durch die Druckerzuordnungstabelle, die dem logischen Driver $LPT in Abhängigkeit vom Arbeitsplatz einen physikalischen Drucker zuordnet, möglich.

Ein-/Ausgabe-Programmierung

Die Synchronisation zwischen Zentraleinheit und dem RAP ist bezüglich der Druckerdaten auftragsbezogen. Werden in BASIC nacheinander mehrere Felder auf die gleiche Zeile ausgegeben, so führt das zu extremer Druckverlangsamung. Deshalb ist unbedingt ZEILENWEISE auszugeben.

Eröffnen von Druckerdateien

Vor der Datenausgabe auf einem Drucker muß dieser auf einem beliebigen Kanal mit der Anweisung

- OPEN #

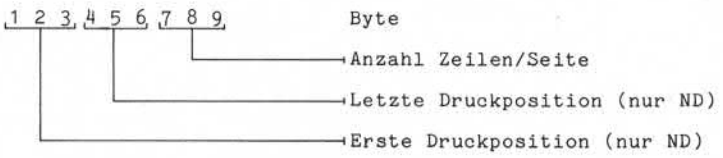
eröffnet werden.

Als Dateiname ist der Name des jeweiligen Drucker-Kanalprogrammes anzugeben:

- $LPT = logischerDruckername oder Nadeldrucker, linker Vorschub
- $LPTR = Nadeldrucker, rechter Vorschub
- $LPT1 = Zeilendrucker
- $LPTS = 2. Nadeldrucker, linker Vorschub
- $LPTRS = 2.Nadeldrucker, rechter Vorschub
- $RPLn = ND an Kanal n, linker Vorschub
- $RPRn = ND an Kanal n, rechter Vorschub
- $RPFn = ND an Kanal n, Formulareinzug

Zusätzlich können in der OPEN # - Anweisung noch Parameter zur Formularsteuerung vorgegeben werden. Diese Parameter werden in einer String-Variablen, die für mindestens 9 Byte dimensioniert sind, an das Drucker-Kanalprogramm übergeben.

Diese Variable hat folgenden Aufbau:



Beim Einsatz eines Zeilendruckers werden die erste und letzte Druckposition nicht ausgewertet. Die letzte Druckposition muß immer größer sein als die erste.
Bei Nadeldruckern mit zwei Vorschubaggregaten dürfen sich die Druckbereiche nicht überlappen.

Werden keine Angaben zur Formularsteuerung gemacht, gelten folgende Standardwerte:

Zeilendrucker:	Erste Druckposition	= 000
	Letzte Druckposition	= 131
	Formularhöhe	= 048
Nadeldrucker, linker Vorschub:		
	Erste Druckposition	= 000
	Letzte Druckposition	= 176
	Formularhöhe	= 048
Nadeldrucker, rechter Vorschub:		
	Erste Druckposition	= 177
	Letzte Druckposition	= 177
	Formularhöhe	= 048

Ausgabe auf einem Drucker

Die Ausgabe am Drucker erfolgt mit den Anweisungen:
- PRINT #
- WRITE #

Es dürfen nur String-Literale und/oder der Inhalt von String-Variablen ausgegeben werden.

Als Besonderheiten sind zu beachten:

- Falls <N-EXPR2> und <N-EXPR3> codiert sind, werden diese Angaben ignoriert.
- In einer WRITE # - Anweisung dürfen keine numerischen Ausdrücke und keine numerischen Variablen angegeben sein.
- Die unterschiedlichen Funktionen der Trennzeichen ";" und "," sind bei der Anweisung PRINT # unbedingt zu beachten.

Formularsteuerung

Die Formularsteuerung erfolgt mit Hilfe von Formularsteuerzeichen. Zur Verfügung stehen:

- CR = Carriage return (oktal 215)
- LF = Line feed (oktal 212)
- TF = Top of form (oktal 214)
- FF = Form feed (oktal 214)
- VT = Vert. tabulation (oktal 213)

die oktal im auszugebenden String (Variable, Literal) vorgegeben werden müssen.

Ein-/Ausgabe-Programmierung

Steuerzeichen	Funktion
CR	Zeilenvorschub
LF	Zeilenvorschub oder Nadelkopf nach rechts fahren, Schacht öffnen
TF	Seitenwechsel
FF	Formular-Auswurf
VT	Formular auf Formularkopf einziehen

Ein Zeilenvorschub wird automatisch dann durchgeführt, wenn:

- PRINT # ohne abschließendes Semikolon zur Ausführung kommt,
- mehr Zeichen ohne Steuercode ausgegeben werden, als eine Druckzeile Zeichen aufnehmen kann (Bereich zwischen erster und letzter Druckposition).

Die Behandlung von Textdateien ist weitgehend identisch mit der Behandlung von Druckerdateien, d.h. es kann, falls Druckausgaben gemacht werden und der Drucker belegt ist, der auszugebende Datenbestand in einer Textdatei zwischengespeichert werden.

Ein-/Ausgabe-Programmierung

CHF

Mit der Funktion CHF besteht für Druckerdateien die Möglichkeit, die aktuelle Druckzeile festzustellen. Als <N-EXPR> ist die Kanalnummer des Kanals vorzugeben, auf dem die Druckerdatei eröffnet ist.

Beispiel: Vorschub auf die nächste Seite, wenn die Zeilennummer gleich der Formularhöhe -4 ist. Die Formularhöhe ist in der Variablen F abgestellt.

```
.
PRINT #C;A$;        /*DRUCKAUSGABE
IF CHF(C) = F-4 PRINT #C;"<-214<-";
/*SEITENW.
.
```

Beispiel für das Zwischenspeichern einer Druckdatei in einer Textdatei:

```
1000 IF ERR 0 GOTO 9000   /*FEHLERBEHANDLUNG
1010 OPEN #3;P$,"$LPT"    /*DRUCKERERÖFFNUNG
.
.                         /*VERARBEITUNG
.
1100 PRINT #3;R$          /*DRUCKAUSGABE
1110 GOTO 1020
.
.
9000 IF ERR 0 GOTO 9100   /*FEHLER BEI BUILD
9010 BUILD #3,+D$         /*DATEINAME IN D$
9020 IF ERR 0 GOSUB       /*STANDARD FEHLERBEH
9030 GOTO 1020            /*AUFSETZE NACH OPEN
.
.
9100 LET D$(LEN D$+1)="!" /*DATEI ERSETZEN
9110 GOTO 9010
.
```

Anmerkung:

Allerdings ist zu beachten, daß die Auswirkung der Funktion "CHF" bei Textdateien nicht identisch ist mit der bei Druckerdateien.

Ein Beispiel für die Ausgabe von Textdateien auf einem Drucker ist unter Punkt 7.3.8.1 beschrieben.

Ein-/Ausgabe-Programmierung

7.5 Der Lochkartenleser

An das System kann ein Lochkartenleser mit einer Leistung von 500 Karten/min oder 90 Karten/min angeschlossen werden.

7.5.1 Zugriff auf Lochkartendateien

Alle über den Lochkartenleser eingelesenen Daten sind im Rahmen von Dateien organisiert. Eine Lochkartenleserdatei besteht aus einer beliebigen Anzahl von Datenblöcken mit einheitlicher Länge. Ein Block bzw. ein Datensatz entspricht in der Regel der Informationslänge der verwendeten Lochkarten = 80 Byte.
Die einzig mögliche Organisationsart ist sequentiell.
Der Lochkartenleser kann nicht gleichzeitig von mehreren Teilnehmern oder von einem Teilnehmer auf mehreren Kanälen eröffnet werden.
Die Umwandlung des Standard Lochkartencodes in ASCII-Code wird intern vom IOCS vorgenommen.

Ein-/Ausgabe-Programmierung

7.5.2 Die Programmierung

Eröffnen von Lochkartenleserdateien

Bevor Daten über den Lochkartenleser gelesen werden, muß dieser auf einem beliebigen Kanal mit der Anweisung

- OPEN #

eröffnet werden.

Als Dateiname ist der Name des Lochkartenleser-Kanal-Programmes

- $CRD

anzugeben.

Eingaben über Lochkartenleser

Die Eingabe von Daten über den Lochkartenleser erfolgt mit der Anweisung

- READ #

Die angegebenen Zielvariablen müssen Strings sein. Sind mehrere Zielvariablen angegeben, werden soviele Karten gelesen wie Variablen codiert sind.
Die Übertragung wird beendet, wenn das Ende der Zielvariablen erreicht ist.

Zum Schließen des Kanals, auf dem der Kartenleser eröffnet ist, dient die Anweisung:

- CLOSE #

Ein-/Ausgabe-Programmierung

CHF

Die Anwendung der Funktion auf Lochkartenleserdateien bleibt ohne Ergebnis.

Beispiel: Übernahme einer Lochkartenleserdatei in eine relative Datei auf einer Platte. Als Endekriterium für die Lochkartendatei ist eine /* -Karte vorgelegt.

```
.
9000 OPEN #3,"$CRD"        /*LOCHKARTENDATEI
9010 OPEN #4,D$            /*RELATIVE DATEI
9020 READ #3,R$            /*LOCHKARTE LESEN
9030 IF R$="/*" GOTO 9100 /*DATEIENDE
9040 WRITE #4;R$           /*DATENSATZ SCHREIBE
9050 GOTO 9020             /*NÄCHSTE KARTE
.
.
9100 CLOSE #3              /*LOCHKARTENDATEI
9110 CLOSE #4              /*RELATIVE DATEI
9120 PRINT "PROGRAMMENDE";
9130 CHAIN ""              /*CHAIN SCOPE
```

Ein-/Ausgabe-Programmierung

7.6 Das Magnetband

Das Magnetband kann zum Datenaustausch mit anderen Systemen und zur Datensicherung eingesetzt werden.

7.6.1 Zugriff auf Magnetbanddateien

Alle auf einem Magnetband gespeicherten Daten werden im Rahmen von Dateien organisiert. Die einzig mögliche Organisationsform ist sequentiell. Die Verwaltung der Dateien muß vom Anwenderprogramm durchgeführt werden.
Eine Magnetbandstation kann nicht von mehreren Teilnehmern gleichzeitig eröffnet werden.

Für die Programmierung des Magnetbandes steht die Anweiung

- CALL 70

zur Verfügung.

"CALL 70" repräsentiert ein physikalisches IOCS und bildet die Schnittstelle zwischen dem Basic-Interpreter "RUN" und dem Kanalprogramm $MTX.

Syntax: Syntaktisch werden 4 Formate unterschieden:

Format 1:
<CALL> ::= CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>

Format 2:
<CALL> ::= CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,
<N-VAR4>,<N-VAR5>

Format 3:
<CALL> ::= CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,
<N-VAR4>,<N-VAR5>,<S-VAR1>

Format 4:
<CALL> ::= CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,
<S-VAR2>,<S-VAR3>

Ein-/Ausgabe-Programmierung

Funktion:

CALL = Anweisung

<N-EXPR> = Nummer des aufzurufenden Unterprogrammes, muß in diesem Fall 70 sein.

<N-VAR1> = Numerische Variable, in der die Nummer der anzusprechenden Bandstation vorgegeben werden muß.
<N-VAR1> muß immer = 0 sein!

<N-VAR2> = Numerische Variable, in der die Nummer der durchzuführenden Funktion vorgegeben werden muß.
<N-VAR2> muß im Bereich von 1 bis 14 liegen.

<N-VAR3> = Numerische Variable, in der nach der Durchführung der Funktion ein Statuscode übergeben wird.
<N-VAR3> muß vor Ausführung der Funktion = 0 sein.

<N-VAR4> = In Abhängigkeit von der Funktion die Anzahl zu verarbeitender Dateien, Blöcke oder Byte.

<N-VAR5> = In Abhängigkeit von der Funktion, die Anzahl verarbeitender Dateien, Blöcke oder Byte.

<S-VAR1> = String-Variable zur Aufnahme des gelesenen oder zu schreibenden Blocks.

<S-VAR2> = String-Variable zur Aufnahme der Eingabe-Codetabelle. Diese Tabelle muß vom Anwenderprogramm geladen werden.
<S-VAR2> muß 256 Byte lang sein!

<S-VAR3> = String-Variable zur Aufnahme der Ausgabe-Codetabelle. Diese Tabelle muß vom Anwenderprogramm geladen werden.
<S-VAR3> muß 256 Byte lang sein!

Das jeweilige Format ist von der durchzuführenden Funktion abhängig.

Ein-/Ausgabe-Programmierung

Format 1:	CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>
<N-VAR2>	Funktion
1	Band auf BOT (begin of tape) zurückspulen.
2	Bandgerät "OFF LINE" schalten.
6	Bandmarke schreiben.
7	Gap setzen (2,5 Zoll Bandlücke).
10	Bandmarke schreiben und Band auf BOT zurückspulen.
11	Bandgerät für die Verarbeitung ohne Codeumwandlung eröffnen.
12	Bandgerät für Verarbeitung ohne Codeumwandlung eröffnen und Band auf BOT zurückspulen.
13	Bandgerät schließen.
14	Bandgerät "OFF LINE" schalten und schließen.

Format 2:	CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>, <N-VAR4>,<N-VAR5>
<N-VAR2>	Funktion
3	Band um die in <N-VAR4> vorgegebene Anzahl Blöcke vorsetzen.
4	Band um die in <N-VAR4> vorgegebene Anzahl Blöcke zurücksetzen.
8	Band um die in <N-VAR4> vorgegebene Anzahl Bandmarken vorsetzen.
9	Band um die in <N-VAR4> vorgegebene Anzahl Bandmarken zurücksetzen.

Anmerkung: Die Anzahl Blöcke/Bandmarken wird in <N-VAR4> vorgegeben. <N-VAR5> 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 <N-VAR3> ein Statuscode übergeben.

Ein-/Ausgabe-Programmierung

Format 3:	CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>, <N-VAR4>,<N-VAR5>,<S-VAR1>
<N-VAR2>	Funktion
0	Block lesen. Die Übertragung wird beendet wenn die dimensionierte Länge von <S-VAR1> oder der in <N-VAR4> angegebenen Anzahl Zeichen erreicht ist.
5	Block schreiben. Die Übertragung wird beendet wenn die dimensionierte Länge von <S-VAR1> oder der in <N-VAR4> vorgegebenen Anzahl Zeichen erreicht ist.
Anmerkung:	Nach Ausführung der Funktion wird in <N-VAR5> die tatsächlich gelesene Blocklänge, bzw. die Anzahl tatsächlich geschriebener Zeichen übergeben. Dieser Wert muß nicht mit dem in <N-VAR4> vorgegebenen Wert übereinstimmen.

Format 4:	CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>, <S-VAR2>,<S-VAR3>
<N-VAR2>	Funktion
11	Magnetbandstation zur Verarbeitung mit Codewandlung eröffnen. Die Angabe <S-VAR3> muß nur bei der Ausgabe auf Band angegeben sein.
12	Wie <N-VAR2> = 11, jedoch mit Zurückspulen de Bandes auf BOT.
Anmerkung:	Die Umwandlungstabellen müssen vom Basic-Programm in <S-VAR2> und <S-VAR3> zur Verfügung gestellt werden und dürfen, nachdem das Band eröffnet wurde, nicht mehr verändert werden. Folgende Standard-Codetabellen stehen in relativen Dateien auf einer Magnetplatte 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 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.

Statuscodes

Nach Ausführung einer Magnetband-Operation ist unbedingt der Status in <N-VAR3> zu überprüfen.
Folgende Werte können in <N-VAR3> durch das Unterprogramm übergeben werden:

<N-VAR3>	Bedeutung
0	Operation erfolgreich
1	Bandgerät nicht verfügbar (bereits eröffnet)
2	Bandgerät nicht bereit (off line)
3	Band ist schreibgeschützt (kein Schreibring)
4	Schreib/Lesefehler
5	EOF (End of file) Bandmarke gelesen
6	EOT (End of tape) Bandende-Spiegel
7	Unzulässige Operation

Beispiel: Ausgabe einer relativen Datei auf Magnetband.
Folgende Variablen werden benutzt:

```
U    =  Gerätenummer Magnetband
F    =  Funktion
S    =  Status
C    =  Block-/Satzlänge
A    =  Aktuelle (geschriebene) Länge
R$   =  Datensatz
D$   =  Name Plattendatei
```

Die Ausgabe erfolgt ohne Codewandlung.

Ein-/Ausgabe-Programmierung

```
10   IF ERR 0 GOSUB 330
20   DIM D$(20)
30   PRINT 'CS'; TAB (0,24);"RELATIVE DATEI AUF MB
     KONVERTIEREN"
40   INPUT 'CR',"DATEINAME:", TAB (30)D$ /*RELATIVE DATEI
50   INPUT 'CR',"SATZLÄNGE:", TAB (30)C
60   IF C>2048 GOTO 50                   /*MAX. SL = 2048
70   DIM R$(C)                           /*DIM MIT SATZLÄNGE
80   OPEN #2,D$                          /*OPEN REL. DATEI
90   LET F=12                            /*FUNKTION LADEN
100  CALL 70,U,F,S                       /*OPEN UND REWIND
110  IF S GOTO 220                       /*FEHLER BEI OPEN
120  LET F=6                             /*FUNKTION LADEN
125  CALL 70,U,F,S                       /*BANDM. SCHREIBEN
130  IF S GOTO 200                       /*FEHLER
140  LET F=5                             /*FUNKTION LADEN
150  READ #2;R$;                         /*PLATTENSATZ LESEN
160  CALL 70,U,F,S,C,A,R$                /*BLOCK A.BAND SCHR
170  IF S GOTO 200                       /*FEHL.BEI SCHREIB.
180  LET R$=" ",R$                       /*STRING LÖSCHEN
190  GOTO 150                            /*NÄCHSTEN PLATTENS
200  LET S1=S                            /*FEHLERSCHLÜSSEL
                                           SICHERN
210  GOSUB 250                           /*BAND SCHLIESSEN
220  PRINT 'CR';"BANDFEHLER = ";S
230  CLOSE #2                            /*CLOSE RELATIVE
                                           DATEI
240  CHAIN ""                            /*CHAIN NACH SCOPE
250  LET F=10                            /*FUNKTION LADEN
260  LET S=0                             /*STATUS AUF 0
270  CALL 70,U,F,S                       /*BANDMARKE SCHR.
280  LET F=13                            /*FUNKTION LADEN
290  LET S=0                             /*STATUS AUF 0
300  CALL 70,U,F,S                       /*BANDGERÄT SCHL.
310  LET S=S1                            /*STATUS ZURÜCK
320  RETURN
330  IF SPC 8>97 RETURN -1               /*ESC ODER FALSCH-
                                           EINGABE
340  IF SPC 10<110 GOTO 360              /*NOCH KEIN BAND-
                                           OPEN
350  GOSUB 250
360  IF SPC 8=51 GOTO 400                /*DATEIENDE?
370  PRINT 'CR';"BASIC-FEHLER # "; SPC 8;" ZEILE # ";
          SPC 10
380  IF SPC 10<90 CHAIN ""               /*NOCH KEIN OPEN
390  GOTO 230
400  PRINT 'CR';"DATEIENDE"
410  GOTO 230
```

8 Inter-Task Kommunikation

In Systemen, an denen mehrere Teilnehmer gleichzeitig arbeiten können, muß es möglich sein, Daten oder Nachrichten von Programmen der einzelnen Teilnehmer (Tasks), während der Verarbeitung auszutauschen.

Basic bietet folgende Möglichkeiten:

SIGNAL 1 - Übergabe von numerischen Werten an einen beliebigen Teilnehmer.

SIGNAL 2 - Übernahme von numerischen Werten, die mit "SIGNAL 1" übergeben wurden.

CALL 2 - Übergabe des Inhalts von beliebigen Variablen im gemeinsamen Bereich (Common Area) eines beliebigen Teilnehmers.

CALL 3 - Übernahme des Inhalts des gemeinsamen Bereiches (Common Area) eines beliebigen Teilnehmers in beliebiger Variable.

CALL 4 - Ausgabe einer Nachricht auf dem Bildschirm des Masterplatzes.

8.1 Kommunikation mit SIGNAL-Anweisungen

Die Anweisungen

- SIGNAL 1 (Senden) und
- SIGNAL 2 (Empfangen)

ermöglichen den Austausch von numerischen Werten zwischen beliebigen Teilnehmern.
Das System verfügt über eine Signal-Liste, in der die mit der Anweisung "SIGNAL 1" übermittelten Werte, die Nummern der sendenden Arbeitsplätze und die Nummern der adressierten Arbeitsplätze abgestellt werden. Standardmäßig faßt diese Liste vier Einträge.

Ein Eintrag in der Signal-Liste wird auch dann vorgenommen, wenn an der Tastatur die Tasten CTL B gedrückt werden. Dadurch besteht für den Bediener die Möglichkeit, dem am betreffenden Arbeitsplatz laufenden Programm ein "Signal" zu übermitteln.

Inter Task-Kommunikation

8.1.1 Die Anweisung SIGNAL 1

Überstellen von drei numerischen Werten in die Signal-Liste.

Aufbau:

<SIGNAL 1> ::= SIGNAL 1,<N-EXPR1>,<N-EXPR2>,<N-EXPR3>

Funktion:

SIGNAL 1 = Anweisung.

<N-EXPR1> = BA-Nummer des adressierten Teilnehmers.

<N-EXPR2>
<N-EXPR3> = In die Signal-Liste zu überstellende numerische Werte.
Es darf kein Wert > 32767 übergeben werden.

Die Werte von <N-EXPR1>, <N-EXPR2>, <N-EXPR3> und die Nummer des sendenden Arbeitsplatzes werden in die Signal-Liste überstellt.
Die Werte bleiben solange in der Signal-Liste, bis vom adressierten Teilnehmer eine Anweisung "SIGNAL 2" ausgeführt wird.
Ist ein Teilnehmer angegeben, der nicht konfiguriert ist, oder sind in der Signal-Liste bereits vier Einträge vorhanden, wird Basic-Fehler # 62.

SIGNALPUFFER VOLL/PORT-NR. NICHT VORHANDEN

gemeldet.
Ein Eintrag in der Signal-Liste wird gestrichen, wenn er ca. 2 Stunden in der Liste steht, ohne daß eine SIGNAL 2-Anweisung ausgeführt wurde.

Beispiel: Überstellen der Werte 0 und 10 in die Signal-Liste. Adressiert wird der Masterplatz.

```
.
.
LET P=0          /*ADRESSIERTER ARBEITSPLATZ
LET X=0          /*<N-EXPR2>
LET Y=10         /*<N-EXPR3>
SIGNAL 1,P,X,Y
```

Inter Task-Kommunikation

8.1.2 Die Anweisung SIGNAL 2

Übertragen (empfangen) numerischer Werte aus der Signal-Liste in numerische Variable.

Aufbau:

<SIGNAL 2> ::= SIGNAL 2,<N-VAR1>,<N-VAR2>,<N-VAR3>,
[<N-EXPR>]

Funktion:

SIGNAL 2 = Anweisung

<N-VAR1> = Enthält nach der Ausführung der Anweisung die BA-Nummer des Teilnehmers, von dem das "Signal" in die Signal-Liste überstellt wurde.
Liegt kein Signal vor, enthält <N-VAR1> den Wert -1.

<N-VAR2> <N-VAR3> = Enthalten nach der Ausführung der Anweisung die mittels "SIGNAL 1" in die Signal-Liste überstellten numerischen Werte.
Liegt kein Signal vor, bleiben diese Variablen unverändert.

<N-EXPR> = Ein Wert von 1 bis 32767.
Es kann ein Zeitraum in Zehntelsekunden angegeben werden, für den auf die Übermittlung eines "Signals" gewartet werden kann.
Wird ein Signal empfangen, bevor die Zeit abgelaufen ist, läuft das Programm weiter.

Beispiel: Ein Teilnehmer wartet auf ein Signal vom Masterplatz. Signale anderer Arbeitsplätze werden ignoriert. Das Programm läuft erst weiter, wenn das Signal empfangen wurde.

```
.
100 SIGNAL 2,T,X,Y,100  /*10 SEKUNDEN WARTEN
110 IF T<>0 GOTO 100    /*KEIN SIGNAL VOM
                          MASTER
```

8

Inter Task-Kommunikation

8.2 Kommunikation über den gemeinsamen Bereich (Common Area)

Jeder angeschlossene (konfigurierte) Arbeitsplatz verfügt über einen gemeinsamen Bereich in Größe von 512 Byte. Dieser Bereich liegt außerhalb der Partition, in der die Task abläuft, und wird durch Segmentwechsel (CHAIN) nicht verändert.
Der Bereich wird genutzt:

- zum Datenaustausch zwischen verschiedenen Teilnehmern.
- zur Übergabe von Daten an ein nachfolgendes mit einer der Anweisungen "CHAIN" oder "LINK" aufgerufenes Segment.

Jeder Teilnehmer kann Daten aus dem gemeinsamen Bereich eines beliebigen anderen Teilnehmers lesen und Daten in den gemeinsamen Bereich eines beliebigen Teilnehmers schreiben.

Zur Durchführung dieser Funktion stehen die Anweisungen:

- CALL 2 = Schreiben in den gemeinsamen Bereich
- CALL 3 = Lesen aus dem gemeinsamen Bereich

zur Verfügung.

8.2.1 Die Anweisung CALL 2

Transport des Inhaltes von bis zu 11 Variablen (<S-VAR>, <N-VAR>) in den gemeinsamen Bereich eines beliebigen Teilnehmers.

Aufbau:

Funktion:

CALL = Anweisung.

Inter Task-Kommunikation

<N-EXPR> = Die Nummer des aufzurufenden Unterprogrammes (In diesem Fall = 2).

<N-VAR>
<S-VAR> = 1 bis 12 Variable.
Die erste codierte Variable muß <N-VAR> sein und die BA-Nummer des Teilnehmers enthalten, in dessen gemeinsamen Bereich die Inhalte der im folgenden codierten Variablen übertragen werden soll.
Die BA-Nummer des adressierten Teilnehmers wird nicht übertragen.
Es kann also der Inhalt von max. 11 Variablen übertragen werden.
Die Adressierung von Sub-Strings (z.B. A$(X,Y)) ist nicht möglich.
Bei Vektoren und Matrizen ist die Adressierung einzelner Elemente möglich.

Die dimensionierte Gesamtlänge aller zu übertragenden Variablen darf 512 Byte nicht überschreiten.

Beispiel: Übergabe des Inhaltes der Variablen B und A$ in den gemeinsamen Bereich der Phantom-Task (BA-Nr. = 1).

```
REM   DATEN AN PHANTOMTASK ÜBERGEBEN
LET   T=1
CALL  2,T,B,A$
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Die dimensionierte Gesamtlänge der zu übertragenden Variablen ist > 512 Byte. - Die angegebene BA-Nummer ist nicht gültig.

8

Inter Task-Kommunikation

8.2.2 Die Anweisung CALL 3

Transport von Daten aus dem gemeinsamen Bereich eines beliebigen Teilnehmers in bis zu 11 Variablen.

Aufbau:

<CALL> ::= CALL <N-EXPR> { <N-VAR> } [, { <S-VAR> / <N-VAR> }] 11 1

Funktion:

CALL = Anweisung.

<N-EXPR> = Nummer des aufzurufenden Unterprogrammes (In diesem Fall = 3).

<N-VAR>
<S-VAR> = 1 bis 12 Variable.
Die erste codierte Variable muß <N-VAR> sein und die BA-Nummer des Teilnehmers enthalten, aus dessen gemeinsamen Bereich Daten übernommen werden sollen.
Die Daten werden 1:1 übertragen.
Stimmen die angegebenen Variablen in Typ und/oder Länge nicht mit den zu übernehmenden Daten überein, ist das Ergebnis undefiniert. Die Adressierung von Sub-Strings (z.B. A$(X,Y)) ist nicht möglich.
Bei Vektoren und Matrizen ist die Adressierung einzelner Elemente möglich.

Die dimensionierte Gesamtlänge aller aufnehmenden Variablen darf 512 Byte nicht überschreiten.

Beispiel: Übernahme von Daten aus dem gemeinsamen Bereich der Phantom-Task in die Variable B und A$.

```
REM   DATEN VON PHANTOMTASK ÜBERNEHMEN
LET T=1
CALL 3,T,B,A$
```

Inter Task-Kommunikation

Fehler:

Basic-Fehler	Bedeutung
# 38	- Die dimensionierte Gesamtlänge der aufnehmenden Variablen ist > 512 Byte. - Die angegebene BA-Nummer ist nicht gültig.

Inter Task-Kommunikation

8.3 Ausgabe von Nachrichten am Masterplatz (CALL 4)

Um jedem Teilnehmer (jeder Task) zu ermöglichen, Nachrichten auf dem Bildschirm des Masterplatzes auszugeben, stellt Business-Basic die Anweisung

```
CALL 4
```

zur Verfügung.

Aufbau:

<CALL> ::= CALL <N-EXPR>,<S-VAR>

CALL	= Anweisung.
<N-EXPR>	= Nummer des aufzurufenden Unterprogrammes, muß in diesem Fall = 4 sein.
<S-VAR>	= String, dessen Inhalt auf dem Bildschirm des Masterplatzes ausgegeben werden soll.

Diese Anweisung wird nur dann erfolgreich ausgeführt, wenn der Masterplatz im "INPUT"-Modus steht und noch keine Zeichen eingegeben wurden.
Jeder andere Zustand des Masterplatzes bewirkt den Basic-Fehler # 38 bei der Ausführung des "CALL 4". Die Anweisung ist deshalb so oft zu wiederholen, bis sie erfolgreich ausgeführt werden kann!
Da die Ausgabe auf dem Bildschirm des Masterplatzes ab der aktuellen Cursorposition erfolgt, sollte <S-VAR> geeignete Display-Funktionen (z.B. TAB) vor der eigentlichen Nachricht enthalten, um ein Zerstören des aktuellen Bildschirminhaltes am Masterplatz zu vermeiden.
Die Bildschirmpositionen, die auf dem Masterplatz für solche Meldungen benutzt werden können, sind vom Anwender zu definieren. Lediglich beim Einsatz von TAMOS sind in der Zeile 24 die Positionen 66 bis 78 reserviert.

Inter Task-Kommunikation

Beispiel: Ausgabe einer Meldung auf Position 15 in Zeile 24 (Nachrichtenzeile) am Masterplatz. Die Ausgabe wird beliebig oft wiederholt, bis die Anweisung korrekt ausgeführt ist. Die Routine ist als Unterprogramm aufgebaut.

```
100  DIM M$(50)                  /* VARIABLE ZUR AUF-
                                    NAHME DER NACHRICHT
.
.
5000 LET M$="←207←←376←←211←←376←←221←←217←
            ←230←MELDUNG←376←←212←"
5010 IF ERR 0 GOTO 5050          /*FEHLER # 38 ??
5020 CALL 4,M$
5030 IF ERR 0 GOSUB 9000         /*STANDARD-FEHLERBEHANDLUNG
5040 RETURN                      /*RÜCKSPRUNG INS HAUPTPROGR.
5050 IF SPC 8 <> 38 GOTO 1010 /*PROGRAMMABBRUCH
5060 SIGNAL 3,100                /*10 SEKUNDEN WARTEN
5070 GOTO 5020                   /*ERNEUTER VERSUCH
```

FEHLER:	Basic-Fehler	Bedeutung
	# 38	- Der Masterplatz befindet sich nicht im Input-Modus.

8

Funktionen von Basic

9 Funktionen von Basic

Basic stellt dem Programmierer eine Reihe von Funktionen zur Verfügung (<N-FUNC>), die folgenden Aufbau haben:

```
<N-FUNC> ::= [ { <U-FUNC> } (<N-EXPR>)                          ]
               { <F-NAME> }
             [                                                  ]
             [ TAB (<N-EXPR> [,<N-EXPR>] )                      ]
             [ LEN (<S-VAR>)                                    ]
```

Definition: Funktionen, mit Ausnahme der E/A - unterstützenden TAB-Funktion, liefern numerische Funktionswerte. Es werden Anwenderfunktionen und im Basic-System standardmäßig eingebaute Funktionen unterschieden.
Die Angabe der Klammern kann entfallen, wenn <N-EXPR> ein nicht zusammengesezter numerischer Ausdruck ist. Ebenso können bei der LEN-Funktion die Klammern entfallen. Bei der Funktion TAB sind Klammern zwingend vorgeschrieben.

Aufbau von Anwenderfunktionen "<U-FUNC>"

```
<U-FUNC>    ::= FN <BU> (<N-EXPR>)
```

Vom System zur Verfügung gestellte Funktionen

```
<F-NAME>    ::= | SIN | COS | TAN | ATN | SQR | LOG | EXP |
                | ABS | SGN | INT | FRA | RND | NOT | MAN |
                | CHR | IXR | SPC | DET | CHF |
```

Funktionen von Basic

9.1 Mathematische Funktionen

9.1.1 Trigonometrische Funktionen

Die zur Verfügung stehenden trigonometrischen Funktionen sind:

- SIN (Sinus)
- COS (Cosinus)
- TAN (Tangens)
- ATN (Arcustangens)

Der Wert von <N-EXPR> für die Sinus-, Cosinus- bzw. Tangensfunktion ist ein in Radianten ausgedrückter Winkel. Ist der Winkel in Graden angegeben, muß er mit den Formeln:

```
  o
X   * 3,1415926535898/180 oder

  o
X   / 57,295779513085
```

in Radianten umgewandelt werden.
Der Wert von <N-EXPR> für die Arcustangens-Funktion kann jede reelle Zahl sein (der Tangens jedes Winkels). Das Ergebnis ist dann ein in Radianten angegebener Winkel, im Bereich von +- 1/2 π.
Ein Beispiel für diese Anwendung ist im Anhang aufgeführt.

Funktionen von Basic

9.1.2 Transzendente Funktionen

Die zur Verfügung stehenden transzendenten Funktionen sind:

- SQR (Quadratwurzel)
- LOG (natürlicher Logarithmus)
- EXP (Exponentialfunktion)

SQR(<N-EXPR>) ::= Quadratwurzel von <N-EXPR>

Beispiel: SQR(4) = 2 oder SQR(36) = 6

EXP(<N-EXPR>) ::= Exponentialfunktion (zur Basis e)

Beispiel: EXP 3 = e ↑ 3

e = 2,71828182.. (Eulersche Zahl)

LOG(<N-EXPR>) ::= Natürlicher Logarithmus (zur Basis e)

Beispiel: LOG(e) = 1.

Funktionen von Basic

9.1.3 Arithmetische Funktionen

Die zur Verfügung stehenden arithmetischen Funktionen sind:

- ABS (Absolutwert)
- SGN (Signumfunktion)
- INT (Ganzzahlenwert)
- FRA (Bruchteil)
- RND (Zufallszahl)

9.1.3.1 ABS (Absolutwert)

Die Funktion ABS (<N-EXPR>) liefert den absoluten Wert von <N-EXPR>, d.h. den Wert von <N-EXPR> mit positivem Vorzeichen.

Beispiel: LET A = ABS (A/B)

Der Absolutwert der sich aus (A/B) ergibt, wird in der Variablen A abgestellt.

9.1.3.2 SGN (Signumfunktion)

Die Funktion SGN (<N-EXPR>) liefert:

0, wenn der Wert von <N-EXPR> = 0 ist.
+1, wenn der Wert von <N-EXPR> positiv ist.
-1, wenn der Wert von <N-EXPR> negativ ist.

Beispiel: LET X = SGN (A/B)

X erhält entsprechend dem Ergebnis aus (A/B) den Wert: -1, 0 oder +1.

Funktionen von Basic

9.1.3.3 INT (Ganzzahlwert)

Die Funktion INT (<N-EXPR>) liefert die größte Ganzzahl, die den Wert von <N-EXPR> nicht überschreitet.

Beispiel:

<N-EXPR>	INT
0,5	0
-1,999	-2
1,99	1

9.1.3.4 FRA (Bruchteil)

Die Funktion FRA (<N-EXPR>) liefert den Nachkommawert und das Vorzeichen von <N-EXPR>.

Beispiel:

<N-EXPR>	FRA
2,0	0,0
+1,1234	+0,1234
-1,5	-0,5

9.1.3.5 RND (Zufallszahl)

Die Funktion RND (<N-EXPR>) liefert eine Zahl, die der Rechner aus einer Sequenz von Pseudozufallszahlen generiert. <N-EXPR> definiert den Bereich in dem die Zufallszahl gebildet werden soll.

Aufbau:

```
RND (X)
    |
    |_____größte Zahl
    |     (Obergrenze)
    |
    |_____Funktionsname
```

Beispiel: Bilden von Ganzzahlen im Bereich von +1 bis +7. Die ermittelte Zahl wird in der Variablen "A" abgestellt.

```
LET A = INT (RND(6)) +1
```

Funktionen von Basic

9.2 Logische Funktionen

Als logische Funktion steht die Funktion

NOT (logische Inversion)

zur Verfügung.

9.2.1 NOT (logische Inversion)

Diese Funktion liefert als Ergebnis:

0, wenn das Argument ungleich 0 ist
1, wenn das Argument gleich 0 ist.

Beispiele: 1) Abstellen einer 0 oder 1 in der Variablen B, abhängig vom Inhalt der Variablen A.

```
.
LET B = NOT A
.
```

2) Aufrunden des Inhaltes der Variablen B

```
.
LET B = INT B + NOT NOT FRA B
```

Funktionen von Basic

9.3 Funktionen zur Zahlenmanipulation

Es stehen die Funktionen:

- MAN (Mantisse)
- CHR (Charakteristik)
- IXR (Ganzzahl-Exponent zur Basis 10)

zur Verfügung.

Diese Funktionen ermöglichen die Zerlegung von Gleitkommazahlen in ihre Bestandteile.
Eine Gleitkommazahl wird in der Form:

```
X = a * b ↑ n
```

dargestellt.

- a = Mantisse
- b = Basis (10)
- n = Charakteristik (Exponent)

Durch die o.g. Funktionen werden die folgenden Werte geliefert:

- MAN (X) = a
- CHR (X) = n
- IXR (CHR(X)) b↑n
- IXR (Z) = b INT(Z)

"X" läßt sich mit diesen Funktionen wie folgt darstellen:

```
X = MAN (X) * IXR (CHR↑(X))
```

Funktionen von Basic

9.3.1 MAN (Mantisse)

Die Funktion "MAN" liefert die Mantisse einer beliebigen Gleitkommazahl.

MAN (X) = X/b ↑ CHR (X)

Das von "MAN" gelieferte Ergebnis ist immer eine Nachkommazahl.

Beispiel: Die Variable X enthält den Wert 57,60.
Nach Durchführung der Anweisung

LET Y = MAN (X)

enthält die Variable Y den Wert ,576.

9.3.2 CHR (Charakteristik)

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 /* FALSCHEINGABE
130 .        /* EINGABE OK
```

Funktionen von Basic

9.3.3 IXR (Ganzzahl-Exponent der Basis)

Die Funktion "IXR" liefert den Ganzzahl-Exponenten einer Basis.

Die Schreibweise ist:

```
IXR (CHR(X))
```

```
IXR (CHR(X)) = b ↑ n
```

Wird die Funktion in der Form:

```
IXR (X)
```

ausgeführt, gilt die Formel:

```
IXR (X) = b ↑ INT (X)
```

Funktionen von Basic

9.4 Anwender-Funktionen

Der Programmierer hat die Möglichkeit, bis zu 26 Funktionen zu definieren. Diese Möglichkeit findet dann Verwendung, wenn gleiche Ausdrücke (Formeln) in einem Programm mehrmals auftreten.
Definierte Funktionen können während des Programmlaufes geändert (neu definiert) werden.

9.4.1 DEF-Anweisung

Definition von Anwender-Funktionen

Aufbau:

<DEF>::= DEF FN<BU> (<N-VAR>) = <N-EXPR>

Funktion:

DEF = Anweisung.

FN = Muß angegeben werden und dient dem Basic-Interpreter "RUN" bei der Laufzeit zur Erkennung der Anwender-Funktion.

<BU> = Beliebiger Buchstabe von A bis Z. Dadurch werden die maximal 26 verschiedenen Anwender-Funktionen unterschieden.
Z.B.: FNA oder FNX.

(<N-VAR>) = Scheinvariable. Diese Variable nimmt das Ergebnis der Anwenderfunktion auf. Der angegebene Variablenname kann bereits vergeben sein. Der in der DEF-Anweisung angegebene Name dient nur zur internen Zwischenspeicherung des Funktions-Ergebnisses. Der Variablenname steht immer in Klammern und darf nur aus einem Buchstaben "<BU>" bestehen. Ist an anderer Stelle im Programm derselbe Variablenname vergeben, wird der Inhalt dieser Variablen nicht beeinflußt.

Funktionen von Basic

= = Ergibt Zeichen (siehe auch Kap.: LET)

<N-EXPR> = Beliebiger Ausdruck (Formel), der die Funktion bildet.

Definierte Anwenderfunktionen können geschachtelt werden, d.h. eine Funktion kann aus mehreren bereits definierten Funktionen bestehen. Es ist möglich, bis zu 5 bereits definierte Funktionen in einer DEF-Anweisung zu schachteln. Eine Funktion kann erst dann ausgeführt werden, wenn sie mit DEF definiert ist.

Eine definierte Funktion kann während des Programmlaufes umdefiniert werden.

Beispiel: An mehreren Stellen innerhalb eines Programmes muß der Inhalt von Variablen aufgerundet werden.

```
DEF FNX (Z) = INT Z + NOT NOT FRA Z
.
.
LET Z = FNX (A+B)
.
.
In der Variablen A steht  1,90
In der Variablen B steht  5,00

Nach der Addition steht in der Variablen Z
der Wert 7,00
```

Funktionen von Basic

9.5 Sonderfunktionen

Als Sonderfunktion stehen:

SPC	KEY
LEN	LKY
TAB	CKY
CHF	DET

zur Verfügung.

9.5.1 SPC

Die Funktion "SPC" liefert abhängig von dem unter <N-EXPR> angegebenen Wert, folgende Informationen:

<N-EXPR>	Information
0 -	Beanspruchte ZE-Zeit in Zehntelsekunden seit der Anmeldung.
1 -	Belegungszeit in Minuten seit der Anmeldung.
2 -	Vergangene Zeit in Stunden seit dem 1.1.1973.
3 -	Stundenteil in Zehntelsekunden (Ergänzung zu SPC 2).
4 -	System-Erstellungsdatum, als Differenz in Stunden zum 1.1.1973.
5 -	Konto-Nummer bestehend aus: - Privilege - Group - User des Kontos (ACCOUNT), unter dem der ausführende Arbeitsplatz angemeldet ist. Diese Informationen werden als Dezimalwerte übergeben. Die Aufschlüsselung dieses Wertes in die drei o.g. Angaben ist als Beispiel im Anschluß an diese Übersicht beschrieben.
6 -	Nummer des ausführenden Arbeitsplatzes.

Funktionen von Basic

7 -	Keine Funktion. Die Vorgabe dieser Funktionsnummer erzeugt den Basic-Fehler ≠ 29.
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.
(-105) -	Liefert die Nummer der physikalischen Plateneinheit, von der zuletzt ein IPL durchgeführt wurde. Folgende Werte sind möglich: 0 = Festplatte, Laufwerk 0 1 = Wechselplatte, Laufwerk 0 2 = Festplatte, Laufwerk 1 3 = Wechselplatte, Laufwerk 1

Anmerkung: Die Anzahl Stunden seit dem 1.1.1973 setzt voraus, daß alle Monate mit 31 Tagen gerechnet werden.

Beispiele:

1) Auswertung des durch SPC (5) gelieferten Wertes. Die Angaben:
   - Privilege
   - Group
   - User

   werden in den Variablen P, G und U abgestellt.

```
.
REM    P = PRIVILEGE
REM    G = GROUP
REM    U = USER
LET    P = INT (SPC 5/16384)
LET    G = INT ((SPC 5-P*16384)/64)
LET    U = SPC 5-P*16384-G*64
```

Funktionen von Basic

2) Auswertung von Basic-Fehlern in einer Fehlerbehandlungsroutine mit gezielter Reaktion auf die Fehler 98 und 99. Die Routine ist als Unterprogramm aufgebaut.

```
9000 REM FEHLERAUSWERTUNG
9005 IF SPC 8 > 97 GOTO 9025
9010 PRINT TAB (15,24);"BASIC-FEHLER # ";SPC 8;
     "ZEILENNUMMER : "; SPC 10;
9015 SIGNAL 3,20
9020 CHAIN A$
9025 IF SPC 8 = 99 RETURN-1
9030 PRINT TAB (15,24);"EINGABE MUSS NUMERISCH SEIN";
9035 SIGNAL 3,20
9040 PRINT 'LD';
9045 RETURN -1
```

9.5.2 LEN

Die Funktion LEN (<S-VAR>) ermittelt Grenzzeichenpositionen in String-Variablen und übergibt die jeweils aktuelle Länge. Dadurch wird z.B. die Adressierung von Strings in Abhängigkeit von ihrer aktuellen Länge ermöglicht.

Beispiel: Abstellen des Zeichens ":" hinter dem z.Zt. letzten Zeichen im String A$.

```
LET A$ (LEN A$+1) = ":"
```

Die Wirkungsweise dieser Funktion ist im Kapitel "Datendarstellung/Datenformate" (4.2.5.3) eingehend beschrieben.

Funktionen von Basic

9.5.3 TAB

Die Funktion TAB (<N-EXPR> [,<N-EXPR>]) ermöglicht:

- beliebige Cursorpositionierung am Bildschirm vorzunehmen.
- eine beliebige Anzahl von Leerzeichen (Blank) innerhalb einer Druckzeile auszugeben.

Die detaillierte Beschreibung dieser Funktion erfolgt im Kapitel "Ein-/Ausgabe-Programmierung" unter den Punkten:

- 7.1.5 für Textdateien und Drucker,
- 7.2.2.1 für Bildschirm.

9.5.4 CHF

Die Funktion CHF (<N-EXPR>) ermöglicht dem Programmierer, die aktuelle Größe einer Datei oder die augenblickliche Zugriffsposition innerhalb einer Datei zu ermitteln.
Der Wert von <N-EXPR> definiert die Kanalnummer und die durchzuführende Funktion.
Entspricht der Wert von <N-EXPR> der Kanalnummer, wird die augenblickliche Zugriffsposition in der Datei, deren Kanalnummer vorgegeben ist, übergeben.
Entspricht der Wert von <N-EXPR> der Kanalnummer + 100, wird die aktuelle Größe der Datei übergeben, die auf dem Kanal eröffnet ist und sich aus <N-EXPR> - 100 ergibt.

Unterschiede, die sich aufgrund der verschiedenen Dateiorganisationen bzw. des jeweiligen Peripheriegerätes ergeben, sind bei der Beschreibung des Zugriffs auf den entsprechenden Dateityp bzw. auf das entsprechende Peripheriegerät behandelt.

9.5.5 DET

Die Funktion DET (<N-EXPR>) liefert den Determinantenwert der zuletzt invertierten Matrix.

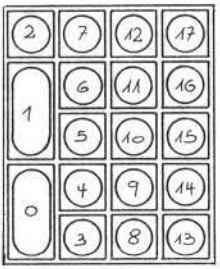
<N-EXPR> repräsentiert den Namen der zuletzt invertierten Matrix.

Funktionen von Basic

9.5.6 KEY

Die Funktion "KEY (0)" liefert den numerischen Wert der zuletzt betätigten Auslösetaste.

Die Werte für die Tasten (A0 bis A17) sind:



Beispiel:

```
100  IF KEY (0)= 4 GOTO 200
  .
  .
  .
200 REM *** A4 BETÄTIGT ***
```

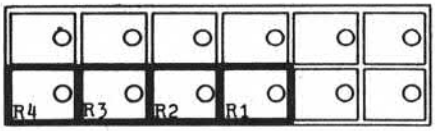
Anmerkung:

Die Tasten "Y" und "N" im numerischen Tastenfeld sind wirkungsgleich mit [Y] [CR] und [N] [CR] und erzeugen -da auch Auslösetasten- wie [CR] den Wert 0.

Beim Betätigen von Tasten, die keine Auslösetasten sind, bleibt KEY (0) ohne Auswirkung.

9.5.7 LKY (lock key)

Die Funktion "LKY (i)" liefert den Wert der gesetzten Rasttaste (-n):




```
LKY (0)  -  liefert die Summe der Werte aller gesetzten
            Rasttasten.
LKY (1)  -  liefert den Wert der Rasttaste   'R1'
LKY (2)  -                  -                'R2'
LKY (3)  -                  -                'R3'
LKY (4)  -                  -                'R4'
```

Die einzelnen Werte der Tasten sind:

```
 R1  -  256   (2↑8)
 R2  -  512   (2↑9)
 R3  -  1024  (2↑10)
 R4  -  2048  (2↑11)
```

Beispiel:

```
100 IF LKY (0) = 1536 GOTO 200
  .
  .
200 REM *** R2, R3  GESETZT ***
  .
  .
500 IF LKY (4) = 2048 GOTO 600
  .
  .
600 REM *** R4 GESETZT ***
```

Funktionen von BASIC

9.5.8 CKY (clear key)

Die Funktion "CKY (0)" löscht alle gesetzten Rasttasten und setzt die Rasttastenwerte = 0.

Die Funktion ist ausschließlich in den Anweisungen "PRINT" und "INPUT" (analog den TAB-Funktionen) erlaubt.

Beispiel:

```
100  INPUT CKY (0),A
  .
  .
```

oder

```
100 PRINT CKY (0)
  .
  .
```

Beschreibung der Anweisungen

10 Beschreibung der Anweisungen

Zur Beschreibung sind die Anweisungen in 5 Gruppen:

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

gegliedert.

10.1 Deklaration von Datenfeldern und Funktionen

Beschreibung der Anweisungen:

- DIM
- DEF
- RANDOM
- DEALLO

Beschreibung der Anweisungen

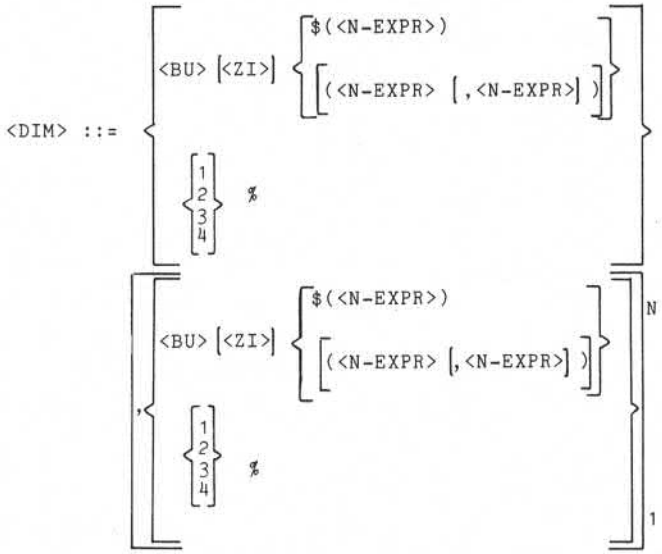
10.1.1 "DIM"-Anweisung

Reservierung von Speicherbereich für:

- Numerische Variablen
- Vektoren
- Matrizen
- Strings

Mit der DIM-Anweisung wird zusätzlich zur Reservierung des Speicherbereiches für jeder Variable ein Name festgelegt, unter dem sie während des Programmablaufes angesprochen werden kann.

Syntax:



Beschreibung der Anweisungen

Funktion:

DIM = Anweisung.

<BU> [<ZI>] = Variablenname, bestehend aus einem Buchstaben von A bis Z und wahlweise einer zusätzlichen Ziffer von 0 bis 9.

$(<N-EXPR>) = Nur für Strings. Kennzeichen und Länge in Byte. $ bedeutet, daß die zu dimensionierende Variable ein String ist. In Klammern muß die Anzahl Byte angegeben werden, die der String aufnehmen soll.

(<N-EXPR> [,<N-EXPR>]) = Nur für die Dimensionierung von Vektoren und Matrizen erforderlich.
Mit der Angabe (<N-EXPR>) wird ein Vektor dimensioniert. "<N-EXPR>" repräsentiert die Anzahl Elemente des Vektors.
Mit der Angabe (<N-EXPR>,<N-EXPR>) wird eine Matrix dimensioniert. "(<N-EXPR>,<N-EXPR>)" repräsentiert die Anzahl Reihen und die Anzahl Spalten der Matrix.

{1 2 3 4} % = Formatangabe für Numerische Variablen, Vektor- und Matrix-Elemente.
Diese Angabe legt die Größe in Worten für alle bis zu einer erneuten "%"-Angabe dimensionierten

- numerischen Variablen
- Elemente von Vektoren
- Elemente von Matrizen

fest.
Ist keine "%"-Angabe gemacht, gilt die in einer vorher durchlaufenen "DIM"-Anweisung definierte "%"-Angabe. Falls im Programmablauf noch kein Format festgelegt wurde, gilt als Standardwert 2%.

Beschreibung der Anweisungen

Dimensionierung von numerischen Variablen

Es muß nur der Variablenname und ggf. eine Formatangabe (1-4 %) vor dem Variablenname gemacht werden.

Beispiel: DIM 1%,A,B1,2%,X

Die Variablen A und B1 werden als 1-Wort Variable und X als 2-Wort Variable dimensioniert. Die Formatangabe (2%) bleibt so lange wirksam, bis sie durch eine erneute Formatangabe aufgehoben bzw. geändert wird.

Dimensionierung von Vektoren

Bei der Dimensionierung von Vektoren ist die Angabe der Anzahl Elemente, die der Vektor haben soll, erforderlich. Dies wird durch die Angabe der Nummer des letzten Elements des Vektors erreicht. Die Elemente eines Vektors werden von 0 bis n durchnumeriert. Das bedeutet, daß die Anzahl der Elemente eines Vektors der Nummer des letzten Elements +1 entspricht.

Beispiel: DIM 1%,A,A1(5),2%,A2(5)

A wird als numerische 1-Wort Variable dimensioniert. Der Vektor A1 wird mit 6 1-Wort Elementen und der Vektor A2 mit 6 2-Wort Elementen dimensioniert.

Dimensionierung von Matrizen

Bei der Dimensionierung von Matrizen ist die Angabe der Anzahl Reihen und Spalten erforderlich. Dies wird durch Angabe der Nummer der letzten Reihe und der letzten Spalte der Matrix erreicht. Die Numerierung der Reihen und Spalten läuft von 0 bis n. Das bedeutet, daß die Anzahl Reihen/Spalten der angegebenen Nummer +1 entsprechen. Die Anzahl Elemente einer Matrix errechnet sich aus:

Anzahl Reihen * Anzahl Spalten

Beispiel: DIM 4%,A(5,5)

Dimensionierung der Matrix A mit 6 Reihen und 6 Spalten. Die Elemente der Matrix haben 4-Wort Format.

Beschreibung der Anweisungen

Dimensionierung von Strings

Bei der Dimensionierung von Strings ist die Anzahl Bytes vozugeben, die der Aufnahme von Anwender-Daten dienen sollen. Intern werden ein oder zwei zusätzliche Bytes zur Aufnahme eines Grenzzeichens reserviert.

Beispiel: DIM A$(10)

Dimensionierung des Strings A$ in einer Länge von 10 Byte.

Zum Zeitpunkt der Deklaration wird der jeweilige Variablen-Name von "RUN" in eine Variablen-Liste eingetragen. Diese Liste kann die Namen von maximal 93 Variablen aufnehmen.

Nach durchgeführter Dimensionierung enthalten numerische Variablen sowie die Elemente von Vektoren und Matrizen den Wert "0". Strings sind mit Grenzzeichen gefüllt (Binär 0).

Redimensionierung von Variablen

Die Redimensionierung von Variablen ist lediglich nach der Durchführung einer DEALLO-Anweisung erlaubt. Eine Ausnahme besteht bei matrizen und Vektoren. Bei ihnen ist eine Redimensionierung ohne DEALLO möglich sofern sich die Anzahl der Elemente durch die Redimensionierung nicht vergrößert.

Beschreibung der Anweisungen

10.1.2 "DEF"-Anweisung

Definition von Anwender-Funktionen

Syntax:

<DEF> :: = DEF FN<BU> (<N-VAR>) = <N-EXPR>

Funktion:

DEF	= Anweisung.
FN	= Muß angegeben werden und dient dem Basic-Interpreter "RUN" bei der Laufzeit zur Erkennung der Anwender-Funktion.
<BU>	= Beliebiger Buchstabe von A bis Z, wodurch die maximal 26 verschiedenen Anwender-Funktionen unterschieden werden. Z.B.: FNA oder FNX
(<N-VAR>)	= Scheinvariable. Diese Variable nimmt das Ergebnis der Anwenderfunktion auf. Der vorgegebene Variablenname kann bereits vergeben sein. Der in der DEF-Anweisung angegebene Variablenname dient nur zur internen Zwischenspeicherung des Funktions-Ergebnisses. Der Variablenname steht immer in Klammern und darf nur aus einem Buchstaben "<BU>" bestehen. Ist an anderer Stelle im Programm derselbe Variablenname vergeben, wird der Inhalt dieser Variablen nicht beeinflußt.
=	= Zuordnungssymbol "=". Weist <N-VAR> den Wert von <N-EXPR> zu.
<N-EXPR>	= Beliebiger Ausdruck (Formel), der die Funktion bildet.

Beschreibung der Anweisungen

Definierte Anwenderfunktionen können geschachtelt werden, d.h. eine Funktion kann aus mehreren bereits definierten Funktionen gebildet werden. Es besteht die Möglichkeit, bis zu 5 bereits definierte Funktionen in einer DEF-Anweisung zu schachteln. Eine Funktion kann erst dann ausgeführt werden, wenn sie mit DEF definiert ist.
Eine definierte Funktion kann während des Programmlaufes umdefiniert werden.

Beispiel:

```
An mehreren Stellen innerhalb eines
Programmes muß der Inhalt von Variablen
aufgerundet werden.
.
DEF FNX (Z) = INT Z + NOT NOT FRA Z
.
.
LET Z = FNX (A+B)
.
.
In der Variablen A steht 1,90
In der Variablen B steht 5,00

Nach der Ausführung der LET-Anweisung
steht in der Variablen Z der Wert 7,00.
```

10.1.3 "RANDOM"-Anweisung

Kontrolle der durch die Funktion "RND" generierten Folge von Zufallszahlen.
Bei Arbeiten mit Zufallszahlen entsteht grundsätzlich das nicht vorhersehbare Problem, wie sich das Programm bei verschiedenen, den Programmablauf beeinflussenden Werten verhalten wird. Aus diesem Grunde besteht die Möglichkeit die Folge der Zufallszahlen mit der "RANDOM"-Anweisung zu steuern.

Syntax:

<RANDOM> :: = RANDOM <N-EXPR>

Funktion:

RANDOM = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck, der zur Steuerung der Zufallszahlen-Folge dient.

Die Wirkung auf die Funktion "RND" ist abhängig vom Wert in <N-EXPR>.

<N-EXPR>	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. Unterschiedliche Werte <> 0 führen zu verschiedenen Folgen von Zufallszahlen. Gleiche Werte bewirken die Generierung der gleichen Folgen. RANDOM 1 bewirkt zum Beispiel immer die gleiche Folge von Zufallszahlen.

Die Funktion "RND" (ohne "RANDOM") ist in ihrer Wirkungsweise identisch mit "RANDOM 0".

10.1.4 DEALLO - Anweisung

Mit dieser Anweisung können beliebige, in einem Programm (-Segment) angegebenen, Namen von Numerischen- und/oder String-Variablen und der ihnen zugewiesene Speicherplatz zur erneuten Verwendung freigegeben werden.

Die angegebenen Variablen können anderweitig zugewiesen und erneut dimensioniert werden.

Syntax:

<DEALLO> ::= DEALLO [<N-VAR> | <M-VAR> | <S-VAR>] [, [<N-VAR> | <M-VAR> | <S-VAR>]]$_{1}^{63}$

Funktion:

DEALLO = Anweisung.

<N-VAR>
<S-VAR>
<M-VAR> = String- und/oder numerische-, bzw. Matrix-Variablen. Maximal sind 63 Variablennamen erlaubt.

Anmerkung:

Die Ausführungszeit der Anweisung ist vom jeweiligen Standort der betreffenden Variablen in der Partition abhängig, da die Freigabe eine Lücke in dem sonst geschlossenen Programmbereich hinterläßt, die anschließend wieder geschlossen werden muß.

Um die Ausführungszeit dieser Anweisung zu minimieren, ist zu empfehlen, die Variablennamen in der DEALLO-Anweisung, in umgekehrter Reihenfolge wie in der DIM-Anweisung, anzugeben.

Beispiel:

```
100 DIM A$(100),1%,A,B,B$(200)
    .
    .
500 DEALLO B$,B,A,A$
```

Beschreibung der Anweisungen

Fehler:

Sofern Fehler bei der Anweisung auftreten, wird der Vorgang bei der verursachenden Variable abgebrochen.
Diese, und die in der Anweisung folgenden Variablen wernicht mehr bearbeitet.

10.2 Datenmanipulation und Arithmetik

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

Im einzelnen sind dies:

- LET
- MAT
- DATA
- READ
- RESTOR

10.2.1 "LET"-Anweisung

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

Das Zeichen "=" ist zu interpretieren als:

....."erhält den Wert von"...

Im Einzelnen übernimmt "LET" die Funktionen:

- Durchführung arithmetischer Operationen
- Zuweisung von Werten zu Variablen

Wenn erforderlich werden die Daten rechts des Gleichheitszeichens in das Format der Variablen links des Gleichheitszeichens (aufnehmende Variable) umgewandelt.

Syntax:

<LET>::= LET [<S-VAR>= { <S-EXPR> / <N-EXPR> [USING { <S-VAR> / <S-LIT> }] } / <N-VAR>= { <N-EXPR> / <S-VAR> / <S-LIT> }]

Beschreibung der Anweisungen

Funktion:

LET = Anweisung. Bei der Eingabe von Anweisungen kann das Schlüsselwort "LET" entfallen.

String-Variable als Ziel-Variable

<S-VAR>= = Beliebige String-Variable und das Zuweisungszeichen "=". Diese Variable dient als Ziel-Variable.

<S-EXPR> = Beliebiger String-Ausdruck. Die Angabe von <S-EXPR> ist nur möglich, wenn die Ziel-Variable eine String-Variable ist.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert von <N-EXPR> wird intern in ASCII-Code umgewandelt.
Die Umwandlung erfolgt, falls sie nicht mittels "USING" über eine Maske erfolgt, in der Form:

- Vorzeichen (1 Byte), für "+" wird ein Leerzeichen und für "-" ein Minus-Zeichen im Ziel-String abgestellt.
- Im Anschluß an das Vorzeichen werden die signifikanten Ziffern übertragen. Führende Nullen werden unterdrückt.
- Im Anschluß an die zuletzt übertragene Ziffer wird ein Leerzeichen übertragen.

Ist der Wert von <N-EXPR> = 0, wird eine Null übertragen.

Bei Gleitkommazahlen ist besonders zu beachten, daß ohne Maskenangabe ("USING") die Übertragung im Gleitkommaformat erfolgt, wenn:

- Die Anzahl der signifikanten Ziffern 14 überschreitet.
- Ein Wert kleiner 0,1 entsteht.

Beschreibung der Anweisungen

USING [<S-LIT> / <S-VAR>] = Mit dieser Angabe wird eine Aufbereitungsmaske vorgegeben, die die Aufbereitung des zu übertragenden Wertes steuert. Die Maske selbst wird in <S-LIT> oder <S-VAR> vorgegeben.

Die Angabe "USING" darf in einer "LET"-Anweisung nur einmal vorkommen.
Der Aufbau von Masken und die Funktionen der zur Verfügung stehenden Masken-Steuerzeichen ist unter Pkt. 4.4 beschrieben.

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

- Erreichen der dimensionierten Anzahl Byte im Ziel-String.
- Erreichen des Grenzzeichens im Quell-String bzw. nach der Übertragung des letzten Zeichens eines "String-Literals".
- Erreichen des Endekriteriums im letzten Element, wenn als Quelle "<S-EXPR>" dient.
- Nach der Übertragung des abschließenden Leerzeichens bzw. der letzten Ziffer (bei "USING") des Wertes eines numerischen Ausdrucks.

Beschreibung der Anweisungen

Numerische Variable als Ziel-Variable

<N-VAR>= = Beliebige numerische Variable, Vektor-Element oder Matrix-Element und das Zuweisungszeichen "=". Diese Variable dient als Ziel-Variable.

<N-EXPR> = Beliebiger numerischer Ausdruck, numerische Variable, Vektor-Element, Matrix-Element, String-Variable oder String-Literal.
<S-VAR>
<S-LIT>
Der Wert dieser Angabe wird, wenn er numerisch ist, nach <N-VAR> übertragen. Ist als Quelle eine String-Variable oder ein String-Literal angegeben, wird solange übertragen, bis das Ende oder ein nicht numerisches Zeichen auftritt. Sind nur nichtnumerische Zeichen vorhanden, erhält <N-VAR> den Wert 0. In diesem Fall erfolgt keine Fehlermeldung.

Anmerkung: Die Darstellung von Daten, die Formate der verschiedenen Variablentypen, die Adressierung von Variablen sowie der Aufbau von Konstanten und Masken sind im Abschnitt "4" dieses Handbuches ausführlich erläutert.

Beispiele: String-Variable als Ziel-Variable

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

```
M$ = DATEI NICHT GEFUNDEN

.
1000 LET A$="STATUS: ",M$
.
```

A$ nach der Übertragung:

```
STATUS:  DATEI NICHT GEFUNDEN
```

2) Füllen der Variablen M$, die für 50 Byte dimensioniert ist, mit:
....5....0....5...

```
50 DIM M$(50)
.
1000 M$="....5....0", M$
.
```

M$ nach der Übertragung:

....5....0....5....0....5....0....5....0....5....0....5....0....5

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.

3) Alle Byte von M$ werden mit Leerzeichen gefüllt.

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

Die Übertragung erfolgt nach dem gleichen Schema wie bei Beispiel 2.

4) Übertragen des Ergebnisses aus:

A * B

in den String M$, beginnend an der aktuellen Grenzzeichenposition. Die Aufbereitung der Zeichen wird über eine Maske gesteuert.

Beschreibung der Anweisungen

```
A = 5
B = 100
M$ =        |S|U|M|M|E|:| |Z|
                           G
.
1000 LET M$(LEN M$+1) = A*B USING "#####"
```

M$ nach der Übertragung:

```
|S|U|M|M|E|:| | | |5|0|0|Z|
                         G
```

Numerische Variable als Ziel-Variable.

1) Verdoppelung des Wertes von "A"

```
.
1000 LET A= A+A
.
```

2) Übertragen des Inhaltes der Stellen 50 bis 54 des Strings M$ in die Variable A.

```
M$(50,54) = |1|2|3|4|A|

.
1000 LET A= M$(50,54)
.
```

A nach der Übertragung = 1234.

Beschreibung der Anweisungen

10.2.2 "MAT"-Anweisung

Die "MAT"-Anweisung ermöglicht die Bearbeitung von Matrizen in ihrer Gesamtheit.
Matrizen sind zweidimensionale Datenfelder, bestehend aus:

1 bis n Reihen und 1 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.

Aufbau einer Matrix, bestehend aus 5 Reihen und 5 Spalten:

Spalte 0-4

Reihe 0-4	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
4	4,0	4,1	4,2	4,3	4,4

Reihe 0 und Spalte 0 werden nicht bearbeitet.

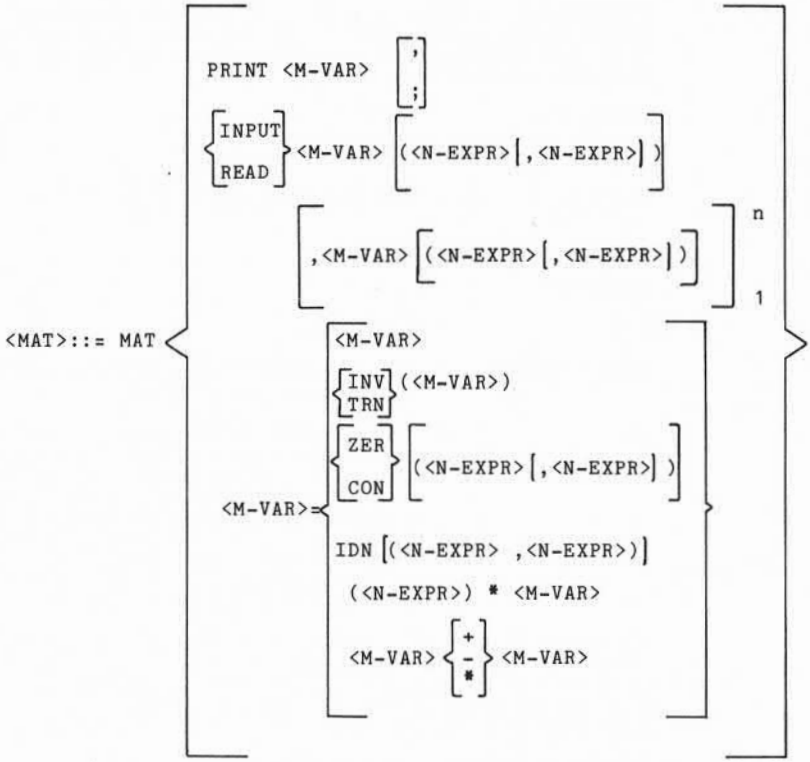
Beschreibung der Anweisungen

Im Einzelnen stellt die "MAT"-Anweisung die folgenden Funktionen zur Verfügung:

- Funktion "PRINT"
- Funktion "INPUT"
- Funktion "READ"
- Funktion "INV"
- Funktion "TRN"
- Funktion "ZER"
- Funktion "CON"
- Funktion "IDN"
- Matrix - Zuordnung
- Matrix - Addition
- Matrix - Subtraktion
- Matrix - Multiplikation
- Matrix - Multiplikation mit Skalaren

Beschreibung der Anweisungen

Syntax:



Beschreibung der Anweisungen

10.2.2.1 Funktion "PRINT"

Ausgabe von Matrizen in ihrer Gesamtheit ohne Reihe 0 und Spalte 0 auf dem Bildschirm ab der aktuellen Cursorposition. Die Ausgabe erfolgt reihenweise und innerhalb der Reihen von links nach rechts.

Syntax:

<MAT> :: = MAT PRINT <M-VAR> [,;]

Funktion:

MAT = Anweisung.

PRINT <M-VAR> = Funktionsbezeichnung "PRINT" und Name der auszugebenden Matrix.

[,;] = Trennzeichen. Diese Angabe steuert das Format der Ausgabe. Komma und Semikolon haben die gleiche Bedeutung wie bei der Anweisung "PRINT". Das abschließende Semikolon oder Komma wird als Trennzeichen zwischen den Elementen der Matrix interpretiert. Ist keines der beiden Zeichen angegeben, hat das dieselbe Wirkung wie die Angabe eines ",".

Mit einer Anweisung kann nur eine Matrix ausgegeben werden. Für die Aufbereitung des Inhalts der einzelnen Elemente, die ausgegeben werden sollen, gelten die gleichen Regeln wie bei der Anweisung "PRINT".
Eine maskierte Ausgabe über "USING" ist nicht möglich.

Beispiel: Ausgabe der Matrix "A".

```
.
MAT PRINT A;
.
.
```

Beschreibung der Anweisungen

10.2.2.2 Funktion "INPUT"

Eingabe in Matrizen. Mit einer Anweisung können Eingaben in alle Elemente einer Matrix - und zwar für jedes Element getrennt - gemacht werden.
Die Eingabe erfolgt reihenweise und innerhalb der Reihen von links nach rechts. Die Eingaben sind durch Komma zu trennen. Nachdem alle Elemente einer Reihe eingegeben sind, muß die Eingabe mit der Taste "CR" ausgelöst werden.

Syntax:

<MAT> ::= MAT INPUT <M-VAR>[(N-EXPR>[,<N-EXPR>])][,<M-VAR>[(<N-EXPR>[,<N-EXPR>])]]$_1^n$

Funktion:

MAT = Anweisung.

INPUT = Funktionsbezeichnung "INPUT".

<M-VAR> = Matrix-Variable, in die eingegeben werden soll.

(<N-EXPR>[,<N-EXPR>]) = Angabe zur Dimensionierung der Matrix. Diese Angabe wird dann gemacht, wenn die unter <M-VAR> angegebene Matrix noch nicht dimensioniert ist bzw. eine neue Dimension erhalten soll.
Ist nur (<N-EXPR>) angegeben, wird die Matrix mit der Anzahl <N-EXPR> Reihen und 2 Spalten (incl. Spalte 0) dimensioniert. Ist (<N-EXPR>,<N-EXPR>) angegeben, muß die Anzahl Reihen und die Anzahl Spalten vorgegeben sein. Siehe auch Pkt. 4.2.4.2 (Dimensionierung von Vektoren und Matrizen). Wird eine bestehende Matrix neu dimensioniert, darf sich die Anzahl der Elemente nicht vergrößern!

Beschreibung der Anweisungen

Beispiel: Eingabe von Werten in die Matrix "A". Die Matrix wird in dieser Anweisung mit 2 Reihen und 4 Spalten dimensioniert.

```
MAT INPUT A(2,4)
```

10.2.2.3 Funktion "READ"

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

Syntax:

<MAT> :: = MAT READ <M-VAR> [(<N-EXPR> [,<N-EXPR>])]

Funktion:

MAT = Anweisung.

READ = Funktionsbezeichnung "READ".

<M-VAR> = Matrix-Variable, in welche die Konstanten übertragen werden sollen.

(<N-EXPR> [,<N-EXPR>]) = Angabe zur Dimensionierung der Matrix. Diese Angabe wird dann gemacht, wenn die unter <M-VAR> angegebene Matrix noch nicht dimensioniert ist bzw. eine neue Dimension erhalten soll.

Ist nur (<N-EXPR>) angegeben, wird die Matrix mit der Anzahl <N-EXPR> Reihen und 2 Spalten (incl. Spalte 0) dimensioniert. Ist (<N-EXPR>,<N-EXPR>) angegeben, muß die Anzahl Reihen und die Anzahl Spalten vorgegeben sein. Siehe auch Pkt. 4.2.4.2 (Dimensionierung von Vektoren und Matrizen).

Wird eine bestehende Matrix neu dimensioniert, darf sich die Anzahl der Elemente nicht vergrößern.

Beschreibung der Anweisungen

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 3 Reihen und 3 Spalten (excl. 0) dimensioniert. Die Matrix "B" wird für die Anzahl Reihen und Spalten dimensioniert, die in den Variablen X und Y vorgegeben sind.

```
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
100 MAT READ A,B(X,Y)          /* X = 2   Y = 4
```

Matrix "A"

5	5	5
8	8	8
3	3	3

Matrix "B"

1	2	3	4
5	6	7	8

10.2.2.4 Funktion "INV"

Die Funktion "INV" invertiert eine vorgegebene Matrix. Das Produkt der definierten Matrix (<M-VAR2>) und der zu erstellenden inversen Matrix (<M-VAR1>) ergibt die Einheitsmatrix (siehe auch Pkt. 10.2.2.8, MAT IDN).

Syntax:

<MAT> :: = MAT <M-VAR1> = INV (<M-VAR2>)

Funktion:

MAT = Anweisung.

<M-VAR1> = Die zu erstellende, zu <M-VAR2> inverse Matrix.
<M-VAR1> kann = <M-VAR2> sein.
<M-VAR1> muß nicht dimensioniert sein.
Ist <M-VAR1> dimensioniert, muß deren Anzahl Elemente mindestens gleich der Anzahl Elemente von <M-VAR2> sein. Ist dies nicht der Fall, wird Basic-Fehler = 23 gemeldet.
Nach der Inversion hat <M-VAR1> die gleiche Dimension wie <M-VAR2>.

=INV = Zuordnungssymbol "=" und Funktionsbezeichnung "INV".

(<M-VAR2>) = Diese Angabe muß in Klammern stehen und bezeichnet die Matrix, zu der die inverse "<M-VAR1>" erstellt werden soll. Diese Matrix muß quadratisch sein.

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

Beschreibung der Anweisungen

Beispiel: Das Beispiel berechnet von einer Matrix mit 2 Reihen und zwei Spalten die zugehörige Inverse und das Produkt der beiden Matrizen.

```
.
MAT B= INV(A)
MAT C= A*B
.
.
```

Matrix "A"

1	2
3	4

Matrix "B"

-2	1
1,5	-,5

Matrix "C"

1	0
0	1

10.2.2.5 Funktion "TRN"

Erstellen einer zu <M-VAR2> transponierten Matrix. Die zu <M-VAR2> transponierte Matrix wird gebildet, indem Reihen und Spalten vertauscht werden.

Syntax:

<MAT> :: = MAT <M-VAR1> = TRN (<M-VAR2>)

Funktion:

MAT = Anweisung

<M-VAR1> = Die zu erstellende Matrix, in die <M-VAR2> transponiert wird. <M-VAR1> darf nicht = <M-VAR2> sein. <M-VAR1> muß nicht dimensioniert sein. Ist <M-VAR1> dimensioniert, muß die Anzahl Elemente mindestens gleich der Anzahl Elemente von <M-VAR2> sein. Ist dies nicht der Fall, wird Basic-Fehler = 23 gemeldet.

Die transponierte Matrix <M-VAR1> hat die gleiche Dimension wie <M-VAR2> bei vertauschter Anzahl Reihen/Spalten.

= TRN = Zuordnungssymbol "=" und Funktionsbezeichnung "TRN".

(<M-VAR2>) = Matrix, zu der die Transponierte gebildet wird. Diese Angabe muß in Klammern stehen!

Beispiel: Als Beispiel wird die Transponierte der Matrix "A" gebildet. "A" besteht aus 2 Reihen und 3 Spalten.

```
.
MAT B= TRN(A)
.
.
```

Matrix "A" (2,3)

1	2	3
4	5	6

Matrix "B" (3,2)

1	4
2	5
3	6

10.2.2.6 Funktion "ZER"

Alle Elemente der angegebenen Matrix (<M-VAR>) werden auf "0" gesetzt. Eine bestehende Matrix kann eine andere Dimension erhalten, bzw. eine nicht dimensionierte Matrix kann in dieser Anweisung dimensioniert werden.

Syntax:

<MAT> ::=MAT <M-VAR> = ZER [(<N-EXPR>[,<N-EXPR>])]

Funktion:

MAT = Anweisung.

<M-VAR> = Matrix-Variable. Sämtliche Elemente dieser Matrix erhalten den Wert "0". <M-VAR> muß nicht dimensioniert sein.

= ZER = Zuordnungssymbol "=" und Funktionsbezeichnung "ZER".

(<N-EXPR> ,<N-EXPR>) = Angabe zur Dimensionierung der Matrix. Diese Angabe wird dann gemacht, wenn die unter <M-VAR> angegebene Matrix noch nicht dimensioniert ist bzw. eine neue Dimension erhalten soll.

Ist nur (<N-EXPR>) angegeben, wird die Matrix mit der Anzahl <N-EXPR> Reihen und 2 Spalten (incl. Spalte 0) dimensioniert. Ist (<N-EXPR>,<N-EXPR>) angegeben, muß die Anzahl Reihen und die Anzahl Spalten vorgegeben sein. Siehe auch Pkt. 4.2.4.2 (Dimensionierung von Vektoren und Matrizen).

Wird eine bestehende Matrix neu dimensioniert, darf sich die Anzahl der Elemente nicht vergrößern!

Beschreibung der Anweisungen

Beispiel: Die Elemente der Matrix "A" mit 3 Reihen und 3 Spalten werden auf 0 gesetzt. Die Matrix wird neu dimensioniert für 4 Reihen und 2 Spalten.

```
.
MAT A= ZER (4,2)
.
```

Matrix "A", vorher

1	2	3
4	5	6
7	8	9

Matrix "A", nachher

0	0
0	0
0	0
0	0

10.2.2.7 Funktion "CON"

Alle Elemente der angegebenen Matrix (<M-VAR>) werden auf "1" gesetzt. Eine bestehende Matrix kann eine andere Dimension erhalten.
Ist die angegebene Matrix nicht dimensioniert, kann sie in dieser Anweisung dimensioniert werden.

Syntax:

<MAT> ::= MAT <M-VAR> = CON [(<N-EXPR> [,<N-EXPR>])]

Funktion:

MAT = Anweisung.

<M-VAR> = Matrix-Variable. Sämtliche Elemente dieser Matrix erhalten den Wert "1". <M-VAR> muß nicht dimensioniert sein.

= CON = Zuordnungssymbol "=" und Funktionsbezeichnung "CON".

(<N-EXPR> [,<N-EXPR>]) = Angabe zur Dimensionierung der Matrix. Diese Angabe wird dann gemacht, wenn die unter <M-VAR> angegebene Matrix noch nicht dimensioniert ist bzw. eine neue Dimension erhalten soll.
Ist nur (<N-EXPR>) angegeben, wird die Matrix mit der Anzahl <N-EXPR> Reihen und 2 Spalten (incl. Spalte 0) dimensioniert. Ist (<N-EXPR>,<N-EXPR>) angegeben, muß die Anzahl Reihen und die Anzahl Spalten vorgegeben sein. Siehe auch Pkt. 4.2.4.2 (Dimensionierung von Vektoren und Matrizen).

Wird eine bestehende Matrix neu dimensioniert, darf sich die Anzahl der Elemente nicht vergrößern.

Beschreibung der Anweisungen

Beispiel: Die Elemente der Matrix "A" mit 3 Reihen und 3 Spalten werden auf 0 gesetzt. Die Matrix wird neu dimensioniert für 4 Reihen und 2 Spalten.

```
.
MAT A= CON (4,2)
.
```

Matrix "A", vorher

1	2	3
4	5	6
7	8	9

Matrix "A", nachher

1	1
1	1
1	1
1	1

Beschreibung der Anweisungen

10.2.2.8 Funktion "IDN"

Erstellen einer Einheitsmatrix. Die Elemente, welche die Hauptdiagonale bilden, haben den Wert "1". Sämtliche anderen Elemente haben den Wert "0".
Die Einheitsmatrix ist immer quadratisch!

Syntax:

<MAT> :: = MAT <M-VAR> = IDN [(<N-EXPR>,<N-EXPR>)]

Funktion:

MAT = Anweisung.

<M-VAR> = Matrix-Variable. Die Elemente dieser Matrix, welche die Hauptdiagonale bilden, erhalten den Wert "1". Sämtliche anderen Elemente erhalten den Wert "0". <M-VAR> muß nicht dimensioniert sein.

= IDN = Zuordnungssymbol "=" und Funktionsbezeichnung "IDN".

(<N-EXPR>,<N-EXPR>) = Angabe zur Dimensionierung der Matrix. Diese Angabe ist dann erforderlich, wenn die unter <M-VAR> angegebene Matrix noch nicht dimensioniert ist bzw. eine neue Dimension erhalten soll.
<M-VAR> sollte als quadratische Matrix dimensioniert werden (sein).
Ist dies nicht der Fall, erfolgt keine Fehlermeldung, das Ergebnis ist jedoch undefiniert.

Wird eine bestehende Matrix neu dimensioniert, darf sich die Anzahl der Elemente nicht vergrößern.

Beschreibung der Anweisungen

Beispiel: Die Matrix "A" mit 3 Reihen und 3 Spalten wird eine Einheitsmatrix.

```
.
MAT A= IDN
.
```

Matrix "A", vorher

1	2	3
4	5	6
7	8	9

Matrix "A", nachher

1	0	0
0	1	0
0	0	1

10.2.2.9 Matrix-Zuordnung

Alle Elemente der in "<M-VAR1>" angegebenen Matrix erhalten den Wert der entsprechenden Elemente von Matrix "<M-VAR2>". Die Matrix, deren Elementen die Werte zugewiesen werden, muß nicht dimensioniert sein. Eine bestehende Matrix kann eine andere Dimension erhalten, allerdings darf die Anzahl der Elemente nicht vergrößert werden.
"<M-VAR1>" hat nach der Zuweisung die gleiche Dimension wie "<M-VAR2>".

Syntax:

<MAT> :: = MAT <M-VAR1> = <M-VAR2>

Funktion:

MAT = Anweisung.

<M-VAR1> = Matrix-Variable. Sämtlichen Elementen dieser Matrix wird der Wert der entsprechenden Elemente von <M-VAR2> zugewiesen.

<M-VAR1> muß noch nicht dimensioniert sein. Ist sie bereits dimensioniert, muß die Anzahl der Elemente gleich oder größer als die Anzahl Elemente von <M-VAR2> sein.
Nach der Zuweisung hat <M-VAR1> die gleiche Dimension wie <M-VAR2>.

= = Zuordnungssymbol "="

<M-VAR2> = Matrix-Variable. Der Wert sämtlicher Elemente dieser Matrix wird in die entsprechenden Elemente von <M-VAR1> übertragen.

Beschreibung der Anweisungen

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

```
.
MAT B= A
.
```

Matrix "A"

1	2	3
4	5	6
7	8	9

Matrix "B" vorher

0	0	0
0	0	0
0	0	0
0	0	0

Matrix "B" nachher

1	2	3
4	5	6
7	8	9

10.2.2.10 Matrix-Addition

Addition sämtlicher Elemente zweier Matrizen. Das Ergebnis der Addition kann einer beliebigen Matrix zugewiesen werden.

Syntax:

<MAT> :: = MAT <M-VAR1> = <M-VAR2> + <M-VAR3>

Funktion:

MAT = Anweisung.

<M-VAR1> = Matrix-Variable. Den Elementen dieser Matrix wird das Ergebnis der Addition von "<M-VAR2>" und "<M-VAR3>" zugewiesen.
<M-VAR1> kann identisch mit <M-VAR2> und/oder <M-VAR3> sein.
Ist <M-VAR1> nicht dimensioniert, erhält es die gleiche Dimension wie <M-VAR2> und <M-VAR3>. Ist <M-VAR1> bereits dimensioniert, aber nicht mit <M-VAR2> und/oder <M-VAR3> identisch, muß die Anzahl der Elemente gleich oder größer als die Anzahl von <M-VAR2> und <M-VAR3> sein.

= = Zuordnungssymbol "=".

<M-VAR2>+<M-VAR3> = Zwei Matrixvariablen werden durch ein "+"-Zeichen verknüpft. Diese beiden Matrizen werden Element für Element addiert. Das Ergebnis dieser Addition wird in den entsprechenden Elementen von <M-VAR1> abgestellt.
<M-VAR2> und <M-VAR3> müssen unbedingt gleich dimensioniert sein. <M-VAR2> und <M-VAR3> können dieselbe Matrix sein.

Beschreibung der Anweisungen

Beispiel: Addition der Matrix "A" in sich selbst. Die Matrix ist für 3 Reihen und 3 Spalten dimensioniert:

```
.
MAT A= A+A
.
```

Matrix "A", vorher

1	2	3
4	5	6
7	8	9

Matrix "A", nachher

2	4	6
8	10	12
14	16	18

10.2.2.11 Matrix-Subtraktion

Subtraktion sämtlicher Elemente zweier Matrizen. Das Ergebnis der Subtraktion kann einer beliebigen Matrix zugewiesen werden.

Syntax:

<MAT> :: = MAT <M-VAR1> = <M-VAR2> - <M-VAR3>

Funktion:

MAT = Anweisung.

<M-VAR1> = Matrix-Variable. Den Elementen dieser Matrix wird das Ergebnis der Subtraktion von "<M-VAR2>" minus "<M-VAR3>" zugewiesen.
<M-VAR1> kann identisch sein mit <M-VAR2> und/oder <M-VAR3>.
Ist <M-VAR1> nicht dimenioniert, erhält sie die gleiche Dimension wie <M-VAR2> und <M-VAR3>. Ist <M-VAR1> bereits dimensioniert, aber nicht mit <M-VAR2> und/oder <M-VAR3> identisch, muß die Anzahl der Elemente gleich/größer sein als die Elemente von <M-VAR2> und <M-VAR3>.

= = Zuordnungssymbol "=".

<M-VAR2>-<M-VAR3> = Zwei Matrixvariablen werden durch ein "-"-Zeichen verknüpft. Diese beiden Matrizen werden Element für Element subtrahiert. Das Ergebnis dieser Subtraktion wird in den entsprechenden Elementen von <M-VAR1> abgestellt. <M-VAR2> und <M-VAR3> müssen unbedingt gleich dimensioniert sein. <M-VAR2> und <M-VAR3> können identisch sein.

Beschreibung der Anweisungen

Beispiel: Subtraktion des Wertes der Elemente der Matrix "C" von den Elementen der Matrix "B". Das Ergebnis der Subtraktion wird in der Matrix "A" abgestellt.
Matrix "B" und "C" ist mit 3 Reihen und 3 Spalten dimensioniert. Matrix A ist mit 4 Reihen und 3 Spalten dimensioniert.

```
MAT A= B-C
.
```

Matrix "B"

3	4	5
6	7	8
9	10	11

Matrix "C"

1	2	3
4	5	6
7	8	9

Matrix "A", vorher

0	0	0
0	0	0
0	0	0
0	0	0

Matrix "A", nachher

2	2	2
2	2	2
2	2	2

Beschreibung der Anweisungen

10.2.2.12 Matrix-Multiplikation

Multiplikation zweier Matrizen. Das Ergebnis dieser Multiplikation kann einer beliebigen Matrix zugewiesen werden.
Die Multiplikation von Matrizen ist keine elementweise Multiplikation der entsprechenden Matrixelemente. Die Elemente der Matrix, der das Multiplikationsergebnis zugewiesen wird, entstehen als Skalarprodukt einer Zeile von <M-VAR2> mit einer Spalte von <M-VAR3>.

Syntax:

<MAT> :: = MAT <M-VAR1> = <M-VAR2> * <M-VAR3>

Funktion:

MAT = Anweisung.

<M-VAR1> = Matrix-Variable. Den Elementen dieser Matrix wird das Ergebnis der Multiplikation von <M-VAR2> mit <M-VAR3> zugewiesen. "<M-VAR1>" darf weder mit "<M-VAR2>" noch mit "<M-VAR3>" identisch sein.
<M-VAR1> erhält durch die Multiplikation immer die folgende Dimension:

- Anzahl Reihen wie <M-VAR2>,
- Anzahl Spalten wie <M-VAR3>.

Ist <M-VAR1> bereits dimensioniert, darf sich durch die Neu-Dimensionierung die Anzahl ihrer Elemente nicht vergrößern.

= = Zuordnungssymbol "=".

<M-VAR2>*<M-VAR3> = Zwei Matrix-Variablen, werden durch ein "*"-Zeichen verknüpft. Das Ergebnis aus der Multiplikation wird in <M-VAR1> abgestellt.
Die beiden Matrizen müssen nicht quadratisch sein, jedoch muß die Anzahl der Spalten von <M-VAR2> gleich der Anzahl der Reihen von <M-VAR3> sein.

Beschreibung der Anweisungen

Beispiel: Es werden 2 Matrizen ("B" und "C") multipliziert. Das Ergebnis der Multiplikation wird in der Matrix "A" abgestellt.
Die Matrizen sind mit 3 Reihen und 3 Spalten dimensioniert.

```
.
MAT A= B*C
.
```

Matrix B			Matrix C			Matrix A		
1	2	3	1	2	3	30	36	42
4	5	6	4	5	6	66	81	96
7	8	9	7	8	9	102	126	150

Die Berechnung geschieht folgendermaßen:

```
A(1,1)= B(1,1)*C(1,1) + B(1,2)*C(2,1) + B(1,3)*C(3,1)
A(1,2)= B(1,1)*C(1,2) + B(1,2)*C(2,2) + B(1,3)*C(3,2)
A(1,3)= B(1,1)*C(1,3) + B(1,2)*C(2,3) + B(1,3)*C(3,3)

A(2,1)= B(2,1)*C(1,1) + B(2,2)*C(2,1) + B(2,3)*C(3,1)
A(2,2)= B(2,1)*C(1,2) + B(2,2)*C(2,2) + B(2,3)*C(3,2)
A(2,3)= B(2,1)*C(1,3) + B(2,2)*C(2,3) + B(2,3)*C(3,3)

A(3,1)= B(3,1)*C(1,1) + B(3,2)*C(2,1) + B(3,3)*C(3,1)
A(3,2)= B(3,1)*C(1,2) + B(3,2)*C(2,2) + B(3,3)*C(3,2)
A(3,3)= B(3,1)*C(1,3) + B(3,2)*C(2,3) + B(3,3)*C(3,3)
```

10.2.2.13 Matrix-Multiplikation mit Skalaren

Elementweise Multiplikation einer Matrix mit einem Skalarfaktor. Das Ergebnis dieser Multiplikation wird einer Matrix zugewiesen.

Syntax:

<MAT> :: = MAT <M-VAR1> = (<N-EXPR>)*<M-VAR2>

Funktion:

MAT = Anweisung.

<M-VAR1> = Matrix-Variable. Den Elementen dieser Matrix wird das Produkt der Multiplikation von <N-EXPR> mit den Elementen von <M-VAR2> zugewiesen. <M-VAR1> muß nicht dimensioniert sein. Ist <M-VAR1> bereits dimensioniert, muß die Anzahl ihrer Elemente gleich/größer der Anzahl Elemente von <M-VAR2> sein.
Nach der Multiplikation hat <M-VAR1> die gleiche Dimension wie <M-VAR2>.
<M-VAR1> und <M-VAR2> können identisch sein.

= = Zuordnungssymbol "=".

(<N-EXPR>) = Beliebiger numerischer Ausdruck mit dem die einzelnen Elemente von <M-VAR2> multipliziert werden.

* = Arithmetischer Operand: "Multiplikation".

<M-VAR2> = Matrix-Variable. Die Elemente dieser Matrix werden mit dem Wert von <N-EXPR> multipliziert.

Beschreibung der Anweisungen

Beispiel: Multiplikation der Matrix "A" (3 Reihen und 3 Spalten) mit dem Wert der Variablen "B".
"B" hat den Wert 3.

```
.
MAT A=(B)*A
.
```

Matrix "A", vorher

1	2	3
4	5	6
7	8	9

Matrix "A", nachher

3	6	9
12	15	18
21	24	27

Beschreibung der Anweisungen

10.2.3 "DATA"-Anweisung

Abstellen von numerischen Konstanten im Datenbereich des Basic-Programmes.

Syntax:

<DATA> ::= DATA [{ 1 2 3 4 } % ,] [{ + - }] <N-LIT> [, [{ + - }] <N-LIT>]n 1

Funktion:

DATA = Anweisung.

{ 1 2 3 4 } % = Formatangabe, die das Format der zu definierenden Konstanten in Worten festlegt. Diese Angabe kann pro "DATA"-Anweisung nur einmal gemacht werden und gilt nur für die Konstanten dieser einen Anweisung.
Ist keine Formatangabe gemacht, wird 2% angenommen.

+/- = Vorzeichen "+" oder "-". Kann für jede Konstante angegeben werden. Ist kein Vorzeichen angegeben, wird "+" angenommen.

<N-LIT> = Beliebige numerische Literale.

Die Interpretation der "DATA"-Anweisungen und die Übernahme der Konstanten erfolgt 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" für numerische Variablen.

Beschreibung der Anweisungen

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

```
.
DATA 1%,10,20,30
DATA 10000,20000
.
```

Beschreibung der Anweisungen

10.2.4 "READ"-Anweisung

Übernahme von Konstanten, die mit der "DATA"-Anweisung angelegt wurden, in beliebige numerische Variablen.

Syntax:

<READ> ::= READ <N-VAR> [,<N-VAR>]$_{1}^{n}$

Funktion:

READ = Anweisung.

<N-VAR> = Beliebige numerische Variable(n), Vektor- oder Matrix-Element(e), in welche die Konstante(n) übertragen werden soll(en).

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 (siehe Pkt. 10.2.5 "RESTOR").

Beispiel: Übertragen der unter Zeilennummer 30 und 40 definierten Konstanten in die Vektoren A(5) und B(5) jeweils ab Element "1".

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

Beschreibung der Anweisungen

10.2.5 "RESTOR"-Anweisung

Der intern geführte Zeiger für die mit "DATA" angelegten Konstanten wird auf die erste Konstante zurückgesetzt, bzw., wird eine Zeilennummer angegeben, steht der Zeiger auf dem ersten Datenelement, der ersten DATA-Anweisung, die nach dieser Zeilennummer auftritt.

Syntax:

<RESTOR> :: = RESTOR [<Zeilen-Nr.>]

Funktion:

RESTOR = Anweisung.

<Zeilen-Nr> = Eine Zeilennummer im Programm, die gezielt angegeben werden kann.

Eine auf Restor folgende "READ"-Anweisung beginnt die Übertragung ab der ersten Konstanten.

Beispiel: Übertragen der unter Zeilennummer 30 definierten Konstanten in Variablen. Im Anschluß daran wird der "Konstanten"-Zeiger auf die erste Konstante zurückgesetzt.

```
.
30 DATA 1,2,5,7,9
.
50 READ A0,A1,A2,A3,A4
.
70 RESTOR
```

Beschreibung der Anweisungen

10.3 Steuerung des Programmablaufes

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
- FOR
- NEXT
- CHAIN
- STOP
- END
- LINK

10.3.1 "GOTO"-Anweisung

Unbedingte Verzweigung zu der Anweisung mit der angegebenen Zeilennummer.

Syntax:

<GOTO> :: = GOTO <ZL-NR>

Funktion:

GOTO = Anweisung.

<ZL-NR> = Zeilennummer der Anweisung, zu der verzweigt werden soll.
Die Programmausführung wird mit dieser Anweisung fortgesetzt.

Beispiel: Von Zeilennummer 100 wird zu der Anweisung mit der Zeilennummer 50 verzweigt.

```
.
100 GOTO 50
.
```

10.3.2 "GOSUB"-Anweisung

Unbedingtes Verzweigen in ein Unterprogramm. Es wird die Zeilennummer der ersten auszuführenden Anweisung im Unterprogramm angegeben.

Syntax:

<GOSUB> :: = GOSUB <ZL-NR>

Funktion:

GOSUB = Anweisung.

<ZL-NR> = Zeilennummer der Anweisung, zu der verzweigt werden soll.
Die Ausführung des Programmes wird mit der Anweisung fortgesetzt, deren Zeilennummer angegeben ist.

Zusätzlich zur Verzweigung wird der Unterprogramm-Stufenzähler erhöht und die Zeilennummer der auf den "GOSUB" folgenden Anweisung als Rückkehr-Adresse gespeichert. Innerhalb eines Unterprogrammes können weitere "GOSUB"-Anweisungen gegeben werden. Es stehen 8 Unterprogramm-Stufen zur Verfügung. Das heißt, es können maximal 8 "GOSUB"-Anweisungen ohne dazwischenliegende "RETURN"-Anweisung ausgeführt werden. Ein Unterprogramm muß durch eine "RETURN"-Anweisung beendet werden.

Beispiel: Es wird in ein Unterprogramm verzweigt. Die erste auszuführende Anweisung des Unterprogrammes steht auf Zeilennummer 9000.

```
.
100 GOSUB 9000
.
```

Beschreibung der Anweisungen

10.3.3 "RETURN"-Anweisung

Rücksprung aus einem Unterprogramm. Der Rücksprung erfolgt auf die dem "GOSUB" folgende Anweisung. Durch Angabe von "<N-EXPR>" ist die Modifikation der Rückkehradresse möglich.

Syntax:

<RETURN> :: = RETURN [<N-EXPR>]

Funktion:

RETURN = Anweisung.

<N-EXPR> = Beliebiger numerischer Wert, der die Anzahl Anweisungen, um die die Rückkehradresse modifiziert werden soll, repräsentiert. Als Basis für die Ermittlung der Rückkehradresse dient die bei der Ausführung des "GOSUB" gespeicherte Zeilennummer. Es kann auf die gespeicherte Zeilennummer plus n Anweisungen oder minus n Anweisungen zurückgesprungen werden.
Ist <N-EXPR> nicht angegeben oder = 0, erfolgt der Rücksprung auf die dem "GOSUB" folgende Anweisung.

Beschreibung der Anweisungen

Beispiele: 1)

```
1000 GOSUB 9000 ─┐
>1005 ...        │
1010 ...         │
.                │
.                │
.                │
.                │
9000 ...        <┘
.
.
└9050 RETURN
```

2)

```
 >1000 GOSUB 9000 ──┐
 > 1005 ...         │
   .                │
   .                │
   9000 ...        <┘
   .
   .
  └9050 IF X=0 RETURN-1
  ─9055 RETURN
```

Beispiel 1 zeigt eine Verzweigung in ein Unterprogramm und einen Rücksprung ohne Modifikation.

In Beispiel 2 wird, wenn der Inhalt von X = 0 ist, auf die gespeicherte Rückkehradresse -1 Anweisung verzweigt, also auf den "GOSUB", der die Verzweigung in das Unterprogramm durchgeführt hat.

Beschreibung der Anweisungen

10.3.4 "ON"-Anweisung

Direkte Verzweigung zu der Anweisung, deren Zeilennummer aus einer Liste mittels einem in <N-EXPR> abgestellten Wertes ausgesucht wird.

Syntax:

<ON> ::= ON <N-EXPR> [GOTO / GOSUB] <ZL-NR> [,<ZL-NR>]$_1^n$

Funktion:

ON = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Über den Wert von <N-EXPR> wird die entsprechende Zeilennummer aus der Liste der Zeilennummern in dieser Anweisung ausgewählt.

[GOTO / GOSUB] = Anweisung zum Verzweigen. Es kann sowohl mit der Anweisung "GOTO" als auch mit der Anweisung "GOSUB" verzweigt werden.

<ZL-NR> [,<ZL-NR>] =Eine beliebige Anzahl von Zeilennummern, die durch Komma voneinander getrennt werden. Die Zeilennummer zu der verzweigt wird, wird aus der Liste, über den Wert von <N-EXPR> ausgewählt.

Anmerkung: Es wird die Ganzzahl des Wertes von <N-EXPR> ausgewertet. Ist der Wert von <N-EXPR> kleiner 1 bzw. größer als die Anzahl angegebener Zeilennummern, wird keine Verzweigung durchgeführt. In diesem Fall wird das Programm mit der auf "ON" folgenden Anweisungen fortgesetzt.

Beschreibung der Anweisungen

Beispiel: Verzweigung zu der Anweisung mit Zeilennummer:

```
2000 wenn INT(X) = 1
3000 wenn INT(X) = 2
4000 wenn INT(X) = 3
```

Ist INT(X) ein Wert ungleich 1,2 oder 3, wird die auf die "ON"-Anweisung folgende Anweisung ausgeführt.

```
.
1000 ON X GOTO 2000,3000,4000
.
```

Beschreibung der Anweisungen

10.3.5 "IF"-Anweisung

Durchführung einer beliebigen Anweisung, in Abhängigkeit von einer Bedingung.

Syntax:

```
                  ⎡ERR 0 [<ANWEISUNG>]                                    ⎤
<IF>   ::= IF     ⎢<S-EXPR1> [<V-OP><S-EXPR2>] [<ANWEISUNG>]               ⎥
                  ⎣<N-EXPR1> [<V-OP><N-EXPR2>] [<ANWEISUNG>]               ⎦
```

Funktion:

IF = Anweisung.

ERR 0 = Sonderfall für Fehlerbehandlung. Die Beschreibung erfolgt unter Pkt. 10.3.6.

String-Vergleich

<S-EXPR1> = Erster zu vergleichender Operand.

<V-OP><S-EXPR2> = Vergleichsoperand und zweiter zu vergleichender Operand. Ist diese Angabe nicht gemacht, wird nur geprüft, ob das 1. Zeichen von <S-EXPR1> ein Grenzzeichen ist. Ist dies der Fall, wird Gleichheit erkannt, das heißt die Bedingung gilt als erfüllt.

<ANWEISUNG> = Die Anweisung, die auszuführen ist, wenn die Vergleichsbedingung erfüllt ist.
Beim Vergleich von String-Variablen muß der Inhalt der beiden zu vergleichenden String-Variablen inclusive des ersten Grenzzeichens identisch sein.
Wird eine String-Variable mit einem String-Literal verglichen, wird die Anzahl zu vergleichender Zeichen von der Länge des Literals bestimmt.
Beim Vergleich von String-Expressions wird nur das Grenzzeichen des letzten Elements in den Vergleich einbezogen!

Beispiel: Vergleich des Inhalts eines Strings mit dem Inhalt von

drei weiteren Strings.

```
.
1000 IF X$=A$,B$,C$ LET...
.
```

Die Grenzzeichen von A$ und B$ werden ausgeblendet.

Vergleich numerischer Werte

<N-EXPR1> = Erster zu vergleichender Operand.

<V-OP><N-EXPR> = Vergleichsoperand und zweiter zu vergleichender Operand. Ist diese Angabe nicht codiert, wird nur überprüft, ob der Wert von <N-EXPR1> ungleich 0 ist. Ist dies der Fall, gilt die Vergleichs-Bedingung als erfüllt.

<ANWEISUNG> = Die Anweisung, die auszuführen ist, wenn die Vergleichsbedingung erfüllt ist.

Ist die angegebene Variable eine Matrix oder ein Vektor, wird nur Element 0,0 bei Matrizen bzw. 0 bei Vektoren verglichen. Es wird - unabhängig von dem Format der zu vergleichenden Variablen bzw. Ausdrücke und der Datendarstellung - der algebraische Wert verglichen.
+0 und -0 werden als gleich erkannt.

Beschreibung der Anweisungen

Verknüpfen mehrerer "IF"-Anweisungen

Es besteht die Möglichkeit, in einer Anweisung mehrere Bedingungen zu verknüpfen, indem das Schlüsselwort "IF" mehrmals angegeben wird.

Beispiel: IF X>0 IF X<10 IF FRA X=0 GOTO 9000

Sind mehrere Vergleichsbedingungen angegeben, werden diese durch "und" miteinander verknüpft. Das heißt, daß alle Vergleichsbedingungen erfüllt sein müssen, damit die angegebene Anweisung zur Ausführung kommt.

Beispiele:

1) Vergleich zweier String-Variablen auf Gleichheit. Ist die Bedingung erfüllt, wird eine Programmverzweigung zur Zeilennummer 9000 durchgeführt.

```
1000 IF A$=B$ GOTO 9000
```

2) Überprüfen eines Strings, ob er ein Leerstring ist. Ist dies der Fall, wird der Text "FALSCHE EINGABE" ab der aktuellen Cursorposition am Bildschirm angezeigt.

```
1000 IF A$="" PRINT "FALSCHE EINGABE";
```

3) Abprüfen eines Strings, auf eine Länge von 3 Byte und den Inhalt "END". Ist dies der Fall wird ein "CHAIN" zu dem TAMOS-Programm "TA.END" durchgeführt.

```
1000 IF LEN A$=3 IF A$="END" CHAIN "TA.END"
```

4) Vergleich einer numerischen Variablen auf minimal 1 und maximal 9. Der Bruchteil (FRA) muß 0 sein. Ist dies der Fall wird zur Zeilennummer 9000 verzweigt.

```
1000 IF A>0 IF A<10 IF FRA A=0 GOTO 9000
```

10.3.6 "IF ERR 0"-Anweisung

Ermöglicht eine gezielte Reaktion beim Auftreten von Basic-Fehlern.

Syntax:

<IF ERR 0> :: = IF ERR 0 <ANWEISUNG>

Funktion:

IF ERR 0 = Anweisung.

<ANWEISUNG> = Die Anweisung, die beim Auftreten eines Basic-Fehlers auszuführen ist. In der Regel wird dies ein Verzweig in ein Fehler-Behandlungsprogramm sein. Ist <ANWEISUNG> nicht angegeben, werden auftretende Basic-Fehler direkt am BA, in der Form "ERROR # .. AT <ZL-NR>", angezeigt.

"RUN" kann nur eine "IF ERR 0"-Bedingung speichern. Das heißt, wenn mehrere "IF ERR 0"-Anweisungen abgesetzt werden, ist jeweils die Bedingung der zuletzt durchlaufenen "IF ERR 0"-Anweisung aktuell.

Es ist zu beachten, daß zum Zeitpunkt der Fehlererkennung die verursachende Anweisung bereits teilweise abgearbeitet sein kann. Das kann besonders bei den Anweisungen zu Problemen führen, die mehrere Operanden enthalten. Z.B. können die Anweisungen

- OPEN #
- CLOSE #
- KILL

für die Fehlerbehandlung problematisch werden.

Beispiel: CLOSE #2,#3,#4

Tritt bei der Interpretation dieser Anweisung (Schließen eines Kanals) ein Fehler auf, kann nicht festgestellt werden, welcher Kanal ihn verursacht hat.

Die Fehlernummern:

3,6,16,18,19,20,27,31,59,75

können mit "IF ERR 0" nicht maskiert werden. Tritt einer dieser Fehler auf, wird

```
ERROR #.. AT <ZL-NR>
READY
```

ausgegeben. Alle Kanäle werden geschlossen und in den Basic-Processor verzweigt!

Ist keine "IF ERR 0"-Anweisung durchlaufen, werden Fehler mit:

```
ERROR # .. AT <ZL-NR>
```

gemeldet und das Programm weiter ausgeführt.

10.3.7 "FOR"-Anweisung

Start-Anweisung für eine Programmschleife.

Syntax:

<FOR> ::= FOR<N-VAR>=<N-EXPR1>TO<N-EXPR2> [STEP<N-EXPR3>]

Funktion:

FOR = Anweisung.

<N-VAR> = Laufvariable. Dieser Variablen wird ein Wert zugewiesen, bei dessen Über-/Unterschreitung die Programmschleife verlassen wird.

=<N-EXPR1> = Der Laufvariablen einen Anfangswert zuweisen. Der Wert von <N-EXPR1> wird bei der Ausführung der Anweisung "FOR" in <N-VAR> abgestellt.

TO<N-EXPR2> = Endwert. Wert, bei dessen Über-/Unterschreitung die Programmschleife nach der Ausführung einer "NEXT"-Anweisung verlassen wird.

STEP <N-EXPR3> = Schrittweite. Der Wert von <N-EXPR3> wird bei der Ausführung der die Programmschleife beendenden "NEXT"-Anweisung auf <N-VAR> addiert. Ist STEP<N-EXPR3> nicht angegeben, wird als Schrittweite +1 angenommen.

In Abhängigkeit von der Schrittweite wird geprüft, ob der Anfangswert der Laufvariablen bereits über-/unterschritten ist. Ist dies der Fall, wird die Schleife beendet und das Programm mit der Anweisung fortgesetzt, die auf die "NEXT"-Anweisung folgt.

Es können bis zu 8 Programm-Schleifen ineinander geschachtelt werden.

Beschreibung der Anweisungen

10.3.8 "NEXT"-Anweisung

Letzte Anweisung einer Programmschleife.

Syntax:

<NEXT> :: = NEXT <N-VAR>

Funktion:

NEXT = Anweisung.

<N-VAR> = Es ist der Name der in der "FOR"-Anweisung zum Schleifenanfang definierten Laufvariablen anzugeben.

"NEXT" addiert den als <N-EXPR3> in der einleitenden "FOR"-Anweisung vorgegebenen Wert auf die Laufvariable (<N-VAR>).
Ist der Endwert über-/unterschritten, wird die Schleife beendet und die auf "NEXT" folgende Anweisung ausgeführt. Ist dieser Wert noch nicht erreicht, wird zu der auf die einleitende "FOR"-Anweisung folgenden Anweisung verzweigt.

Beispiele zu "FOR" und "NEXT"

1) Zehnmaliges Durchlaufen einer Programmschleife.

```
  .
  1000 FOR I=1 TO 10
 |>1005 ...
 |  .
 |  .
 |_1100 NEXT I
   1105 ...
```

Beschreibung der Anweisungen

2) 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
.
```

3) Verlassen einer Programmschleife mit der Anweisung "GOSUB" in Abhängigkeit von einer Bedingung.

```
.
1000 FOR I=X TO X+Y STEP N
1005 ...
.
.
1100 IF C GOSUB 2000
1105 ...
1110 ...
.
.
1200 NEXT I
1205 ...
.
.
2000 ...
.
.
2100 RETURN 1
```

Beschreibung der Anweisungen

10.3.9 "CHAIN"-Anweisung

Aufruf eines beliebigen Programmes oder Processors.

Syntax:

<CHAIN> :: = CHAIN { <S-VAR> / <S-LIT> }

Funktion:

CHAIN = Anweisung.

<S-VAR> = String-Variable oder String-Literal, die den Namen des aufzurufenden Programmes/Processors enthält.

Die direkte Datenübergabe von einem zum ande-Programmsegment innerhalb der Partition ist mit den CHAIN-Anweisungen möglich.Zur Datenübergabe an ein nachfolgendes Programm bestehen die folgenden Möglichkeiten:

- Übergabe der Daten im Common-Bereich.
  Für jeden Teilnehmer besteht ein Common-Bereich in der Größe von 512 Byte. Dieser Bereich kann mittels "CALL" angesprochen werden.
  Es können also maximal 512 Byte im Common-Bereich übergeben werden.

- Datenübergabe in Plattendateien. Diese Möglichkeit sollte nur in Ausnahmefällen angewandt werden, da sie relativ zeitaufwendig ist (s. auch "LINK").

Der Aufruf von Processoren und ihre Versorgung mit den erforderlichen Parametern ist im Anhang unter Pkt. 12.2 beschrieben.

10.3.10 "STOP"-Anweisung

Unterbrechen des laufenden Programmes und Ausgabe der Meldung:

STOP AT <ZL-NR>

Es wird in den Basic-Processor verzweigt.

Syntax:

<STOP> ::= STOP

Funktion:

STOP = Anweisung.

Diese Anweisung sollte nur während der Testphase angewendet werden, z.B. um das Programm gezielt an verschiedenen Stellen anzuhalten, um Variablen-Inhalte zu überprüfen.

10.3.11 "END"-Anweisung

Beendigung eines Basic-Programmes. Sämtliche eröffneten Kanäle werden geschlossen und in den Basic-Processor verzweigt.

Syntax:

<END> :: = END

Funktion:

END = Anweisung.

Diese Anweisung sollte nur in der Testphase angewendet werden. Ein Beenden von Programmen erfolgt in der Regel über "CHAIN" in ein Steuerprogramm bzw. unter TAMOS in eines der entsprechenden TAMOS-Programme.

10.3.12 LINK - Anweisung

Diese Anweisung dient zum Laden eines Basic-Programmes (-Segment) in eine Anwenderpartition, ohne, daß die Variablen des aufrufenden Programmes gelöscht werden.

Es bleiben sowohl die vom aufrufenden Programm eröffneten Kanäle, als auch die in der Partition verbleibenden Variablen mit ihren Namen, Werten, Dimensionen und Genauigkeit für das neue (aufgerufene) Programm erhalten.

Anmerkung:

Zur Ausführungszeit dieser Anweisung dürfen max. 79 Variablen dimensioniert sein. Sind mehr vorhanden sind sie mit DEALLO zu deaktivieren.

Syntax:

<LINK> ::= LINK [<S-VAR> / <S-LIT>]

Funktion:

LINK = Anweisung.

[<S-VAR> / <S-LIT>] = String-Variable oder String-Literal, mit dem Namen des aufzurufenden Programmes.

Anmerkung:

In einer LINK-Anweisung ist die Angabe von jeweils einem Programmnamen erlaubt.

Variablen des aufrufenden Programmes, müssen im aufgerufenen Programm definiert sein, d.h., in mindestens einer Anweisung auftreten und dürfen außerdem nicht erneut dimensioniert werden.

Fehler # 75

"STORAGE OR VARIABLES DON'T MATCH IN CHAIN"

- Die angegebene Datei ist kein Basic-Programm.
- Das Programm ist nicht auf der angegebenen LU
- Es wurde mehr als 79 Variablen übergeben.
- Die Kapazität der Partition ist zu gering um das Pro gramm (-Segment) aufzunehmen .
  In diesem Fall bleibt die Steuerung beim aufgerufenen Programm.
- Eine, oder mehrere Variablen die verfügbar bleiben sollen, sind im aufgerufenen Programm nicht definiert.

Beschreibung der Anweisungen

10.4 Ein-/Ausgabe-Anweisung

In diesem Kapitel werden sämtliche Anweisungen beschrieben, die Ein-/Ausgaben auf beliebige angeschlossene Peripheriegeräte erlauben.
Im Einzelnen sind dies:

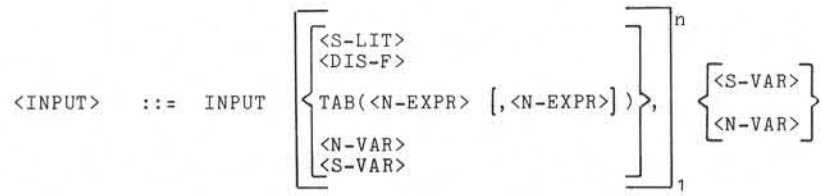
- INPUT
- PRINT
- OPEN #
- CLOSE #
- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #
- SEARCH #
- BUILD #
- KILL

Funktionen der Anweisung "CALL", die ebenfalls Ein-/Ausgaben ermöglichen, sind im Anhang bei der Beschreibung der CALL - Funktionen erläutert.

10.4.1 "INPUT" - Anweisung

Eingabe von Daten über die Tastatur in beliebige Variable. Wahlweise können zusätzlich beliebige Führungstexte und/oder beliebige Bildschirmfunktionen ausgegeben werden. Die eingegebenen Daten werden ab der aktuellen Cursorposition, bzw. nachdem der Führungstext angezeigt ist, am Bildschirm angezeigt.

Syntax:



Beschreibung der Anweisungen

Funktion:

INPUT = Anweisung.

<S-LIT> = Beliebiges String-Literal. Dient zur Ausgabe von Führungstexten. <S-LIT> wird ab der aktuellen Cursorposition am Bildschirm angezeigt.

<DIS-F>. = Beliebige Display-Funktion. Es ist die Angabe der von Pkt. 7.2.2.2 bis 7.2.2.15 beschriebenen Funktionen möglich.

TAB = Tabulation des Cursors auf eine beliebige Bildschirmposition. Es kann:

- eine Spalte in der aktuellen Zeile angegeben werden: TAB (<N-EXPR>)
- eine Zeile und eine Spalte angegeben werden: TAB (<N-EXPR>,<N-EXPR>)

Die Spaltenangabe muß in dem Bereich von 0 bis 79, die Zeilenangabe im Bereich von 0 bis 24 liegen, andernfalls ist das Ergebnis undefiniert. Wird nur die Position innerhalb der Zeile angegeben, ist darauf zu achten, daß diese größer als die aktuelle Position ist.

<N-VAR> bzw. <S-VAR> = Variable, in die eingegeben werden soll. Es können in einer "INPUT"-Anweisung mehrere Eingabevariablen angegeben werden.

Sämtliche Operanden sind durch Komma zu trennen. Sind mehrere Eingabevariable angegeben, muß jede Eingabe mit der Taste "CR" ausgelöst werden. Als letzter Operand muß eine Eingabevariable angegeben sein! Ist der vorletzte Operand keine Eingabevariable, kann das Komma zwischen den beiden letzten Operanden entfallen.

Bei Auftreten einer "INPUT"-Anweisung wird das Programm so lange auf "INPUT-mode" geschaltet, bis die Eingabe beendet ist.
Das bedeutet, daß das Programm in den Wartezustand versetzt wird, also bis zur Beendigung der Eingabe keine Rechnerzeit mehr erhält.

Jedem BA ist im System ein Eingabepuffer in Größe von 256 Byte zugeordnet. Von diesen 256 Byte sind 12 Byte als

Beschreibung der Anweisungen

Voreingabepuffer reserviert und davon dienen 11 zur Pufferung von eingegebenen Daten und ein Byte als Zeiger im Voreingabepuffer. Das ermöglicht, außerhalb des "INPUT-mode" bis zu 11 Zeichen einzugeben. Werden trotzdem mehr als 11 Zeichen eingegeben, wird der akustische Signalgeber gesetzt und die zuviel eingegebenen Zeichen nicht übernommen.
Wird außerhalb des "INPUT-mode" eine der Auslösetasten betätigt, wird ebenfalls der akustische Signalgeber gesetzt und der Code 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 zum Zeitpunkt des Umschaltens auf "INPUT-mode" ausgegeben.
Werden zwei Tasten gleichzeitig gedrückt, wird das durch den akustischen Signalgeber gemeldet und kein Zeichen übernommen.
Steht der Cursor in einem Hintergrundfeld, ist keine Eingabe möglich.
Ist eine Eingabefeld am Bildschirm durch ein Hintergrundzeichen begrenzt und es werden mehr Zeichen eingegeben als erlaubt, wird die gesamte Eingabe gelöscht.

"INPUT" bietet zusätzlich die Möglichkeit, Führungstexte und Bildschirm-Funktionen anzugeben, bevor die eigentliche Eingabe (Input-mode) erfolgt. Das bedeutet, daß die "INPUT"-Anweisung in beschränktem Maße auch Ausgabefunktionen übernehmen kann. Führungstexte können allerdings nicht in Variablen stehen, sondern müssen als alphanumerische Konstante in der Anweisung angegeben werden.

Beispiel:
1) Ausgabe des Führungstextes "RABATT" ab Spalte 0 in Zeile 10.
2) Ausgabe von 2 Punkten zur Anzeige der maximal einzugebenden Anzahl Stellen ab Position 30.
3) Ausgabe eines Hintergrund-Blanks zur Begrenzung des Eingabefeldes.
4) Umschalten auf Vordergrund.
5) Tabulation auf das erste Zeichen des Eingabefeldes.
6) Eingabe in die Variable "R".

```
INPUT TAB(0,10),"RABATT",TAB(30)"..",'SB'," ",'SF',TAB(30,10)R
        1                2        3    4    5    6
```

Beschreibung der Anweisungen

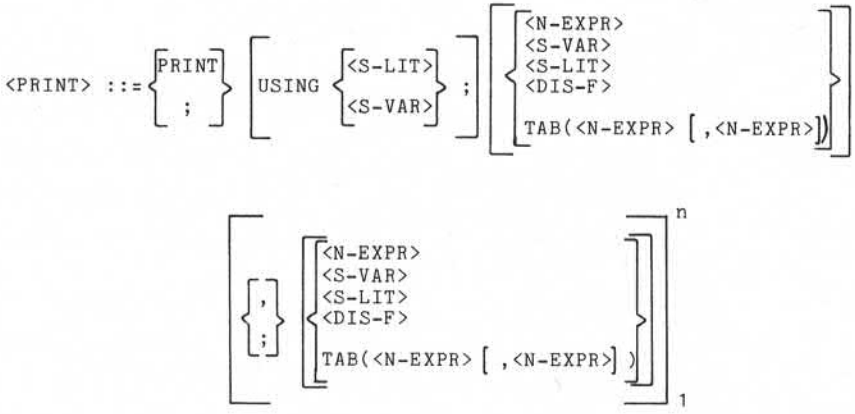
10.4.2 "PRINT"-Anweisung

"PRINT" ermöglicht die Ausgabe von:

- String-Variablen
- numerischen Variablen
- Konstanten
- BA-Funktionen

in beliebiger Reihenfolge am Bildschirm.

Syntax:



Funktion:

PRINT
;
= Anweisung.
Bei der Eingabe der Anweisung kann statt "PRINT" auch ein Semikolon ";" angegeben werden.

USING <S-LIT> / <S-VAR>
= Mit dieser Angabe wird eine Aufbereitungsmaske vorgegeben, welche die Aufbereitung aller mit der Anweisung auszugebenden numerischen Variablen steuert.
Die Maske selbst wird in <S-LIT> oder <S-VAR> angelegt.

Die Angabe "USING" darf in einer "PRINT"-Anweisung nur einmal angegeben werden.
Der Aufbau von Masken und die Funktionen der zur Verfügung stehenden Masken-Steuerzeichen sind unter Pkt.: 4.4 beschrieben.
Die "USING"-Angabe muß vom folgenden Operanden unbedingt durch ein Semikolon ";" getrennt sein.

<N-EXPR> = Eine beliebige Anzahl und Folge von:
<S-VAR> - String-Variablen
<S-LIT> - numerischen Ausdrücken
<DIS-F> - Konstanten
TAB - BA-Funktionen

Die Ausgabe von Gleitkommazahlen bzw. BCD-Ganzzahlen, die nicht über eine Maske gesteuert wird, erfolgt ab der aktuellen Cursorposition in der Form:

- Vorzeichen (1 Zeichen). Für "+" wird Blank "␣" und für "-" wird Minus "-" ausgegeben.
- Im Anschluß an das Vorzeichen die signifikanten Ziffern. Führende Nullen werden unterdrückt.
- Im Anschluß an die zuletzt ausgegebene Ziffer wird ein Blank "␣" ausgegeben.

Enthält die auszugebende Variable keine signifikanten Ziffern, wird eine Null "0" ausgegeben.
Bei der Ausgabe von Gleitkommazahlen ist besonders darauf zu achten, daß die Ausgabe ohne Maskenangabe ("PRINT USING") im Gleitkommaformat erfolgt, wenn:

- die Anzahl der signifikanten Ziffern 14 überschreitet.

- ein Wert kleiner 0,1 entsteht.

Beispiel: Der Wert 0,01 wird als
1E -2
ausgegeben.

Der Wert 999999999999999 wird als
9,9999999999999E +14
ausgegeben.

Anmerkung: Formularsteuerzeichen für Drucker, mit Ausnahme von "CR", bewirken keine Ausgabe am Bildschirm.

Beschreibung der Anweisungen

Als Trennzeichen zwischen den Operanden dienen "," (Komma) und ";" (Semikolon).

Trennzeichen "," (Komma)

Nach Bearbeitung des vorhergehenden Operanden wird der Cursor auf die nächsthöhere (ohne Rest durch 15 teilbare) Spaltenposition tabuliert. Ist dieser Wert größer als 79, wird ein Zeilenvorschub (siehe Pkt. 7.2.2.5) durchgeführt.

Trennzeichen ";" (Semikolon)

Nach Bearbeitung des vorhergehenden Operanden bleibt der Cursor auf der aktuellen Position stehen.

Nachdem der letzte Operand einer "PRINT"-Anweisung abgearbeitet ist, wird ein Zeilenvorschub durchgeführt. Soll das verhindert werden und der Cursor auf der aktuellen Position stehenbleiben, ist hinter dem letzten Operanden ein ";" anzugeben.

Jedem BA ist im System ein Ausgabepuffer in Größe von 256 Byte zugeordnet.
Ausgaben am Bildschirm erfolgen erst, wenn:

- ein Task-Wechsel durchgeführt wird, d.h. ein gleichzeitig laufendes Programm seine Zeitscheibe erhält oder
- der Ausgabepuffer voll ist oder
- eine "SIGNAL 3" - Anweisung oder
- eine "INPUT" - Anweisung oder
- eine PRINT # - Anweisung oder
- eine LINK - Anweisung oder eine "CHAIN" - Anweisung auftritt.

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, Zeile 24 wird frei.

Anmerkung: Alle Ausgaben erfolgen grundsätzlich ab der aktuellen Cursorposition. Der Cursor wird intern geführt und nur durch die "INPUT" - Anweisung sichtbar gemacht.

Beschreibung der Anweisungen

10.4.3 "OPEN #" - Anweisung

Eröffnen von Dateien auf beliebigen Kanälen für deren Bearbeitung.

Syntax:

<OPEN #> ::= OPEN #<N-EXPR> [;<S-VAR1>] , < [<S-LIT>] [<S-VAR2>] >

[[,#<N-EXPR>] [;<S-VAR1>] , < [<S-LIT>] [<S-VAR2>] >] n 1

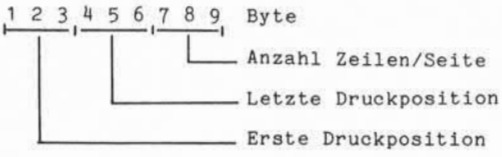
Funktion:

OPEN # = Anweisung.

<N-EXPR> = Angabe der Nummer des Kanals, auf dem die Datei eröffnet werden soll. Standardmäßig stehen 14 Kanäle (0 - 13) pro Teilnehmer zur Verfügung. Beim Einsatz von TAMOS sind die Kanäle 0 und 1 belegt.

<S-VAR1> = Diese Angabe wird nur bei Eröffnung von Druckerdateien ausgewertet. In <S-VAR1> werden Parameter zur Formularsteuerung vorgegeben und muß für mindestens 9 Byte dimensioniert sein.

Aufbau:



Beim Einsatz eines Zeilendruckers werden die erste und die letzte Druckposition nicht ausgewertet. Die letzte

Druckposition muß immer größer sein als die erste Druckposition.
Beim Einsatz eines Nadeldruckers mit zwei Vorschubaggregaten dürfen sich die Druckbereiche nicht überlappen.

Werden keine Angaben zur Formularsteuerung gemacht, gelten folgende Standardwerte:

— Zeilendrucker:

Erste Druckposition	= 000
Letzte Druckposition	= 131
Formularhöhe	= 048

— Nadeldrucker linker Vorschub:

Erste Druckposition	= 000
Letzte Druckposition	= 176
Formularhöhe	= 048

— Nadeldrucker rechter Vorschub

Erste Druckposition	= 177
Letzte Druckposition	= 177
Formularhöhe	= 048

<S-LIT>
<S-VAR> = Name der Datei, die eröffnet werden soll.

Die Angabe der Kanalnummer ist nur für die erste mit einer "OPEN #"-Anweisung zu eröffnende Datei zwingend erforderlich. Sollen mehrere Dateien mit einer "OPEN #"-Anweisung auf aufeinanderfolgenden Kanälen eröffnet werden, ist nur die Kanalangabe für die erste Datei erforderlich, z.B.:

```
OPEN #2,"$LPT",A$,B$,C$
```

Die Dateien ohne führende Kanalnummer werden in aufeinanderfolgenden Kanälen eröffnet (Kanalnummer der letzten mit dieser Anweisung eröffneten Datei + 1).
Der Nachteil beim Eröffnen mehrerer Dateien mit einer "OPEN #"-Anweisung ist eine unzureichende Fehlerauswertung, d.h. tritt bei der Ausführung ein Fehler auf, kann nicht festgestellt werden, welche Dateieröffnung den Fehler verursacht hat.

10.4.4 "CLOSE #"-Anweisung

Schließen von eröffneten Dateien.

Syntax:

<CLOSE #> ::= CLOSE #<N-EXPR> [,#<N-EXPR>]$_{1}^{n}$

Funktion:

CLOSE # = Anweisung.

<N-EXPR> = Nummer des Kanals, auf dem die zu schließende Datei eröffnet ist. Mit einer "CLOSE #"-Anweisung können beliebig viele Dateien geschlossen werden. Ist unter der angegebenen Kanalnummer keine Datei eröffnet, wird Basic-Fehler # 49 gemeldet. Der Nachteil des Schließens mehrerer Dateien (Kanäle) mit einer "CLOSE #"-Anweisung ist eine unzureichende Fehlerauswertung, d.h. tritt dabei ein Fehler auf, kann nicht festgestellt werden, welche Datei den Fehler verursacht hat.

Beispiel: Routine zum Schließen der Kanäle 2 bis 13. Die von TAMOS belegten Kanäle 0 und 1 werden nicht geschlossen.

```
.
1000 FOR I=2 TO 13
1010 IF ERR 0 GOTO ... /*AUSWERTUNG FEHLER 49
1020 CLOSE #I
1030 IF ERR 0 GOSUB .. /*STANDARD FEHLERBEH.
1040 NEXT I
```

10.4.5 "READ #"-Anweisung

Lesen von Daten aus Dateien und Übertragen der gelesenen Daten in beliebige Variable.
Diese Anweisung ist nur für Magnetplattendateien und Lochkartenleser-Dateien zugelassen.

Syntax:

```
<READ #> ::= READ #<N-EXPR1> [,<N-EXPR2> [,<N-EXPR3>]] ;

             { <N-VAR> }  [ <N-VAR> ]
             { <S-VAR> } , [ <S-VAR> ]   n   [;]
                                     1
```

Funktion:

READ # = Anweisung.

<N-EXPR1> = Nummer des Kanals auf dem die Datei eröffnet ist.

<N-EXPR2> = Satznummer des zu lesenden Satzes relativ zum Dateianfang (Satz # 0).

Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben. Ist keine Satznummer angegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet. Ist <N-EXPR2> = -1, erfolgt die Bearbeitung ebenfalls sequentiell. Ist <N-EXPR2> = -2, wird der Satz gelesen, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde.
(Ausnahme : Textdateien).

Bei der Bearbeitung von Lochkartenleser-Dateien ist die Angabe <N-EXPR2> ohne Bedeutung.

<N-EXPR3> = Vorgabe der Anfangsadresse innerhalb eines Datensatzes.
Diese Angabe ist nur dann erlaubt, wenn <N-EXPR2> codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld # 0) angegeben, ab dem die Übertragung der Daten beginnen soll.

Bei allen anderen Magnetplattendateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0) anzugeben.

Bei der Bearbeitung von Lochkartenleser-Dateien ist die Angabe <N-EXPR3> ohne Bedeutung.

<N-VAR> bzw. <S-VAR> = Beliebig viele numerische und/oder String-Variablen, Elemente von Vektoren und/oder Matrizen. In diese Variablen wird der Inhalt des adressierten Datensatzes übertragen.
Bei der Bearbeitung von Textdateien dürfen keine <N-VAR> angegeben sein.
Eine Überprüfung des Formats der Daten, die übertragen werden sollen, erfolgt nur bei formatierten Dateien.
Bei relativen Dateien erfolgt die Übertragung ohne Rücksicht auf Typ/Format der Zielvariablen.
Die Anzahl Zeichen, die in die jeweiligen Zielvariablen zu übertragen sind, werden in Abhängigkeit von dem Dateityp gesteuert durch:

- das dimensionierte Format (1-4 Worte) bei numerischen Variablen.
- das Auftreten eines Grenzzeichens oder das Erreichen der dimensionierten bzw. durch Indizes angegebenen Länge bei String-Variablen.
- das Auftreten von "CR"-Codes oder das Erreichen der dimensionierten bzw. durch Indizes angegebenen Länge bei String-Variablen.

Besonderheiten, die bei den verschiedenen Dateitypen zu beachten sind, werden unter Pkt.: 7.3.5.2 bis 7.3.5.5 beschrieben!

; = Kennzeichen für Satzsperre.
Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung gelesene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt.
Ist ein Semikolon codiert, erfolgt keine Satzsperre (siehe Pkt.: 7.3.3, File-Sharing).

READ # auf Lochkartenleser-Dateien bewirkt keine Satzsperre.

Die Angaben zur Adressierung der zu lesenden Daten (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon ";" von den Namen der "Ziel-Variablen" getrennt sein.

Für Lochkartenleser-Dateien ist als Ziel-Variable nur eine String-Variable zugelassen, da nur alphanumerische Daten (ASCII-Code) angeliefert werden.

Beispiele: 1) Anweisung zum sequentiellem Lesen.
Die Kanalnummer wird in der Variablen "C" vorgegeben. Die Daten werden in die Variablen A, B, C und A$ übertragen.
Nach der Durchführung dieser Anweisung ist der gelesene Datensatz gesperrt.

```
1000 READ #C;A,B,C,A$
```

2) 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 im adressierten Satz. Der Satz wird nicht gesperrt!

```
1000 READ #C,R,50;A$;
```

Beschreibung der Anweisungen

10.4.6 "WRITE #"-Anweisung

Übertragen von Daten aus beliebigen Variablen in Dateien. Diese Anweisung ist für Magentplattendateien und für Druckerdateien zugelassen.

Syntax:

<WRITE #> ::= WRITE #<N-EXPR1> [,<N-EXPR2> [,<N-EXPR3>]] ;

{ <N-EXPR4> | <S-VAR> | <S-LIT> } [, [<N-EXPR4> | <S-VAR> | <S-LIT>]]n 1 [;]

Funktion:

WRITE # = Anweisung.

<N-EXPR1> = Nummer des Kanals auf dem die Datei eröffnet ist.

<N-EXPR2> = 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. Ist keine Satznummer angegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet. Ebenso wenn <N-EXPR2> = -1 ist.
Ist <N-EXPR2> = -2, wird der Satz geschrieben, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde, z.B.: den zuletzt gelesenen und geänderten Satz zurückschreiben. (Ausnahme: Textdateien).

Bei der Bearbeitung von Druckerdateien ist die Angabe <N-EXPR2> ohne Bedeutung.

<N-EXPR3> = Vorgabe der Anfangsadresse innerhalb eines Datensatzes.
Diese Angabe ist nur dann erlaubt, wenn <N-EXPR2> codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld ≠ 0) angegeben, ab dem die Daten in den adressierten Satz übertragen werden soll. Bei allen anderen Magnetplattendateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte ≠ 0) anzugeben.
Bei der Bearbeitung von Druckdateien ist die Angabe <N-EXPR3> ohne Bedeutung.

<N-EXPR4>
<S-VAR>
<S-LIT>

= Beliebige Anzahl von
- Numerischen Ausdrücken
- String-Variablen
- String-Literalen

in beliebiger Reihenfolge.

Der Inhalt dieser Variablen wird in den adressierten Datensatz übertragen.
Bei der Bearbeitung von Druckdateien und Textdateien dürfen keine <N-EXPR> angegeben werden. Eine Überprüfung des Formats der Daten, die übertragen werden sollen, erfolgt nur bei formatierten Dateien.
"WRITE #" stellt hinter geschriebenen String-Variablen grundsätzlich ein Grenzzeichen ab, wenn der Quell-String in der dimenionierten Länge ausgegeben wird.
Ist ein Substring angegeben, wird kein Grenzzeichen ausgegeben, während bei Stringliteralen immer ein Grenzzeichen ausgegeben wird. Die Anzahl Zeichen, die in den Datensatz übertragen werden, werden in Abhängigkeit von dem Dateityp gesteuert durch:

- das dimensionierte Format (1-4 Worte) bei numerischen Variablen bzw. immer 4 Worte bei numerischen Ausdrücken.
- das Auftreten eines Grenzzeichens oder das Erreichen der dimensionierten bzw. durch Indizes angegebenen Länge bei String-Variablen.
- die Anzahl der Zeichen bei String-Literalen.

Besonderheiten, die bei den verschie-

denen Dateitypen zu beachten sind, werden unter Pkt. 7.3.5.2 bis 7.3.5.5 beschrieben.

; = Kennzeichen für Satzsperre.
Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung geschriebene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt.
Ist ein Semikolon codiert, erfolgt keine Satzsperre (siehe auch Pkt. 7.3.3, File-Sharing).

Bei der Bearbeitung von Druckerdateien ist diese Angabe ohne Bedeutung.

Die Angaben zur Adressierung der zu schreibenden Daten (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon ";" von den Namen der zu schreibenden Variablen getrennt sein!

Beispiele: 1) Direktes Schreiben des Inhalts der Variablen "A" in Feld # 5 einer formatierten Datei.
Die Kanalnummer wird in der Variablen "C" vorgegeben.
Die Satznummer wird aus dem Inhalt der Variablen "R" + "S" gebildet.
Der Satz wird nicht gesperrt!

```
1000 WRITE #C,R+S,5;A;
```

2) Sequentielles Lesen, Verändern und Zurückschreiben in einer relativen Datei.
Die Kanalnummer wird in der Variablen "C" vorgegeben.
Die Variablen A, B, C und A$ werden bearbeitet. Nach dem Lesen wird der betreffende Satz gesperrt und nach dem Schreiben wieder freigegeben.

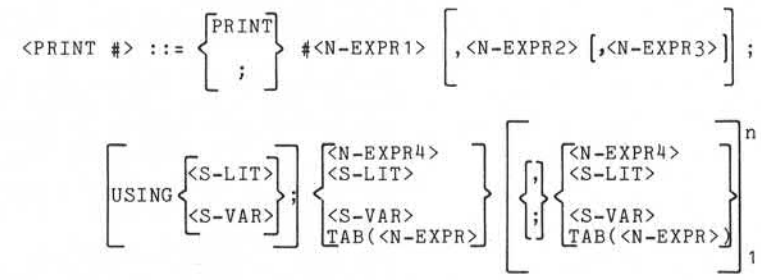
```
1000 READ #C;A,B,C,A$
1010 GOSUB 2000   /*DATEN VERÄNDERN
1020 WRITE #C,-2;A,B,C,A$;
1030 GOTO 1000
```

Beschreibung der Anweisungen

10.4.7 "PRINT #"-Anweisung

Übertragen von Daten aus beliebigen Variablen in Dateien mit gleichzeitiger Umwandlung des Inhalts von numerischen Variablen in alphanumerische (druckbare ASCII) Daten. Diese Anweisung gilt für Magnetplatten und Drucker-Dateien.

Syntax:



Funktion:

PRINT #
;
= Anweisung.
Bei der Eingabe der Anweisung kann statt "PRINT" auch ein Semikolon ";" eingegeben werden.

<N-EXPR1> = Die Nummer des Kanals, auf dem die Datei eröffnet ist.
Ist der angegebene Kanal nicht eröffnet, erfolgen sämtliche Ausgaben ohne Fehlermeldung am Bildschirm.

<N-EXPR2> = 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.

Ist keine Satznummer vorgegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet.
Ebenso wenn <N-EXPR2> = -1 ist.

Ist <N-EXPR2> = -2, wird der Satz geschrieben, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde (Ausnahme: Text-Dateien).

Bei der Bearbeitung von Druckerdateien ist die Angabe <N-EXPR2> ohne Bedeutung.

<N-EXPR3> = Vorgabe der Anfangsadresse innerhalb eines Datensatzes.
Diese Angabe ist nur dann erlaubt, wenn <N-EXPR2> codiert ist.

Bei formatierten Dateien wird die Nummer des Feldes relativ zum Anfang des Datensatzes (Feld # 0) angegeben, ab dem die Übertragung der Daten beginnen soll.
Es können in formatierten Dateien nur Felder bearbeitet werden, die nicht numerisch sind.
Bei allen anderen Dateien ist die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0) anzugeben. Es ist zu beachten, daß mit "PRINT #" nur ASCII-Code ausgegeben wird, da der Inhalt von numerischen Variablen vor der Ausgabe in ASCII-Code umgewandelt wird (automatisches Editieren).

Bei der Bearbeitung von Druckerdateien ist die Angabe <N-EXPR3> ohne Bedeutung.

USING { <S-LIT> / <S-VAR> } = Mit dieser Angabe wird eine Aufbereitungsmaske vorgegeben, welche die Aufbereitung aller mit dieser Anweisung auszugebenden numerischen Ausdrücke steuert. Die Maske selbst wird in <S-LIT> oder <S-VAR> vorgegeben.
Die Angabe "USING" ist in einer "PRINT" Anweisung nur einmal erlaubt.

Der Aufbau von Masken und die Funktion der zur Verfügung stehenden Masken-Steuerzeichen sind unter Pkt. 4.4 beschrieben.
Die "USING"-Angabe muß vom folgenden

Operanden unbedingt durch ein Semikolon ";" getrennt sein.

<N-EXPR4> = Beliebige Anzahl von
<S-LIT> - numerische Ausdrücken,
<S-VAR> - String-Variablen,
TAB (<N-EXPR>) - String-Literalen und
- TAB-Funktionen
in beliebiger Reihenfolge.

; = Zeilenvorschub-Steuerung.
Ist im Anschluß an den letzten Operanden in der "PRINT #"-Anweisung kein Semikolon ";" codiert, wird zusätzlich zu den Anwenderinformationen ein Zeilenvorschub-Code ("CR" = 215 oktal) ausgegeben.

Die Ausgabe des Inhalts von numerischen Variablen und numerischen Ausdrücken erfolgt nach der Umwandlung in ASCII-Code. Die Ausgabe von Gleitkommazahlen bzw. BCD-Ganzzahlen, die nicht über Masken aufbereitet werden, erfolgt in folgender Form:

- Vorzeichen (1 Zeichen), für "+" wird Blank "␣" und für "-" wird Minus "-" ausgegeben.
- Im Anschluß an das Vorzeichen werden die signifikanten Ziffern ausgegeben. Führende Nullen werden unterdrückt.
- Im Anschluß an die zuletzt ausgegebene Ziffer wird ein Blank "␣" ausgegeben.

Enthält die auszugebende Variable den Wert 0, wird "0" ausgegeben.

Bei der Ausgabe von Gleitkommazahlen ist besonders darauf zu achten, daß ohne Maskenangabe ("USING") die Ausgabe im Gleitkommaformat erfolgt, wenn:

- die Anzahl der signifikanten Ziffern 14 überschreitet

- ein Wert kleiner 0,1 entsteht

Beispiele: 1) Der Wert 0,01 wird als
1E -2
ausgegeben.

2) Der Wert 999999999999999 wird als
9,9999999999999E +14
ausgegeben.

Beschreibung der Anweisungen

Das System führt pro Dateikanal einen "TAB - Zeiger", der die Anzahl ausgegebener Zeichen seit dem letzten:

- "CR" - Code (oktal 215)

bzw. seit Eröffnung der Ausgabedatei beinhaltet.

Die Funktion "TAB (<N-EXPR>)" gibt soviel Blanks aus, wie die Differenz zwischen dem aktuellen TAB-Zeiger und der mit <N-EXPR> angegebenen TAB-Position beträgt.

Achtung! Die angegebene TAB-Position (<N-EXPR>) muß größer als der aktuelle TAB-Zeiger sein. Ist dies nicht der Fall, ist das Ergebnis der Funktion undefiniert.

Der TAB-Zeiger wird auf 0 gesetzt, wenn:

- ein "CR"-Code auftritt.
- die Anweisung komplett abgearbeitet und hinter dem letzten Operanden kein Semikolon codiert ist (dann erzeugt PRINT # "CR"-Code).

Die Angaben zur Adressierung der zu schreibenden Daten (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon ";" von dem Folge-Operanden ("USING" oder die Namen der zu schreibenden Variablen) getrennt sein.
Die Namen der Variablen, deren Inhalt ausgegeben werden soll, müssen durch Komma (",") oder Semikolon (";") voneinander getrennt sein.

Trennzeichen "," (Komma)

Nach der Ausgabe des Inhalts der vorhergehenden Variablen werden soviel Blanks ausgegeben, bis die aktuelle TAB-Position ohne Rest durch 15 teilbar ist.

Trennzeichen ";" (Semikolon)

Die Ausgabe des Inhalts der auf ";" folgende Variablen erfolgt bündig zur letzten Ausgabe.

Die Anweisung "PRINT #" für Magnetplattendateien setzt immer Satzsperre für den geschriebenen Datensatz auf.
"PRINT #" in Plattendateien setzt grundsätzlich ein Grenzzeichen hinter das letzte übertragene Zeichen!

Beispiel: Ausgabe des Inhalts der Variablen K$ auf Position 10. Ausgabe des Textes "SEITE: " ab Position 70. Im Anschluß daran wird der Inhalt der Variablen A über die Maske "####" ausgegeben.

```
PRINT #C;USING "####";TAB(10);K$;TAB(70);"SEITE: ";A
```

Im Anschluß daran wird ein "CR"-Code (215 oktal) ausgegeben. Die Ausgabe erfolgt in die Datei, die auf dem in der Variablen C angegebenen Kanal eröffnet ist.

Beschreibung der Anweisungen

10.4.8 "MAT READ #"-Anweisung

Lesen von Daten aus Magnetplattendateien und Übertragen der gelesenen Daten in eine:

- String-Variable,
- Numerische Variable,
- Matrix oder einen
- Vektor.

Syntax:

<MAT READ #> ::= MAT READ #<N-EXPR1> [,<N-EXPR2> [,<N-EXPR3>]];

{<M-VAR> / <S-VAR>} [;]

Funktion:

MAT READ # = Anweisung.

<N-EXPR1> = Nummer des Kanals, auf dem die Datei eröffnet ist.

<N-EXPR2> = Satznummer des zu lesenden Datensatzes, relativ zum Dateianfang (Satz 0). Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben. Ist keine Satznummer angegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet. Ebenso wenn <N-EXPR2> = -1 ist.
Ist <N-EXPR2> = -2, wird der Satz gelesen, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde.

<N-EXPR3> = Die Adresse des ersten zu übertragenden Bytes relativ zum Anfang des Datensatzes (Byte # 0). Die Übertragung erfolgt wortorientiert. Aus diesem Grunde sollte eine geradzahlige Byte-adresse unter <N-EXPR3> vorgegeben werden.
Wird eine ungerade Byte-Adresse angegeben, beginnt die Übertragung bei der

nächsthöheren geradzahligen Adresse.

<M-VAR>
<S-VAR> = Numerische Variable, Matrix, Vektor oder String-Variable.
Es darf nur ein Variablenname vorgegeben werden! In die codierte Variable wird der Inhalt des adressierten Datensatzes übertragen.
Die Übertragung der Daten erfolgt bis:

- das Ende der aufnehmenden Variablen oder
- das Ende der Datei erreicht ist.

Bei der Bearbeitung eines Vektors oder einer Matrix wird die Variable in ihrer Gesamtheit incl. Reihe und Spalte 0 übertragen.

; = Kennzeichen für Satzsperre.
Wird hinter dem letzten Operanden kein Semikolon ";" codiert, ist der mit dieser Anweisung gelesene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt.
Ist ein Semikolon codiert, erfolgt keine Satzsperre (siehe auch Pkt. 7.3.3, File-Sharing). Die Satzsperre wird nur für den adressierten Satz wirksam, auch dann, wenn mit einer Anweisung mehrerer Datensätze gelesen werden.

Die Angaben zur Adressierung der zu lesenden Daten (<N-EXPR1>, <N-EXPR2> und <N-EXPR3>) müssen durch Semikolon von den Namen der Ziel-Variablen getrennt sein.

Beschreibung der Anweisungen

Beispiele: 1) Lesen der Matrix "A" in ihrer Gesamtheit aus dem Satz, dessen Nummer in der Variablen "R" vorgegeben ist. Die Nummer des Kanals, auf dem die Datei eröffnet ist, ist in der Variablen "C" vorgegeben.

```
.
DIM A(3,3)
.
.
MAT READ #C,R;A;
```

2) 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 ist, ist in der Variablen "C" vorgegeben.

```
.
DIM A$(512)
.
.
MAT READ #C;A$;
```

10.4.9 "MAT WRITE #"-Anweisung

Übertragen von Daten aus einer String-Variablen, einer numerischen Variablen, einem Vaktor oder einer Matrix in eine Magnetplattendatei.

Syntax:

<MAT WRITE #> ::= MAT WRITE #<N-EXPR1> [,<N-EXPR2> [,<N-EXPR3>]] ; [<M-VAR> | <S-VAR>] [;]

Funktion:

MAT WRITE = = Anweisung.

<N-EXPR1> = Nummer des Kanals, auf dem die Datei eröffnet ist.

<N-EXPR2> = Satznummer des zu schreibenden Datensatzes relativ zum Dateianfang (Satz # 0).
Bei der Bearbeitung von Textdateien ist die Nummer des Blocks (Sektors) relativ zum Dateianfang (Block 0) anzugeben.
Ist keine Satznummer angegeben (<N-EXPR2> nicht codiert), wird die Datei sequentiell bearbeitet. Ebenso, wenn <N-EXPR2> = -1 ist.

Ist <N-EXPR2> = -2, wird der Satz geschrieben, der in dieser Datei zuletzt bearbeitet (gelesen oder geschrieben) wurde, z.B.: den zuletzt gelesenen und geänderten Satz zurückschreiben (updating).

<N-EXPR3> = Die Adresse des ersten zu übertragenden Bytes, relativ zum Anfang des Datensatzes (Byte # 0).
Die Übertragung erfolgt wortorientiert. Aus diesem Grund sollte unter <N-EXPR3> eine geradzahlige Byte-Adresse angegeben werden.
Wird eine ungerade Byte-Adresse ange-

geben, beginnt die Übertragung bei der nächsthöheren geradzahligen Adresse.

<M-VAR>
<S-VAR> = Numerische Variable, Matrix oder String-Variable.
Es darf nur ein Variablenname angegeben werden. Der Inhalt der codierten Variablen wird ab der mit <N-EXPR2> und <N-EXPR3> definierten Adresse in die Datei übertragen.
Die Übertragung der Daten erfolgt incl. eventuell vorhandener Grenzzeichen, bis:

- die Variable in ihrer dimensionierten Länge ausgegeben ist.
- das Dateiende erreicht ist (nur bei relativen Dateien).
- nicht mehr genügend Plattenkapazität verfügbar ist.

Bei der Bearbeitung eines Vektors oder einer Matrix wird die Variable in ihrer Gesamtheit incl. Element 0 bzw. Reihe und Spalte 0 übertragen.
Es können mehrere Sätze mit einer Anweisung geschrieben werden.

; = Kennzeichen für Satzsperre.
Wird hinter dem letzten Operanden kein Semikolon codiert, ist der mit dieser Anweisung geschriebene (adressierte) Satz vor dem Zugriff anderer Teilnehmer geschützt. Ist ein Semikolon codiert, erfolgt keine Satzsperre (siehe auch Pkt. 7.3.3, File-Sharing).
Die Satzsperre wird nur für den adressierten Satz wirksam, auch dann, wenn mit einer Anweisung mehrere Datensätze geschrieben werden.

Beschreibung der Anweisungen

Beispiele: 1) 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, wird in der Variablen "C" vorgegeben.

```
.
DIM A$(512)
.
.
MAT WRITE #C,-2;A$;
```

2) Schreiben der Matrix "A" in ihrer Gesamtheit ab der in der Variablen "X" angegebenen Byte-Nummer relativ zu Byte # 0 in Satz 0. Die Nummer des Kanals, auf dem die Datei eröffnet ist, wird in der Variablen "C" vorgegeben.

```
.
DIM A(3,3)
.
.
MAT WRITE #C,0,X;A;
```

Beschreibung der Anweisungen

10.4.10 "SEARCH #"-Anweisung

Bearbeitung der Indexbereiche von Index-Dateien.

Syntax:

<SEARCH #> ::= SEARCH #<N-EXPR1>,<N-EXPR2>,<N-EXPR3>;
<S-VAR>,<N-VAR1>,<N-VAR2>

Funktion:

SEARCH # = Anweisung.

<N-EXPR1> = Nummer des Kanals, auf dem die Datei eröffnet ist.

<N-EXPR2> = Modus. Definiert die auszuführende Funktion. Als Funktionen stehen zur Verfügung:

<N-EXPR2>	Bedeutung
0	Verzeichnis im Dateikennsatz eintragen. Im Dateikennsatz eingetragene Verzeichnisse initialisieren.
1	Übergabe von: - Ordnungsbegriffslänge - Nummer des ersten Datensatzes - Anzahl freier Sätze Durchführung von: - Übergabe der Nummer des ersten freien Datensatzes und Löschen dieses Satzes aus der Liste der freien Sätze mit Sperren dieses Datensatzes. - Eintragen einer vorgegebenen Satznummer in die Liste der freien Sätze in Abhängigkeit von <N-EXPR3> oder <N-VAR2> unter Beachtung des Sperrkennzeichens.

<N-EXPR2>	Bedeutung
2	Suchen eines Ordnungsbegriffes im angegebenen Verzeichnis. Satzsperre wird gesetzt und ausgewertet.
3	Im angegebenen Verzeichnis den gegenüber in <S-VAR> angegebenen nächst höheren OB suchen. Satzsperre wird gesetzt und ausgewertet.
4	Einfügen eines Ordnungsbegriffs in das angegebene Verzeichnis. Falls notwendig, wird das Verzeichnis reorganisiert.
5	Löschen eines Ordnungsbegriffs in dem angegebenen Verzeichnis unter Beachtung einer Satzsperre.
6	Löschen einer Indexdatei. Die Datensätze werden nicht gelöscht, sondern nur neu verkettet.
7	Diese Funktion entfällt

<N-EXPR3> = Definiert die Nummer des zu bearbeitenden Schlüsselverzeichnisses (1-15). Die Angabe <N-EXPR3> = 0 ist dann zugelassen, wenn <N-EXPR2> = 0 oder 1 ist.

<S-VAR> = String-Variable, in der der Ordnungsbegriff vorgegeben bzw. vom IOCS übergeben wird. <S-VAR> muß für mindestens so viele Bytes dimensioniert sein, wie der Ordnungsbegriff lang ist.

<N-VAR1> = Numerische Variable, mindestens 2%. Diese Variable dient zur Übergabe der relativen Satznummer an das Anwenderprogramm bzw. an das IOCS.

<N-VAR2> = Numerische Variable zur Parameterübergabe an das Betriebssystem und zur Übergabe von Statusmeldungen an das Anwenderprogramm.

Beschreibung der Anweisungen

Die Funktionen (Modi) von SEARCH #

<N-EXPR2> = 0 und <N-EXPR3> <> 0
Das angegebene Verzeichnis wird mit der in <N-VAR1> vorgegebenen OB-Länge im Dateikennsatz eingetragen. Werden für eine Datei mehrere Verzeichnisse definiert, müssen die Verzeichnisnummern lückenlos aufsteigend mit 1 beginnend vorgegeben werden.

<N-EXPR2> = 0 und <N-EXPR3> = 0
Sämtliche Verzeichnisse, die mit Search Modus 0, <N-EXPR3> <> 0 angelegt wurden, werden initialisiert.
Folgt auf einen Search Modus 0, mit <N-EXPR3> = 0 abermals ein Search Modus 0, wird als Status der Wert 8 übergeben.

<N-EXPR2> = 1 und <N-EXPR3> <> 0
Diese Funktion liefert die Ordnungsbegrifflänge des unter <N-EXPR3> angegebenen Verzeichnisses. Die OB-Länge wird in <N-VAR2> übergeben. Ist das angegebene Verzeichnis nicht vorhanden, wird <N-VAR2> auf 0 gesetzt.

<N-EXPR2> = 1 und <N-EXPR3> = 0 und <N-VAR2> = 0.
Diese Funktion übergibt an <N-VAR1> die relative Satznummer des ersten Datensatzes im Datenbereich.

<N-EXPR2> = 1 und <N-EXPR3> = 0 und <N-VAR2> = 1.
Diese Funktion übergibt an <N-VAR1> die Anzahl freier Datensätze.

<N-EXPR2> = 1 und <N-EXPR3> = 0 und <N-VAR2> = 2.
Diese Funktion übergibt an <N-VAR1> die relative Satznummer des ersten freien Datensatzes und löscht diesen Satz aus der Liste der freien Datensätze. Gleichzeitig wird der übergebene Datensatz gesperrt.

<N-EXPR2> = 1 und <N-EXPR3> = 0 und <N-VAR2> = 3.
Diese Funktion übernimmt aus <N-VAR1> eine relative Satznummer und trägt den Satz in die Liste der freien Sätze ein. Der Datensatz wird erst dann zurückgegeben, wenn kein anderer Teilnehmer den Satz gesperrt hat.

<N-EXPR2> = 2 Diese Funktion sucht in dem angegebenen Verzeichnis den unter <S-VAR> angegebenen Ordnungsbegriff. Die Anzahl Zeichen, die verglichen werden, wird immer von <S-VAR> bestimmt.
Es wird verglichen, bis:
- das dimensionierte Ende von <S-VAR> erreicht ist oder
- ein Grenzzeichen in <S-VAR> auftritt.

Gleichheit wird erkannt, wenn alle verglichenen Zeichen des OB in <S-VAR> mit denen im Verzeichnis übereinstimmen.
Dieses Verhalten kann auch dann zu Gleichheit führen, wenn der in <S-VAR> vorgegebene OB kürzer ist als der im Verzeichnis stehende. Für den Fall, daß die OBs, die in <S-VAR> vorgegeben werden, verschiedene Länge haben, sollte <S-VAR> nach rechts mit Blanks aufgefüllt werden.
Das Verfahren, nur bis zu dem ersten auftretenden Grenzzeichen zu vergleichen, bietet die Möglichkeit, Teilordnungsbegriffe vorzugeben.
Bei Gleichheit wird der gesamte OB aus dem Verzeichnis in <S-VAR> übergeben und in <N-VAR1> steht die relative Satznummer des zugehörigen Datensatzes.
Wird kein dem in <S-VAR> entsprechender OB im Verzeichnis gefunden, bleiben <S-VAR> und <N-VAR1> unverändert und als Status wird an <N-VAR2> der Wert 1 übergeben.
Durch Search Modus 2 werden Satzsperren gesetzt und auch ausgewertet.

<N-EXPR2> = 3 Diese Funktion sucht im angegebenen Verzeichnis den nächstgrößeren OB gegenüber dem in <S-VAR> vorgegebenen.
Ist ein größerer OB gefunden, wird dieser an <S-VAR> übergeben.
In <N-VAR1> wird die relative Satznummer des dazugehörigen Datensatzes abgestellt.
Ist kein größerer OB vorhanden, bleiben <S-VAR> und <N-VAR1> unverändert. Als Statusmeldung wird in <N-VAR2> der Wert 2 übergeben.
Durch Search Modus 3 werden Satzsperren gesetzt und auch ausgewertet.

Beschreibung der Anweisungen

<N-EXPR2> = 4 Diese Funkion fügt den in <S-VAR> vorgegebenen OB in das angegebene Verzeichnis ein. Der OB darf in dem angegebenen Verzeichnis noch nicht vorhanden sein. Gleichheit wird erkannt, wenn <S-VAR> incl. Grenzzeichen identisch mit dem OB im Verzeichnis ist.
Die relative Satznummer des Datensatzes, auf den der OB verweisen soll, muß in <N-VAR1> vorgegeben werden, so daß eine Verkettung zwischen OB und Datensatz vorgenommen werden kann. Konnte der vorgegebene OB in das Verzeichnis eingefügt werden, wird als Statusmeldung in <N-VAR2> der Wert 0 übergeben. Ist der OB im angegebenen Verzeichnis bereits vorhanden, wird in <N-VAR1> die relative Satznummer übergeben, auf den der vorhandene OB verweist. Als Statusmeldung wird in <N-VAR2> der Wert 1 übergeben.
Besteht beim Einfügen eines OB die Notwendigkeit einer Reorganisation, wird diese automatisch durchgeführt.

<N-EXPR2> = 5 Diese Funktion löscht den in <S-VAR> vorgegebenen OB aus dem angegebenen Verzeichnis. Gleichheit wird erkannt, wenn <S-VAR> incl. Grenzzeichen identisch mit dem OB im Verzeichnis ist.
Ist der vorgegebene OB im Verzeichnis vorhanden, wird er gelöscht und an <N-VAR1> die relative Satznummer des zugehörigen Datensatzes übergeben.
<N-VAR2> wird auf 0 gesetzt.
Ist der vorgegebene OB nicht im Verzeichnis vorhanden, bleiben <S-VAR> und <N-VAR1> unverändert. Als Statusmeldung wird in <N-VAR2> der Wert 1 übergeben.
Search Modus 5 nimmt eine Auswertung der Satzsperre vor.

<N-EXPR2> = 6 Dieser Search Modus ermöglicht das Löschen einer Indexdatei. Der Verzeichnisbereich hat nach der Ausführung des Modus 6 den gleichen Zustand wie nach der Neuanlage mit BUILDXF. Die Datensätze werden nicht gelöscht, sondern nur neu verkettet.
Diese Funktion gibt keine Zeit an andere Teilnehmer ab.

Ist die Datei von einem anderen Teilnehmer eröffnet, wird Status 15 gemeldet. Für diese Funktion wird nur <N-EXPR1> und <N-EXPR2> ausgewertet. In <N-VAR1> wird nach erfolgreicher Durchführung der Funktion die Anzahl der freien Datensätze übergeben.

<N-EXPR2> = 7 Durch die Erweiterung des Search Modus 4 ist Search Modus 7 überflüssig geworden. Ein Absetzen dieses Search Modus bleibt für die Datei ohne Auswirkung. Die Status-Variable wird auf 0 gesetzt.

Statusmeldungen von SEARCH #

Nach der Durchführung einer SEARCH - Anweisung ist die Statusvariable <N-VAR2> unbedingt abzuprüfen. Folgende Statusmeldungen können in <N-VAR2> abgestellt sein:

<N-VAR2>	Bedeutung
0	Funktion fehlerfrei durchgeführt.
1	Funktion nicht erfolgreich durchgeführt.
2	Ende des Verzeichnisses erreicht.
3	Kein freier Datensatz vorhanden.
4	Keine Index-Datei.
5	Unbestimmter Fehler. Tritt dieser Fehler auf, kann davon ausgegangen werden, daß die Datei zerstört ist. Weiterarbeiten nicht ratsam.
6	Unzulässige Reihenfolge bei Angabe der Verzeichnisse (nur Modus 0).
7	Datei ist keine relative Datei (nur Modus 0).
8	Modus 0, <N-VAR1> = 0 wurde bereits ausgeführt (nur Modus 0).
10	Zu viele Verzeichnisse (max. 15 pro Datei).
12	Datei ist zu klein, um alle Verzeichnisse aufzunehmen (nur Modus 0, <N-VAR1> = 0).
15	Datei von anderem Teilnehmer eröffnet.

Beschreibung der Anweisungen

10.4.11 "BUILD #"-Anweisung

Erstellen von neuen bzw. Ersetzen bereits vorhandener Dateien. Es wird Platz auf der Magnetplatte reserviert, der Dateiname ins Inhaltsverzeichnis (Datei Index) eingetragen und der Datei-Kennsatz angelegt. Die benötigen Plattenblöcke werden in der Plattenbelegungsliste als belegt eingetragen.

Syntax:

<BUILD #> ::= BUILD #<N-EXPR>, [+] { <S-VAR> / <S-LIT> }

[[,#<N-EXPR>] , [+] { <S-VAR> / <S-LIT> }] n 1

Funktion:

BUILD #	= Anweisung
<N-EXPR>	= Nummer des Kanals, auf dem die Datei erstellt werden soll.
+	= Wird "+" codiert, ist die zu erstellende Datei eine Text-Datei.
<S-VAR> bzw. <S-LIT>	= Bezeichnet die zu erstellende Datei. Die gesamten Angaben können in einer String-Variablen (<S-VAR>) oder als String-Literal (<S-LIT>) übergeben werden.

Beschreibung der Anweisungen

<S-VAR> bzw. <S-LIT> haben folgenden Aufbau:

[<PP>] [COST] [(SA:SL)] [LU/] DATEINAME [!]

<PP> = Schutzstufen-Zuordnung (Protection). Die beiden spitzen Klammern gehören zur Angabe und müssen codiert sein! Für Text-Dateien ist diese Angabe nicht zulässig.

<PP>
- 1 — Benutzer derselben Privilegebene
- 2 — Benutzer niedriger Privilegebene

Eine Schutzmöglichkeit gegen Benutzer höherer Privilegebenen besteht nicht.

Folgende Möglichkeiten des Schutzes bestehen:

P = 0 = kein Schutz

1 = Kopierschutz

2 = Schreibschutz

3 = Kopier- und Schreibschutz

4 = Leseschutz

5 = Lese- und Kopierschutz

6 = Lese- und Schreibschutz

7 = Lese, Schreib- und Kopierschutz

Ist keine Schutzstufe angegeben, wird intern "77" eingetragen.

COST = Der Betrag, mit dem das Konto eines anderen Benutzers für den Zugriff auf diese Datei belastet wird. Für Text-Dateien ist diese Angabe nicht zulässig.

Beschreibung der Anweisungen

Folgende Form muß eingehalten werden:

$DDD.PP

$ muß codiert sein
DDD Betrag vor dem Komma
PP Betrag nach dem Komma

Der angegebene Nachkomma-Betrag wird immer auf einen vollen 10er-Wert abgerundet.
Ist "COST" nicht angegeben, wird 0 eingetragen.

(SA:SL) = Angabe der Datensätze und Satzlänge.

SA = Anzahl Datensätze, für die Platz auf der Magnetplatte reserviert werden soll.

SL = Satzlänge in Worten (1 Wort = 2 Byte).

Diese beiden Angaben stehen in Klammern und sind durch ":" voneinander getrennt.
Wird diese Angabe in einer "BUILD #"-Anweisung gemacht, wird Platz für eine relative Datei reserviert.

LU/ = LU-Nummer der Platte, auf der die Datei erstellt werden soll.
Ist LU/ = 0, kann diese Angabe entfallen. Die LU-Nummer wird durch "/" vom Dateinamen getrennt.

DATEINAME = Der Name, unter dem die Datei auf einer Magnetplatte angelegt, bzw. der Name der Datei, die ersetzt werden soll.

! = Muß dann angegeben werden, wenn eine bereits bestehende Datei ersetzt werden soll.

Anmerkung: Wird eine Datei ersetzt, ist zu beachten, daß die zu ersetzende Datei erst dann gelöscht wird, wenn die neue Datei (die sie ersetzt), bereits angelegt ist.

Beschreibung der Anweisungen

Mit einer "BUILD #" - Anweisung können mehrere Dateien erstellt werden. Wird diese Möglichkeit genutzt 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 (Kanalnummer der letzten mit dieser Anweisung erstellten Datei + 1).

Mit "BUILD #" erstellte Dateien sind gleichzeitig für die Verarbeitung eröffnet.

Mit "BUILD #" erstellte Dateien sind, bevor sie durch das erstellende Programm mit der Anweisung "CLOSE #" geschlossen werden, vor dem Zugriff anderer Teilnehmer geschützt.

Eine mit "BUILD #" angelegten Datei wird gelöscht, wenn das Anwenderprogramm beendet ist und der Kanal, auf dem die Datei erstellt wurde, nicht vorher mit der Anweisung "CLOSE #" geschlossen oder das Programm mit der Anweisung "END" beendet wurde.

Beispiele:

A) Erstellen einer relativen Datei auf Kanal 2. Alle Parameter sind als <S-LIT> in der Anweisung angegeben.

- Schutzstufe = 33
- Kosten = 0
- Anzahl Sätze = 100
- Satzlänge = 50 (Worte)
- LU-Nummer = 9
- Dateiname = TEST
- Neuanlage

```
BUILD #2,"<33>(100:50)9/TEST"
```

B) Erstellen mehrerer Dateien mit einer "BUILD #" - Anweisung:

```
BUILD #2,+A$,"<00>$10.00(100:50)TEST!",B$,#6,C$
```

Erläuterung:

- Erstellen/Ersetzen einer Text-Datei, deren Bezeichnung in der Variablen A$ abgestellt ist, auf Kanal 2
  - Ersetzen der relativen Datei TEST auf Kanal 3. Die Dateibezeichnung ist als <S-LIT> angegeben.
  - Erstellen oder 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.

Anmerkung: Der Nachteil des Erstellens mehrerer Dateien mit einer "BUILD #" - Anweisung ist eine unzureichende Fehlerauswertung. Tritt bei der Ausführung einer solchen Anweisung ein Fehler auf, kann nicht festgestellt werden, bei welcher Datei der Fehler entstanden ist.

10.4.12 "KILL" - Anweisung

Löschen von Anwenderdateien auf einer Magnetplatte. "KILL" löscht die ersten zwei Zeichen des Dateinamens im Inhaltsverzeichnis (Datei INDEX). Die von der Datei belegten Plattenblöcke werden in der Plattenbelegungsliste als frei eingetragen.

Syntax:

<KILL> ::= KILL { <S-LIT> / <S-VAR> } [, { <S-LIT> / <S-VAR> }]n 1

Funktion:

KILL = Anweisung.

<S-LIT>
<S-VAR> = Beliebige Anzahl von String-Variablen und String-Literalen, welche die zu löschende(n) Datei(en) bezeichnet. <S-LIT> und <S-VAR> haben folgenden Aufbau:

[LU/]<DATEINAME>

LU/ = LU-Nummer der Platte, auf der sich die zu löschende Datei befindet. Ist LU = 0, kann diese Angabe entfallen. Die LU-Nummer wird durch "/" vom Dateinamen getrennt.

<DATEINAME> = Name der zu löschenden Datei.

Wird mit "KILL" eine Datei gelöscht, die durch einen anderen Teilnehmer oder auch durch das löschende Programm eröffnet ist, wird 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, die Datei schließt.

Mit einer "KILL" - Anweisung können mehrere Dateien gelöscht werden. Wird diese Möglichkeit genutzt, entsteht das Problem einer unzureichenden Fehlerauswertung, denn

tritt bei der Ausführung einer solchen Anweisung 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.

```
.
KILL A$
```

Beschreibung der Anweisungen

10.5 Spezielle Anweisungen

Beschreibung der Anweisungen:

- CALL
- SIGNAL 1
- SIGNAL 2
- SIGNAL 3
- REM

Zusätzlich ist die Möglichkeit der Angabe von Kommentaren in Basic - Anweisungen beschrieben.

Beschreibung der Anweisungen

10.5.1 "CALL" - Anweisung

Aufruf und Starten von Maschinencode-Unterprogrammen. 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> ::= CALL <N-EXPR> [, { <S-VAR> / <N-VAR> }] (1 bis 12)

Funktion:

CALL = Anweisung.

<N-EXPR> = Nummer des aufzurufenden Unterprogrammes. Eine Liste der verfügbaren Funktionen ist im Anhang vorhanden.

<N-VAR>
<S-VAR> = Maximal 12 Variablennamen. Diese Variablen dienen zur Vorgabe von Parametern an das Unterprogramm bzw. zum Austausch von Informationen zwischen dem Business-Basic-Programm und dem Unterprogramm.

Die Adressierung von Sub-Strings (z.B.: A$(X,Y)) ist nicht möglich. Elemente von Vektoren und Matrizen können nur adressiert werden, wenn sie vorher dimensioniert wurden.

Auf Einschränkungen/Bedingungen, die nicht allgemein gültig sind, wird bei der Beschreibung der zur Verfügung stehenden Unterprogramme (Pkt. 11.) hingewiesen.

10.5.2 "SIGNAL 1" - Anweisung

Überstellen von drei numerischen Werten in die Signal-Liste.

Syntax:

<SIGNAL 1> ::= SIGNAL 1,<N-EXPR1>,<N-EXPR2>,<N-EXPR3>

Funktion:

SIGNAL 1 = Anweisung.

<N-EXPR1> = BA-Nummer des adressierten Teilnehmers.

<N-EXPR2> = In die Signal-Liste zu überstellenden
<N-EXPR3> numerischen Werte.
Es darf kein Wert > 32767 übergeben werden.

Die Werte von <N-EXPR1>, <N-EXPR2>, <N-EXPR3> und die Nummer des sendenden Arbeitsplatzes werden in die Signal-Liste überstellt.
Die Werte bleiben solange in der Signal-Liste, bis vom adressierten Teilnehmer eine Anweisung "SIGNAL 2" ausgeführt wird.
Ist eine Teilnehmernummer angegeben, die nicht konfiguriert ist, oder sind in der Signal-Liste bereits vier Einträge vorhanden, wird Basic-Fehler # 62

SIGNALPUFFER VOLL/PORT-NR. NICHT VORHANDEN

gemeldet.
Ein Eintrag in der Signal-Liste wird gestrichen, wenn er ca. zwei Stunden in der Liste steht, ohne daß eine SIGNAL 2 - Anweisung ausgeführt wurde.

Beispiel: Überstellen der Werte 0 und 10 in die Signal-Liste. Adressiert wird der Masterplatz.

```
.
.
LET P=0            /* ADR. ARBEITSPLATZ
LET X=0            /* <N-EXPR2>
LET Y=10           /* <N-EXPR3>
SIGNAL 1,P,X,Y
.
```

10.5.3 "SIGNAL 2" - Anweisung

Übertragen (empfangen) numerischer Werte aus der Signal-Liste in numerische Variable.

Syntax:

<SIGNAL 2> ::= SIGNAL 2,<N-VAR1>,<N-VAR2>,<N-VAR3>[,<N-EXPR>]

Funktion:

SIGNAL 2 = Anweisung.

<N-VAR1> = Enthält nach Ausführung der Anweisung die BA-Nummer des Teilnehmers, von dem das "Signal" in die Signal-Liste überstellt wurde.
Liegt kein Signal vor, enthält <N-VAR1> den Wert -1.

<N-VAR2>
<N-VAR3> = Enthalten nach Ausführung der Anweisung, die mittels "SIGNAL 1" in die Signal-Liste überstellten numerischen Werte.
Liegt kein Signal vor, bleiben die Werte unverändert.

<N-EXPR> = Ein Wert von +1 bis +32767.
Es wird ein Zeitraum in Zehntelsekunden angegeben, für den auf die Übermittlung eines "Signals" gewartet wird.
Wird ein Signal empfangen, bevor die Zeit abgelaufen ist, läuft das Programm weiter.

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
.
```

10.5.4 "SIGNAL 3" - Anweisung

Das Programm wird für einen anzugebenden Zeitraum in einen Wartezustand versetzt und ein Task-Wechsel durchgeführt. Bis zum Ablauf der angegebenen Zeit erhält das Programm keine Zeitscheibe.

Syntax:

<SIGNAL 3> ::= SIGNAL 3, <N-EXPR>

Funktion:

SIGNAL 3 = Anweisung.

<N-EXPR> = Ein Wert im Bereich von 0 bis +32767. Dieser Wert repräsentiert einen Zeitraum in Zehntelsekunden, für den das Programm in einen Wartezustand versetzt wird.

Diese Anweisung bewirkt die Ausgabe des Ein/Ausgabepuffers des jeweiligen Teilnehmers.

Beispiel: Ausgabe einer Fehlermeldung ab Position 15 in der Zeile 24. Nach 30 Sekunden wird die Fehlermeldung gelöscht, und das Programm arbeitet weiter.

```
.
9000 PRINT TAB(15,24);M$;
9010 SIGNAL 3,300
9020 PRINT 'LD';
.
.
```

Beschreibung der Anweisungen

10.5.5 "REM" - Anweisung

Einfügung von Kommentarzeilen in ein Basic-Programm.
Der Kommentar in "REM" - Anweisungen belegt zur Programm-Laufzeit soviel Speicherplatz, wie seiner Länge in Byte + 5 entspricht. Außerdem wird die "REM" - Anweisung interpretiert wie jede andere Anweisung und wirkt sich dadurch auf die Programm-Laufzeit aus.
Aus diesen Gründen sollten zum Ablauf kommende Programme keine überflüssigen "REM" - Anweisungen enthalten!

Syntax:

<REM> ::= REM [<ZCHN>]

Funktion:

REM = Anweisung

<ZCHN> = Beliebige Anzahl von Zeichen, die als Kommentar dienen.

Beispiel: 100 REM ***** ALLGEMEINE FEHLERMELDUNG *****

Eine weitere Möglichkeit der Angabe von Kommentaren in Basic-Programmen wird unter Pkt. 10.5.6 beschrieben.

10.5.6 Kommentare in Anweisungen

Zusätzlich zur Möglichkeit, Kommentare mit der "REM" - Anweisung im Programm zu definieren, kann auch hinter jeder beliebigen Anweisung ein Kurzkommentar angefügt werden.
Der Kommentar wird durch die Zeichenfolge /* von dem letzten Operanden der Anweisung getrennt.

Bei Ausgabe des Basic-Programmes mit "LIST" und "DUMP" werden zwischen der Anweisung und der Zeichenfolge /* immer zwei Leerzeichen ausgegeben.

Bezüglich Speicherbedarf gilt das gleiche wie unter Pkt. 10.5.5 für die Anweisung "REM".

Beispiel: 10 IF ERR 0 GOSUB 9000 /*FEHLERVERZWEIGUNG

Anhang CALL-Unterprogramme

11 CALL-Unterprogramme

CALL-Unterprogramme sind Maschinencode-Unterprogramme die mit der Anweisung "CALL" aufgerufen werden können. 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> ::= CALL <N-EXPR> [, { <S-VAR> / <N-VAR> }] (1 ... 12)

Funktion:

CALL = Anweisung.

<N-EXPR> = Nummer des aufzurufenden Unterprogrammes. Die verfügbaren Unterprogramme sind unter Pkt.: 11.1 aufgeführt.

<N-VAR>
<S-VAR> = Maximal 12 Variablennamen. Diese Variablen dienen zur Vorgabe von Parametern an das Unterprogramm bzw. zum Austausch von Informationen zwischen dem Business-Basic Programm und dem Unterprogramm.

Anhang CALL-Unterprogramme

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 (Pkt.: 11.1 bis 11.26) hingewiesen.

Fehler:

Basic-Fehler	Bedeutung
# 36	Aufgerufenes Unterprogramm ist nicht vorhanden.
# 37	Statt <S-VAR> oder <N-VAR> ist ein Ausdruck angegeben.
# 38	Fehler vom aufgerufenen Unterprogramm erkannt.

Anhang CALL-Unterprogramme

11.1 Übersicht über die verfügbaren Unterprogramme

UPRO-Nummer	Funktion
1	Bildschirmaufbereitung für eine Eingabe, Prüfung eingegebener Daten und Formatierung.
2	Transport des Inhalts von angegebenen Variablen in den gemeinsamen Bereich der angegebenen Task.
3	Transport von Daten aus dem gemeinsamen Bereich der Task in die angegebene Variable.
4	Ausgabe von Nachrichten am Masterplatz oder Schreiben/Lesen eines Platten-Kennzeichens (ID)
20	Lesen, Setzen oder Löschen eines Bits in einem String.
21	Umwandlung des Inhalts numerischer Variabler in Dualzahlen.
22	Umwandlung von Dualzahlen 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.
51	Unterstützung Bildschirm-Programmierung.
60	Packen eines Strings.
61	Entpacken eines Strings.
62	Prüfung, ob der Inhalt eines Strings numerisch ist.

Anhang CALL-Unterprogramme

UPRO-Nummer	Funktion
63	Prüfung, ob der Inhalt eines Strings ein gültiger arithmetischer Wert ist.
64	Prüfen und Umstellen des Inhalts eines Strings. Der Inhalt des Strings muß ein gültiges Datum enthalten.
65	Speichersortierung.
70	Magnetband-Programmierung.
72	Übertragung des Inhalts von n Variablen in einem String.
73	Übertragung des Inhalts eines Strings in n Variable.
80	MBC - Behandlung.
90	Bearbeitung von String-Variablen.
91	Invertieren von Sub-Strings.
97	Informationen aus Dateikennsätzen von MPL lesen.
98	Systemkommando senden.
99	Datum und Uhrzeit übernehmen/ändern.

Anhang CALL-Unterprogramme

11.1.1 CALL 1

Bildschirmaufbereitung für eine Eingabe, Prüfung eingegebener Daten und Formatierung. Format 1 wird zur Bildschirmaufbereitung, Format 2 zur Prüfung und Formatierung der Eingabedaten benutzt.

Syntax Format 1:

<CALL> ::= CALL <N-EXPR> ,<N-VAR1> ,<S-VAR1>
,<S-VAR2> ,<N-VAR2> ,<S-VAR3>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 1 sein.

<N-VAR1> = Funktionsnummer muß 0 sein; nach Ausführung des CALL enthält die Variable den Wert 1.

<S-VAR1> = Parameterstring zur Steuerung der Operation (siehe Beschreibung); Größe des Strings 12 oder 13 Bytes.

<S-VAR2> = Zeichenmaske für das Eingabefeld. Ist in S-VAR1 der Parameter für Feld Duplizieren gesetzt, so wird der Inhalt von S-VAR2 auf der Position des Eingabefeldes ausgegeben. Ist in S-VAR1 zusätzlich spezifiziert, daß das Eingabeformat 'numerisch Ungepackt' ist, wird vor der Ausgabe der Inhalt von N-VAR2 nach S-VAR2 übertragen.

<N-VAR2> = Numerischer Wert für Felddublizierung bei Eingabe ungepackt numerisch (siehe auch S-VAR2).

<S-VAR3> = Arbeitsfeld für CALL 1; Dimension >= 18 Bytes. Der Inhalt ist nach der Operation undefiniert.

Anhang CALL-Unterprogramme

Falls eine Maske ausgegeben wird, wird hinter das letzte Maskenzeichen ein Hintergrundblank ausgegeben, um Eingaben abzufangen, die länger als die spezifizierte maximale Eingabefeldlänge sind.

Syntax Format 2:

```
<CALL>      ::= CALL <N-EXPR> ,<N-VAR1> ,<S-VAR1>
                    ,<S-VAR2> ,<N-VAR2> ,<S-VAR3>
                   ,<S-VAR4> ,<S-VAR5>
```

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 1 sein.

<N-VAR1> = Funktionsnummer muß 1 sein. Die Variable enthält nach Ausführung den Status der Operation:
-1 =Eingabe war nicht korrekt
0 =Eingabe war korrekt
1 =Taste % für Felddupliziерung wurde betätigt
2 =Taste Reverse für Feldzurücksetzen wurde betätigt
n =n. 3-Zeichen-Funktion aus S-VAR3 wurde eingegeben

<S-VAR1> = Parameterstring zur Steuerung der Operation (siehe Beschreibung); Größe 12 oder 13 Bytes.

<S-VAR2> = Eingabestring; Dimension >= maximale Eingabelänge. Der Inhalt dieser Variablen wird nach der Eingabe gegen die Steuerparameter in S-VAR1 überprüft und formatiert. Bei korrekter Eingabe werden:
Numerische Eingaben rechtsbündig mit führenden Nullen im Zeichenformat abgestellt.
Alphanumerische Eingaben linksbündig abgestellt und eventuell rechts mit Blanks bis zur maximalen Eingabelänge aufgefüllt.
Der formatierte String auf Bildschirm ausgegeben, jedoch mit Vornullenunterdrückung.

<N-VAR2> = Numerische Eingaben werden zusätzlich in diese Variable übertragen.

<S-VAR3> = Enthält in jeweils 3 Byte die Zeichenfolge, die als Steuerinformation interpretiert werden soll.

<S-VAR4> = Gibt in Form einer Maske an, welche Zeichenfolgen in S-VAR3 bei dieser Eingabe zulässig sind und ob die Funktionen 'Duplizieren' (% Taste) und/oder 'Reverse' betätigt werden dürfen. Jedes Byte in S-VAR4 steuert eine Funktion, wobei jeweils der Wert 0 aussagt, daß die Funktion erlaubt ist. Die Zuordnung der Maske zu den Funktionen ist wie folgt:
Byte 1 =Dupliziertaste %
Byte 2 =Reverse Taste
Byte n =(n-2). Zeichenfolge in S-VAR3

<S-VAR5> = Arbeitsfeld für CALL 1; Dimension >= 18 Bytes. Der Inhalt ist nach der Ausführung undefiniert.

Falls ein Fehler auftritt, wird die entsprechende Meldung aus der Datei gelesen, die auf Kanal 1 (TF.PARAM) eröffnet ist (Messages 170 bis 176).
Nach Ausführung von CALL 1 ist die gemerkte Cursorposition gleich der ersten Eingabefeldposition.

Anhang CALL-Unterprogramme

Beschreibung der Steuerparameter

Byte	Wert	Bedeutung
1	0	Keine Eingabemaske ausgeben.
	1	Eingabemaske in Form von Punkten ausgeben.
	2	Wie bei 1, jedoch zuvor 'Clear Foreground'.
	3	Eingabemaske aus Duplizierfeld (S-VAR2/ N-VAR2) ausgeben. Dieses Byte wird bei Format 2 auf 0 gesetzt, wenn ein Fehler auftritt.
2	0	Numerische Eingabe mit anschließendem Packen (Format siehe CALL 60/61).
	1	Alphanumerische Eingabe.
	2	Numerische Eingabe.
3-4	XX	Spaltennummer der Cursorposition.
5-6	XX	Zeilennummer der Cursorposition.
7-8	XX	Max. Anzahl VK-Stellen bzw. Zeichen.
9-10	XX	Min. Anzahl VK-Stellen bzw. Zeichen.
11	X	Anzahl NK-Stellen.
12	0	Kein führendes Vorzeichen erlaubt.
	1	Vorzeichen erlaubt.
13	0	Falls Eingabefeld nur 1-2 Zeichen, 1-2 Blanks ergänzen vor dem Hintergrundblank, um ggf. 3-Zeichen-Steuerinformationseingabe zu ermöglichen.
	1	Keine Blanks freihalten. Ist S-VAR1 auf 12 Bytes dimensioniert, wird 0 angenommen.

Fehler:

Basic-Fehler	Bedeutung
# 38	- Unzulässiger Variablentyp oder -Dimensionierung - TF.PARAM auf Kanal 1 nicht eröffnet - Unzulässige Werte im Parameterstring - Maske für Eingabefeld zu lang - <N-VAR2> ist 1%; die Eingabe >7999 oder <-7999 - Duplizieren bei gepackter Eingabeformatierung nicht zulässig

Anhang CALL-Unterprogramme

11.1.2 CALL 26

Selektive Zeichenersetzung in Stringvariablen.

Syntax:

<CALL> ::= CALL <N-EXPR> ,<N-VAR> ,<S-VAR1>
,<S-VAR2> ,<S-VAR3> ,<S-VAR4>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 26 sein.

<N-VAR> = Funktionscode (wird zur Zeit nicht ausgewertet).

<S-VAR1> = Quell-Zeichenkette.

<S-VAR2> = Ziel-Zeichenkette.

<S-VAR3> = Zeichenkette für Suchzeichen.

<S-VAR4> = Zeichenkette für Ersetzungszeichen.

Quell- und Ziel-Zeichenkette müssen gleich dimensioniert sein. <S-VAR4> muß die gleiche Größe wie <S-VAR3> haben, wobei <S-VAR3> nicht größer als 256 Bytes sein darf.

Der Inhalt der Quell-Zeichenkette wird zeichenweise in die Ziel-Zeichenkette konvertiert. Dabei wird jedes Quellzeichen mit dem Suchzeichen-String verglichen. Herrscht keine Übereinstimmung, wird das Quellzeichen auf die entsprechende Stelle in der Ziel-Zeichenkette abgestellt. Herrscht zwischen Quell- und Suchzeichen Übereinstimmung, wird das Ersetzungszeichen auf die entsprechende Stelle in der Ziel-Zeichenkette abgestellt.

11.2 CALL 2

Transport des Inhaltes von bis zu 11 Variablen (<S-VAR>, <N-VAR>) in den gemeinsamen Bereich eines beliebigen Teilnehmers.

Syntax:

<CALL> ::= CALL <N-EXPR>[,<N-VAR>][,{<S-VAR>|<N-VAR>}]1 11

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes. <N-EXPR> muß in diesem Fall = 2 sein.

<N-VAR> = 1 bis 12 Variable.
<S-VAR> Die erste codierte Variable muß <N-VAR> sein und die BA-Nummer des Teilnehmers enthalten, in dessen gemeinsamen Bereich die Inhalte der im folgenden codierten Variablen übertragen werden sollen.
Die BA-Nummer des adressierten Teilnehmers wird nicht übertragen. Es kann also der Inhalt von max. 11 Variablen übertragen werden.

Die dimensionierte Gesamtlänge aller zu übertragenden Variablen darf 512 Byte nicht überschreiten.

Beispiel: Übergabe des Inhaltes der Variablen B und A$ in den gemeinsamen Bereich der Phantom-Task (BA-Nr. = 1).

```
REM  DATEN AN PHANTOMTASK ÜBERGEBEN
LET  T=1
CALL 2,T,B,A$
```

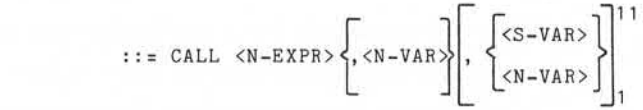
Anhang-CALL-Unterprogramme

Fehler:	Basic-Fehler	Bedeutung
	# 38	- Die dimensionierte Gesamtlänge der zu übertragenden Variablen ist > 512 Byte. - Die angegebene BA-Nummer ist nicht gültig.

11.3 CALL 3

Transport von Daten aus dem gemeinsamen Bereich eines beliebigen Teilnehmers in bis zu 11 Variablen.

Syntax:



Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes. <N-EXPR> muß in diesem Fall = 3 sein.

<N-VAR>
<S-VAR> = 1 bis 12 Variable. Die erste codierte Variable muß <N-VAR> sein und die BA-Nummer des Teilnehmers enthalten, aus dessen gemeinsamen Bereich Daten übernommen werden sollen. Die Daten werden 1:1 übertragen. Stimmen die angegebenen Variablen in Typ und/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 der Phantom-Task in die Variablen B und A$.

```
REM  DATEN VON PHANTOMTASK ÜBERNEHMEN
LET  T=1
CALL 3,T,B,A$
```

Anhang-CALL-Unterprogramme

Fehler:

Basic-Fehler	Bedeutung
# 38	- Die dimensionierte Gesamtlänge der aufnehmenden Variablen ist > 512 Byte. - Die angegebene BA-Nummer ist nicht gültig.

11.4 CALL 4

Dieses Unterprogramm dient zur Ausführung der Funktionen:

- Ausgabe von Nachrichten am Masterplatz
  oder
- Schreiben/Lesen eines Plattenkennzeichens (ID).

11.4.1 Ausgabe von Nachrichten am Masterplatz

Um jedem Teilnehmer (jeder Task) zu ermöglichen, Nachrichten auf dem Bildschirm des Masterplatzes auszugeben, stellt Business-Basic die Anweisung

```
CALL 4
```

zur Verfügung.

Syntax:

```
<CALL>        :: = CALL <N-EXPR>,<S-VAR>
```

Funktion:

CALL = Anweisung

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 4 sein!

<S-VAR> = String, dessen Inhalt auf dem Bildschirm des Masterplatzes ausgegeben werden soll.

Diese Anweisung wird nur dann korrekt ausgeführt, wenn der Masterplatz im "INPUT-mode" steht und noch keine Zeichen eingegeben worden sind. Jeder andere Zustand des Masterplatzes bewirkt bei der Ausführung des "CALL 4"

Basic-Fehler 38. Die Anweisung ist deshalb so oft zu wiederholen, bis sie erfolgreich ausgeführt werden kann! Da die Ausgabe auf dem Bildschirm des Masterplatzes ab der aktuellen Cursorposition erfolgt, sollte <S-VAR> geeignete Display-Funktionen (z.B. TAB) vor der eigentlichen Nachricht enthalten, um den aktuellen Bildschirminhalt nicht zu zerstören.
Die Bildschirmposition, die auf dem Masterplatz für solche Meldungen benutzt werden können, sind vom Anwender zu definieren. TAMOS sieht die Positionen 66 bis 78 in Zeile 24 vor.

Beispiel: Ausgabe einer Meldung auf Position 15 in Zeile 24 (Nachrichtenzeile) am Masterplatz. Die Ausgabe wird so oft wiederholt, bis die Anweisung fehlerfrei durchgeführt wird. Die Routine ist als Unterprogramm aufgebaut.

```
100  DIM   M$(50)              /* VARIABLE ZUR AUFNAHME DER
                                  NACHR.
          .
          .
5000 LET   M$="←207←←376←←211←←-376←←221←←217←
               ←230←MELDUNG←376←←212←"
5010 IF    ERR 0 GOTO 5050     /* FEHLER # 38??
5020 CALL  4,M$
5030 IF    ERR 0 GOSUB 9000    /* STANDARD-FEHLERBEHANDLUNG
5040 RETURN                    /* RÜCKSPRUNG INS HAUPTPROGR.
5050 IF SPC8 <> 38 GOTO 1010   /* PROGRAMMABBRUCH
5060 SIGNAL 3,100              /* 10 SEKUNDEN WARTEN
5070 GOTO 5020                 /* ERNEUTER VERSUCH
```

Fehler:	Basic-Fehler	Bedeutung
	# 38	- Der Masterplatz befindet sich nicht im Input-Modus.

Anhang-CALL-Unterprogramme

11.4.2 Schreiben/Lesen eines Plattenkennzeichens (ID)

Verarbeitung eines Plattenkennzeichens. Das Kennzeichen ist zwei Byte lang und steht im Kennsatz der Datei "INDEX" auf Adresse 32 (oktal).
CALL 4 ermöglicht sowohl das Lesen als auch das Schreiben des Plattenkennzeichens.

Syntax:

<CALL> :: = CALL <N-EXPR>,<N-VAR>,<S-VAR>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 4 sein!

<N-VAR> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird die physikalische Gerätenummer der Magnetplattenstation vorgegeben, auf die das Plattenkennzeichen geschrieben, bzw. von der das Kennzeichen gelesen wird.

<N-VAR>	Adressierte Magnetplatte
0	Gerät 0, Festplatte
1	Gerät 0, Wechselplatte
2	Gerät 1, Festplatte
3	Gerät 1, Wechselplatte
4	Gerät 2, Festplatte
5	Gerät 2, Wechselplatte
6	Gerät 3, Festplatte
7	Gerät 3, Wechselplatte

<S-VAR> = Beliebige Stringvariable. Dieser String muß für mindestens drei Byte dimensioniert sein.

Schreiben des Kennzeichens:

Ist die Länge von <S-VAR> >0, werden die ersten beiden Byte des Strings auf die Magnetplatte geschrieben, deren Nummer unter <N-VAR> vorgegeben ist.
Lesen des Kennzeichens:

Ist <S-VAR> ein Leerstring, wird das Kennzeichen der Magnetplatte, die durch <N-VAR> adressiert ist, in den String übertragen.

Beispiele: 1) Das Kennzeichen "01" wird auf die Magnetplatte - Gerät 0, Wechselplatte - geschrieben.

```
.
1000 LET U=1        /* GERÄTENUMMER LADEN
1005 LET K$="01"    /* KENNZEICHEN LADEN
1010 CALL 4,U,K$    /* KENNZEICHEN SCHREIBEN
```

2) Das Kennzeichen wird von:
- Gerät 0, Festplatte
gelesen.

```
.
1000 LET U=0        /* GERÄTENUMMER LADEN
1005 LET K$=""      /* K$ = LEERSTRING
1010 CALL 4,U,K$ /* KENNZEICHEN LESEN
.
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Schreib- oder Lesefehler Magnetplatte. - K$ für weniger als drei Byte dimensioniert. - Ungültige Gerätenummer.

Anhang-CALL-Unterprogramme

11.5 CALL 20

Lesen, Setzen oder Löschen eines Bits in einem String. Die Bits werden von links nach rechts beginnend mit "0" durchnumeriert.

Syntax:

<CALL>::= CALL <N-EXPR>,<N-VAR1>,<S-VAR>,<N-VAR2>
[,<N-VAR3>]

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 20 sein.

<N-VAR1> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird vorgegeben, welche der drei Funktionen durchgeführt werden soll:

- gesetztes Bit ermitteln,
- Bit setzen,
- Bit löschen.

<N-VAR1>	Funktion
0	Ermitteln des ersten gesetzten Bits, ab der in <N-VAR2> vorgegebenen Bitposition in <S-VAR>. Die ermittelte Adresse wird in <N-VAR3> zur Verfügung gestellt.
1	Das Bit dessen Adresse (relativ zu Bit 0) in <N-VAR2 vorgegeben ist wird gesetzt (erhält den Wert "1"). Bei dieser Funktion muß <N-VAR3> nicht codiert sein.
2	Das Bit, dessen Adresse (relativ zu Bit 0) in <N-VAR2 vorgegeben ist, wird gelöscht (erhält den Wert "0").

	Bei dieser Funktion muß <N-VAR3> nicht codiert sein.

<S-VAR> = String-Variable, in der Bits bearbeitet werden sollen.

<N-VAR2> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird eine Bit-Adresse vorgegeben. Die Behandlung des adressierten Bits ist abhängig von der in <N-VAR1> vorgegebenen Funktion.

<N-VAR1>	Funktion
0	Vorgabe der Adresse des Bits, ab dem der Vergleich gestartet werden soll.
1	Adresse des Bits, das gesetzt werden soll.
2	Adresse des zu löschenden Bits.

<N-VAR3> = Nur erforderlich, wenn <N-VAR1> = "1". Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird die Adresse des ersten gesetzten Bits an das Basic-Programm übergeben. Wird kein Bit gefunden, erhält <N-VAR3> den Wert -1!

Beispiel: Suchen des ersten gesetzten Bits im String A$. Die Adresse des ersten zu vergleichenden Bits wird in "A" vorgegeben. Die Adresse des ersten gesetzten Bits wird in "B" erwartet. Anschliessend wird das erste gesetzte Bit gelöscht

```
.
1000 LET  K=0            /* FUNKTION BIT SUCHEN
1005 CALL 20,K,A$;A,B
1010 IF B=-1 GOTO 9000   /* KEIN BIT IN A$ GES.
1015 LET  K=2            /* FUNKT. BIT LÖSCHEN
1020 LET  A=B            /* ADRESSE 1. GESETZ-
                            TES BIT
1025 CALL 20,K,A$,A      /* BIT LÖSCHEN
.
```

Anhang-CALL-Unterprogramme

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablen-Typ vorgegeben. - Vorgegebene Bit-Adresse negativ. - Fehlende Variable.

Anhang-CALL-Unterprogramme

11.6 CALL 21

Umwandlung des Inhaltes einer beliebigen numerischen Variablen in eine Dual-Zahl.

Syntax:

<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<S-VAR>,<N-VAR2>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 21 sein.

<N-VAR1> = Numerische Variable, Vektor- oder Matrix-Element. Der Inhalt dieser Variablen wird in eine zwei Byte große Dualzahl umgewandelt und ab der in <N-VAR2> vorgegebenen Byte-Adresse in <S-VAR> abgestellt. Der Inhalt von <N-VAR1> darf nicht größer als 65535 und nicht negativ sein.

<S-VAR> = String-Variable. In diesem String wird die Dual-Zahl abgestellt.

Achtung! Es wird nur die Dual-Zahl abgestellt. Ein Grenzzeichen wird nicht generiert.

<S-VAR> muß für mindestens zwei Byte dimensioniert sein.

<N-VAR2> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird die Byte-Adresse vorgegeben, ab der die Dual-Zahl in <S-VAR> abgestellt werden soll. Der Wert von <N-VAR2> muß im Bereich von 1 bis dimensionierte Länge von <S-VAR> - 1 liegen.

Anhang-CALL-Unterprogramme

Beispiel: Abstellen des Inhaltes von "A" als Dual-Zahl in "A$" ab Byte # 10.

```
.
1000 LET  Z=10          /*ADRESSE LADEN
1005 LET  F=65535       /*WERT LADEN
1010 CALL 21,F,A$,Z     /*WANDELN UND ABST.
.
```

Fehler:

Basic-Fehler	Bedeutung
# 38	Falscher Variablen-Typ vorgegeben. - Inhalt von <N-VAR1> ist negativ. - Inhalt von <N-VAR1> ist größer 65535. - Inhalt von <N-VAR2> ist kleiner als "0" oder größer als die dimensionierte Länge von <S-VAR>-1.

11.7 CALL 22

Übertragen einer 2 Byte großen Dualzahl aus einem String in eine beliebige numerische Variable und gleichzeitige Umwandlung in die entsprechende Datendarstellung (BCD oder Gleitkomma).

Syntax:

<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<S-VAR>,<N-VAR2>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 22 sein.

<N-VAR1> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird die Byte-Adresse vorgegeben, ab der die zu übertragende Dualzahl in <S-VAR> steht. Der Wert von <N-VAR1> muß im Bereich von 1 bis dimensionierte Länge von <S-VAR> - 1 liegen.

<S-VAR> = String-Variable. In diesem String wird ab der vorgegebenen Adresse der Inhalt von 2 Byte (max. 65535) in eine numerische Variable übernommen. Je nach Format der Zielvariablen wird die Umwandlung in BCD- oder Gleitkommadarstellung vorgenommen.

<N-VAR2> = Numerische Variable, Vektor- oder Matrix-Element. Diese Variable dient als Zielvariable. Sie nimmt den in die entsprechende Darstellung umgewandelten Wert der Dualzahl aus <S-VAR> auf.

Beispiel: Übernahme der in "A$" ab Byte # 4 stehenden Dualzahl in die Variable "F". Die Byte-Adresse wird in der Variablen "Z" vorgegeben.

```
.
1000 LET  Z=4          /* ADRESSE LADEN
1005 CALL 22,Z,A$,F    /* WERT ÜBERNEHMEN
.
```

Anhang-CALL-Unterprogramme

Fehler:	Basic-Fehler	Bedeutung
	# 15	Arithmetischer Überlauf. Ein Wert > 7999 soll in eine 1%-Variable übernommen werden.
	# 38	Falscher Variablen-Typ vorgegeben. - Inhalt von <N-VAR1> ist kleiner als "0" oder größer als die dimensionierte Länge von <S-VAR>. - Fehlende Variablen-Angabe.

Anhang-CALL-Unterprogramme

11.8 CALL 23

Suchen eines "Sub-Strings" in einem String.

Syntax:

<CALL>::=CALL <N-EXPR>,<S-VAR1>,<S-VAR2>,<N-VAR1>,<N-VAR2>

Funktion:

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 23 sein.

<S-VAR1> = String-Variable. In diesem String wird der in <S-VAR2> zu suchende Sub-String vorgegeben.

<S-VAR2> = String-Variable. In diesem String wird der in <S-VAR1> vorgegebene Sub-String gesucht.

<N-VAR1> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird die Byte-Adresse vorgegeben, ab der in <S-VAR2> mit dem Suchen begonnen werden soll. Der Wert von <N-VAR1> muß im Bereich von 1 bis dimensionierte Länge von <S-VAR2> liegen (Länge von <S-VAR1> -1).

<N-VAR2> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird nach der Durchführung der Anweisung die Anfangsadresse des Sub-Strings in <S-VAR2> übergeben. Ist der gesuchte Sub-String nicht in <S-VAR2> vorhanden, erhält <N-VAR2> den Wert -1.

Anhang-CALL-Unterprogramme

Beispiel: Suchen der Zeichenkombination "/*" in dem String A$ ab Byte # 1.

```
Inhalt A$: 1234505555/*003004005
.
1000 LET  B$="/*"       /* VERGLEICHSWERT
1005 LET  A=1           /* BYTE-ADRESSE
1010 CALL 23,B$,A$,A,B /* SUCHEN
1015 ...
.
Inhalt B: 11
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablen-Typ vorgegeben - Inhalt von <N-VAR1> ist kleiner als "1" oder größer als die dimensionierte Länge von <S-VAR2> - 1.

11.9 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> :: = CALL <N-EXPR>,<S-VAR>

Funktion:

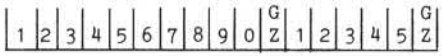
CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 24 sein.

<S-VAR> = Strings-Variable. Der Inhalt dieses Strings 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.

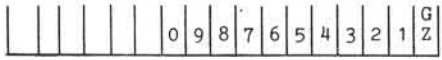
Beispiel: Umkehrung des Inhaltes von A$.

Inhalt von A$ vorher:




```
.
1000 CALL 24,A$
.
```

Inhalt von A$ hinterher:



Anhang-CALL-Unterprogramme

11.10 CALL 25

Datumsumwandlung in Industriedatum. Es bestehen die folgenden Möglichkeiten:

- Umwandlung des Systemdatums in Industriedatum.
- Umwandlung eines vom Anwenderprogramm vorgegebenen Datums in Industriedatum.

Syntax:

<CALL> :: = CALL <N-EXPR>,<S-VAR>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 25 sein.

<S-VAR> = String-Variable, in der das umzuwandelnde Datum vorgegeben wird, bzw. die ein Leerstring ist, wenn das Systemdatum umgewandelt werden soll.
<S-VAR> muß für mindestens 23 Byte dimensioniert sein!
Nach Ausführung der Anweisung steht in <S-VAR> von Byte 1 bis 5 das Industriedatum.

Wird in <S-VAR> ein Datum vorgegeben, muß es folgendes Format haben:

"MMM TT JJJJ"

Diese drei Angaben 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

Anhang-CALL-Unterprogramme

Das durch CALL aufbereitete Industriedatum hat das folgende Format: "JJTTT"

JJ = Jahr

TTT = Laufender Tag im Jahr

Bei der Umwandlung wird überprüft, ob das vorgegebene Datum zulässig ist. Schaltjahre werden bei der Prüfung berücksichtigt.

Beispiel: Umwandlung des Systemdatums in Industriedatum.

```
.
100  DIM A$(25)        /* DIMENSIONIERUNG
.
1000 LET A$= ""        /* <S-VAR> = LEERSTRING
1005 CALL 25,A$        /* DATUM AUFBEREITEN
.
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablentyp vorgegeben. - Datum in falschem Format vorgegeben. - Unmögliches Datum.

Anhang-CALL-Unterprogramme

11.11 CALL 51

Aufbereitung von Strings mit Oktal-Codes zur Unterstützung der Bildschirm-Programmierung.
Dieses Unterprogramm ist ausgelegt für Bildschirme mit 27 Zeilen zu je 74 Zeichen (Hazeltine). Bei Anschluß der NC Bildschirm-Arbeitsplätze werden die Funktionen des CALL 51 durch Funktionssymbole (siehe auch Pkt.: 7.2.2) ersetzt.

<CALL> :: = CALL <N-EXPR>,<S-VAR1>,<S-VAR2>,<S-VAR3>,<S-VAR4>,
<S-VAR5>,<S-VAR6>,<S-VAR7>,<S-VAR8>,<S-VAR9>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 51 sein.

<S-VAR1> = String-Variable, die für 74 Byte dimensioniert sein muß.
In den Bytes 1 bis 73 werden die Werte 1 bis 73 (= 201 bis 311 oktal) abgestellt. Byte 74 erhält den Wert 0 (200 oktal). Diese Werte dienen zur Vorgabe der Cursorpositionen (Zeichen/Zeile).

<S-VAR2> = String-Variable, die für 2 Byte dimensioniert sein muß. <S-VAR2> nimmt die Steuerzeichen für "Cursorpositionierung", (376/221 oktal) auf.

<S-VAR3> = String-Variable, die für 6 Byte dimensioniert sein muß. <S-VAR3> dient zur Aufnahme von 6 "RUB-OUT"-Zeichen. (377/377/377/377/377/377/oktal).

<S-VAR4> = String-Variable, die für 2 Byte dimensioniert sein muß. <S-VAR4> dient zur Aufnahme des Steuerzeichens "Bildschirm löschen" (376/234 oktal).

<S-VAR5> = String-Variable, die für 2 Byte dimensioniert sein muß. <S-VAR5> dient zur Aufnahme des Steuerzeichens "Löschen Vordergrundfelder" (= 376/235 oktal).

<S-VAR6> = String-Variable, die für 2 Byte dimensioniert sein muß. <S-VAR6> dient zur Aufnahme des Steuerzeichens "Start Hintergrund" (376/231 oktal).

<S-VAR7> = String-Variable, die für 2 Byte dimensioniert sein muß. <S-VAR7> dient zur Aufnahme des Steuerzeichens "Start Vordergrund" (376/237 oktal).

<S-VAR8> = String-Variable, die für 2 Byte dimensioniert sein muß. <S-VAR8> dient zur Aufnahme des Steuerzeichens "Zeile einfügen" (376/232 oktal).

<S-VAR9> = String-Variable, die für 2 Byte dimensioniert sein muß. <S-VAR9> dient zur Aufnahme des Steuerzeichens "Zeile löschen" (376/223 oktal).

Die Auswahl der entsprechenden Funktionen erfolgt nach der Durchführung des "CALL 51". Es wird der mit den Funktions-Codes geladene String mit der Anweisung "PRINT" ausgegeben. Die Funktion der einzelnen Steuerzeichen entspricht der Funktion der unter Pkt.: 7.2.2 beschriebenen BA-Funktionen. Eine Besonderheit stellen nur die in <S-VAR3> abgestellten "RUB-OUT"-Characters dar:

Diese Zeichen dienen dazu, für eine Zeit von 5 bis 6 Millisekunden, die von den Funktionen:

- Bildschirm löschen,
- löschen Vordergrundfelder,
- Zeile einfügen und
- Zeile löschen

benötigt wird, ein erneutes Ansprechen des Bildschirmes zu verhindern.

Beispiel: Durchführung des "CALL 51" und ausführen der Funktionen:

- Bildschirm löschen,
- Cursor-Positionierung auf Pos. 10 in Zeile 2.

```
.
1000 CALL 51,N$,N1$,N2$,N3$,N4$,N5$,N6$,N7$,N8$
1005 PRINT N3$;N2$;N1$,N$(10,10);N$(2,2);
.
```

Anhang-CALL-Unterprogramme

11.12 CALL 60

Konvertierung von numerischen (ASCII)-Zeichen in 4-Bit-Format. Die Ausgabe erfolgt ebenfalls in einem String. Durch dieses Unterprogramm besteht die Möglichkeit, speziell auf externen Datenträgern numerische Daten platzsparend abzustellen.

Syntax:

<CALL> :: = CALL <N-EXPR>,<S-VAR1>,<S-VAR2>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 60 sein.

<S-VAR1> = String-Variable. Der Inhalt dieses Strings wird in 4-Bit-Format umgewandelt und nach <S-VAR2> übertragen. Dieser String darf nur numerische ASCII-Zeichen und die Sonderzeichen "␣" "." "," "+" "-" enthalten.

<S-VAR2> = String-Variable. In diesem String wird der in 4-Bit-Format konvertierte Inhalt von <S-VAR1> abgestellt. Der konvertierte Wert entspricht den rechten 4 Bits der ASCII-Ziffer + 1.
Das heißt: NULL (ASCII 3.0) = 1
EINS (ASCII 3.1) = 2
.
NEUN (ASCII 3.9) = A
Die Konvertierung in den Ziel-String wird immer durch ein Grenzzeichen abgeschlossen. <S-VAR2> muß für mindestens die gleiche Anzahl Byte dimensioniert sein, wie Zeichen konvertiert werden.
Die Konvertierung wird beendet durch:
- Auftreten eines Grenzzeichens in <S-VAR1>
- Erreichen der dimensionierten Länge in <S-VAR1>
- Erreichen der dimensionierten Länge in <S-VAR2>

Anhang-CALL-Unterprogramme

Beispiel: Konvertieren des Inhaltes eines Strings in 4-Bit-Format.

```
Quellstring = A$
Zielstring  = B$
.
1000 LET A$ ="12345"
1005 CALL 60,A$,B$
.
```

Inhalt von B$ nach der Durchführung des "CALL 60":

|00L0|00LL|0L00|0L0L|0LL0|0000|

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablentyp vorgegeben. - Es ist ein nicht-numerisches Zeichen in <S-VAR1> aufgetreten.

Anhang-CALL-Unterprogramme

11.13 CALL 61

Rück-Konvertierung des Inhalts eines Strings, der durch CALL 60 aufbereitet wurde.

Syntax:

<CALL> :: = CALL <N-EXPR>,<S-VAR1>,<S-VAR2>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 61 sein.

<S-VAR1> = String-Variable. Der Inhalt dieses Strings muß durch "CALL 60" aufbereitet worden sein. Die in 4-Bit-Format abgestellten numerischen Zeichen werden in ASCII-Code umgewandelt und nach <S-VAR2> übertragen.

<S-VAR2> = String-Variable. In diesem String werden die in ASCII-Code umgewandelten numerischen Zeichen aus <S-VAR1> abgestellt. <S-VAR2> muß für mindestens die gleiche Anzahl 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 <S-VAR1>.
- Erreichen der dimensionierten Länge von <S-VAR2>.

Anhang-CALL-Unterprogramme

Beispiel: Konvertierung des durch "CALL 60" aufbereiteten Inhalts der Variablen A$ in die Variable B$.

Der Inhalt von A$ in binärer Darstellung:

00L0	00LL	0L00	0L0L	0LL0	000L	000L	0000

```
.
100 DIM A$(4),B$(10)    /* DIMENSIONIERUNG
.
200 CALL 61,A$,B$       /* KONVERTIERUNG
.
```

Inhalt des Strings B$ nach der Übertragung:

							G		
1	2	3	4	5	0	0	Z		

Fehler:

Basic-Fehler	Bedeutung
% 38	- Falscher Variablentyp vorgegeben. - <S-VAR2> enthält ein Zeichen das nicht im Bereich von: 0000 bis L0L0 (binär) liegt.

Anhang-CALL-Unterprogramme

11.14 CALL 62

Prüfen, ob der Inhalt eines Strings aus numerischen Zeichen (ASCII-Zeichen 0-9) besteht.

Syntax:

<CALL> :: = CALL <N-EXPR>,<S-VAR>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 62 sein.

<S-VAR> = String-Variable deren Inhalt auf numerische Zeichen überprüft wird.

Beispiel: Überprüfung des Inhaltes von A$, ob nur numerische Zeichen vorhanden sind.

```
.
1000 CALL 62,A$
.
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablentyp vorgegeben. - In <S-VAR> wurde ein nichtnumeriches Zeichen gefunden.

11.15 CALL 63

Prüfen ob der Inhalt eines Strings ein arithmetischer Wert ist.
Zusätzliche zu den ASCII-Zeichen 0 bis 9 ist ein führendes Vorzeichen, - oder +, und an beliebiger Stelle ein Dezimalkomma (Dezimalpunkt) zugelassen.

Syntax:

<CALL> :: = CALL <N-EXPR>,<S-VAR>

Anhang-CALL-Unterprogramme

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 63 sein.

<S-VAR> = String-Variable, deren Inhalt daraufhin überprüft wird, ob er ein arithmetischer Wert ist.

Beispiel: Prüfen des Inhaltes von A$, ob er ein gültiger arithmetischer Wert ist.

```
.
1000 CALL 63,A$
.
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablentyp vorgegeben. - In <S-VAR> wurde ein unzulässiges Zeichen gefunden - Außer einem Vorzeichen kein weiteres Zeichen in <S-VAR>.

Anmerkung: Ein Leerstring (LEN = 0) führt zu keiner Fehlermeldung!

Anhang-CALL-Unterprogramme

11.16 CALL 64

Prüfen eines Strings, ob er ein gültiges Datum enthält. Für das Datum ist folgendes Format vorgeschrieben:

MM.TT.JJ

Ist das geprüfte Datum zulässig, wird es umgestellt und in der Form

JJMMTT

im String abgestellt.

Syntax:

<CALL> :: = CALL <N-EXPR>,<S-VAR>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 64 sein.

<S-VAR> = String-Variable, die für mindestens 8 Byte dimensioniert sein muß.
Der Inhalt dieses Strings wird daraufhin überprüft, ob er ein gültiges Datum enthält. Das Datum muß in folgendem Format in <S-VAR> vorgegeben sein:
MM.TT.JJ
Ist das Datum gültig, wird es umgestellt und hat anschließend die Form:
JJMMTT

Beispiel: Überprüfung und Umstellung des in A$ abgestellten Datums.

```
.
100  DIM A$(8)          /* DIMENSIONIERUNG
.
1000 CALL 64,A$         /* DATUM PRÜFEN
.
```

Inhalt A$: vor CALL 64 nach CALL 64

02.01.78 780201

Anhang-CALL-Unterprogramme

Fehler:	Basic-Fehler	Bedeutung
	# 38	- Falscher Variablentyp vorgegeben. - <S-VAR> für weniger als 8 Byte dimensioniert. - Auf einer der Stellen 1,2,4,5 7 oder 8 wurde ein nicht-numerisches Zeichen gefunden. - Das 3. oder 6. Zeichen ist kein Punkt "." - Monat < 1 oder > 12. - Tag < 1 oder > 30 bzw. 31. Der 29. Februar ist nur in Schaltjahren zugelassen.

Anhang-CALL-Unterprogramme

11.17 CALL 65

Sortieren einer Speicher-Datei in aufsteigender Ordnung (ASCII-Code).
"Speicher-Datei" = String in beliebiger Länge, in der die zu sortierenden Sortierbegriffe im ASCII-Code abgestellt sind.

Syntax:

<CALL> ::=CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,<S-VAR1>,<S-VAR2>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 65 sein.

<N-VAR1> = Numerische Variable, Vektor- oder Matrix- Element. Diese Variable dient zur Übergabe einer Statusmeldung von dem Unterprogramm an das Anwenderprogramm. Der Inhalt von <N-VAR1> muß beim Aufruf des Unterprogrammes = 0 sein.

<N-VAR2> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen muß die Anzahl der zu sortierenden Sortierbegriffe vorgegeben werden. Der Inhalt von <N-VAR2> muß >= 1 sein!

<N-VAR3> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen muß die Sortierbegrifflänge vorgegeben sein. Der Inhalt von <N-VAR3> muß >= 1 sein!

<S-VAR1> = String-Variable. Der Inhalt des Strings wird entsprechend der in <N-VAR2> und <N-VAR3> vorgegebenen Parametern in aufsteigender Folge sortiert.
CALL 65 stellt nach Beendigung des Sortiervorganges kein Grenzzeichen ab!!
Die Mindestgröße von <S-VAR1> ergibt

sich aus <N-VAR2> * <N-VAR3>.

<S-VAR2> = String-Variable. Dieser String dient dem Unterprogramm als Arbeitsfeld. Die Mindestgröße in Byte beträgt:
<N-VAR3> + 8.

Beispiel: Sortieren von den in A$ abgestellten Sortierbegriffen. Die Länge eines Sortierbegriffes ist 4 Byte. Die Anzahl der zu sortierenden Begiffe ist 50.

```
.
100  DIM A$(200)          /* DIM SPEICHERDATEI
105  DIM W$(12)           /* DIM ARBEITSFELD
.
1000 LET S=0              /* STATUS-VAR AUF 0
1005 LET N=50             /* ANZAHL SORTIERBE-
                                GRIFFE
1010 LET L=4              /* LÄNGE
1015 CALL 65,S,N,L,A$,W$ /* SORTIEREN
1020 IF S GOTO ...        /* STATUS AUSWERTEN
.
```

Fehler:

Basic-Fehler ♯	Bedeutung
38	- Falscher Variablentyp vorgegeben.

Status in <N-VAR1>

Status ♯	Bedeutung
0	- Sortierung wurde ohne Fehler durchgeführt.
1	- Parameterfehler: Fehlende(r) Parameter. Falscher Variablentyp. <N-VAR1> bei Aufruf <> 0. <N-VAR2> und/oder <N-VAR3> negativ oder > 33767. Inhalt von <N-VAR2> oder <N-VAR3> kleiner 1.
3	- <S-VAR1> zu klein dimensioniert.
4	- <S-VAR2> zu klein dimensioniert.

	Anhang-CALL-Unterprogramme

5	- Ein Variablenname wurde mehrfach verwendet.

Anhang-CALL-Unterprogramme

11.18 CALL 70

"CALL 70" repräsentiert ein physikalisches IOCS und bildet die Schnittstelle zwischen dem Basic-Interpreter "RUN" und dem Kanalprogramm $MTX.

Syntax: Syntaktisch werden 4 Formate unterschieden:

Format 1:

<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>

Format 2:

<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,<N-VAR4>,<N-VAR5>

Format 3:

<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,<N-VAR4>,<N-VAR5>,<S-VAR1>

Format 4:

<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,<S-VAR2>,<S-VAR3>

Funktion:

CALL = Anweisung.

<N-EXPR> = Nummer des aufzurufenden Unterprogrammes, muß in diesem Fall = 70 sein.

<N-VAR1> = Numerische Variable, in der die Nummer der anzusprechenden Bandstation vorgegeben werden muß. <N-VAR1> muß immer = 0 sein!

<N-VAR2> = Numerische Variable, in der die Nummer der durchzuführenden Funktion vorgegeben werden muß. <N-VAR2> muß im Bereich von 1 bis 14 liegen.

<N-VAR3> = Numerische Variable, in der nach

Anhang-CALL-Unterprogramme

Durchführung der Funktion ein Statuscode übergeben wird. <N-VAR3> muß vor Ausführung der Funktion = 0 sein.

<N-VAR4> = Anzahl der zu verarbeitenden Dateien, Blöcke oder Byte abhängig von der Funktion.

<N-VAR5> = Aktuelle Anzahl der verarbeiteten Dateien, Blöcke oder Byte abhängig von der durchgeführten Funktion.

<S-VAR1> = String-Variable zur Aufnahme des gelesenen oder zu schreibenden Blocks.

<S-VAR2> = String-Variable zur Aufnahme der Eingabe-Codetabelle. Diese Tabelle muß vom Anwenderprogramm geladen werden und 256 Byte lang sein!

<S-VAR3> = String-Variable zur Aufnahme der Ausgabe-Codetabelle. Diese Tabelle muß vom Anwenderprogramm geladen werden und muß 256 Byte lang sein!

Das zu wählende Format ist von der durchzuführenden Funktion abhängig.

Format 1: CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>

<N-VAR2>	Funktion
1	Band auf BOT (begin of tape) zurückspulen.
2	Bandgerät "OFF LINE" schalten.
6	Bandmarke schreiben.
7	Gap schreiben (2,5 Zoll Bandlücke).
10	Bandmarke schreiben und Band auf BOT zurückspulen.
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.

Format 2: CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,<N-VAR4>,<N-VAR5>

<N-VAR2>	Funktion
3	Band um die in <N-VAR4> vorgegebene Anzahl Blöcke vorsetzen.
4	Band um die in <N-VAR4> vorgegebene Anzahl Blöcke zurücksetzen.
8	Band um die in <N-VAR4> vorgegebene Anzahl Bandmarken vorsetzen.
9	Band um die in <N-VAR4> vorgegebene Anzahl Bandmarken zurücksetzen.

Anmerkung: Die Anzahl Blöcke/Bandmarken wird in <N-VAR4> vorgegeben. <N-VAR5> 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 <N-VAR3> ein Statuscode übergeben.

Format 3: CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,<N-VAR4>,<N-VAR5>,<S-VAR1>

<N-VAR2>	Funktion
0	Block lesen. Die Übertragung wird beendet bei Erreichen der dimensionierten Länge von <S-VAR1> oder durch Erreichen der in <N-VAR4> vorgegebenen Anzahl Zeichen.
5	Block schreiben. Die Übertragung wird beendet bei Erreichen der dimensionierten Länge von <S-VAR1> oder durch Erreichen der in <N-VAR4> vorgegebenen Anzahl Zeichen.

Anmerkung: Nachdem die Funktion ausgeführt ist, wird in <N-VAR5> die tatsächliche Blocklänge beim

Anhang-CALL-Unterprogramme

Lesen bzw. die Anzahl tatsächlich geschriebener Zeichen übergeben. Dieser Wert muß nicht mit dem in <N-VAR4> vorgegebenen Wert übereinstimmen.

Format 4: CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,<S-VAR2>,<S-VAR3>

<N-VAR2>	Funktion
11	Magnetbandstation zur Verarbeitung mit Codewandlung eröffnen. Die Angabe <S-VAR3> muß nur bei der Ausgabe auf Magnetband angegeben sein.
12	Wie <N-VAR2> = 11, jedoch mit Zurückspulen des Bandes auf BOT.

Anmerkung: Die Umwandlungstabellen müssen vom Basic-Programm in <S-VAR2> und <S-VAR3> zur Verfügung gestellt, und dürfen nach der Eröffnung des Magnetbandes nicht mehr verändert werden. Folgende Standard-Codetabellen stehen in relativen Dateien auf Magnetplatten 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

Anhang-CALL-Unterprogramme

Statuscodes

Nach der Ausführung einer Magnetband-Operation ist unbedingt der Status in <N-VAR3> zu überprüfen.
Folgende Werte können in <N-VAR3> durch das Unterprogramm übergeben werden:

<N-VAR3>	Bedeutung
0	Operation 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

Beispiel: Ausgabe einer relativen Datei auf Magnetband.
Folgende Variablen werden benutzt:

U = Gerätenummer Magnetband
F = Funktion
S = Status
C = Block-/Satzlänge
A = Aktuelle (geschriebene) Länge
R$ = Datensatz
D$ = Name Plattendatei

Die Ausgabe erfolgt ohne Codeumwandlung:

```
10  IF ERR 0 GOSUB 330
20  DIM D$(20)
30  PRINT 'CS'; TAB (0,24);"RELATIVE DATEI AUF MB KONVERTIEREN"
40  INPUT 'CR',"DATEINAME:", TAB (30)D$      /*RELATIVE DATEI
50  INPUT 'CR',"SATZLÄNGE:", TAB (30)C
60  IF C>2048 GOTO 50                        /*MAX. SL = 2048
70  DIM R$(C)                                /*DIM MIT SATZLÄNGE
80  OPEN #2,D$                               /*OPEN RELATIVE DATEI
90  LET F=12                                 /*FUNKTION LADEN
100 CALL 70,U,F,S                            /*OPEN UND REWIND
110 IF S GOTO 220                            /*FEHLER BEI OPEN
120 LET F=6                                  /*FUNKTION LADEN
```

Anhang-CALL-Unterprogramme

```
125 CALL 70,U,F,S                                   /*BANDMARKE SCHREIBEN
130 IF S GOTO 200                                   /*FEHLER
140 LET F=5                                         /*FUNKTION LADEN
150 READ #2;R$;                                     /*PLATTENSATZ LESEN
160 CALL 70,U,F,S,C,A,R$                            /*BLOCK AUF BAND SCHR.
170 IF S GOTO 200                                   /*FEHLER BEI SCHREIBEN
180 LET R$=" ",R$                                   /*STRING LÖSCHEN
190 GOTO 150                                        /*NÄCHSTEN PLATTENS.
200 LET S1=S                                        /*FEHLERSCHL. SICHERN
210 GOSUB 250                                       /*BAND SCHLIESSEN
220 PRINT 'CR';"BANDFEHLER = ";S
230 CLOSE #2                                        /*CLOSE RELATIVE DATEI
240 CHAIN ""                                        /*CHAIN NACH SCOPE
250 LET F=10                                        /*FUNKTION LADEN
260 LET S=0                                         /*STATUS AUF 0
270 CALL 70,U,F,S                                   /*BANDMARKE SCHREIBEN
280 LET F=13                                        /*FUNKTION LADEN
290 LET S=0                                         /*STATUS AUF 0
300 CALL 70,U,F,S                                   /*BANDGERÄT SCHLIESSEN
310 LET S=S1                                        /*STATUS ZURÜCK
320 RETURN
330 IF SPC 8>97 RETURN -1                           /*ESC ODER FALSCHEING.
340 IF SPC 10<110 GOTO 360                          /*NOCH KEIN BAND-OPEN
350 GOSUB 250
360 IF SPC 8=51 GOTO 400                            /*DATEIENDE?
370 PRINT 'CR';"BASIC-FEHLER # "; SPC 8;"ZEILE = "; SPC 10
380 IF SPC 10<90 CHAIN                              /*NOCH KEIN OPEN
390 GOTO 230
400 PRINT 'CR';"DATEIENDE"
410 GOTO 230
```

Anhang-CALL-Unterprogramme

11.19 CALL 72

Übertragen des Inhalts von bis zu 11 Variablen in einen String. Die Übertragung erfolgt 1:1 ohne Rücksicht auf Typ und Format der Quellvariablen.

Syntax:

<CALL> ::= CALL <N-EXPR> [, { <S-VAR> / <N-VAR> }] (1 bis 12)

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 72 sein.

<S-VAR>
<N-VAR> = Maximal 12 beliebige Variablen.
Die erste codierte Variable sollte eine String-Variable sein. Sie dient zur Aufnahme der (bis zu 11) im folgenden angegebenen Variablen.
Die Übertragung wird nur durch das Erreichen des dimensionierten Endes in der Zielvariablen oder durch Erreichen des Endes der letzten Quell-Variablen beendet. Grenzzeichen in Quellstrings beenden die Übertragung nicht. Ist als erste Variable (Ziel) eine numerische Variable angegeben, erfolgt keine Fehlermeldung!

Anhang-CALL-Unterprogramme

Beispiel: Übertragung des Inhaltes der Variablen:
A,B,C,A$,B$,C$
in den String M$.

```
.
100  DIM A$(10),B$(10),C$(10)
105  DIM 2%,A,B,C
110  DIM M$(42)                    /*ZIELSTRING
.
1000 CALL 72,M$,A,B,C,A$,B$,C$ /*ÜBERTRAGUNG
```

11.20 CALL 73

Übertragen des Inhalts eines Strings in bis zu 11 Variablen. Die Übertragung erfolgt 1:1 ohne Rücksicht auf Typ und Format der Zielvariablen.

Syntax:

<CALL> ::= CALL <N-EXPR> [, { <S-VAR> / <N-VAR> }] 12 / 1

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 73 sein.

<S-VAR>
<N-VAR> = Zwei bis maximal 12 Variablen.
Als erste Variable (Quellvariable) muß ein String angegeben sein. Der Inhalt dieses Strings wird 1:1 in bis zu 11 Zielvariablen übertragen.
Die Übertragung wird beendet durch:
- Erreichen des dimensionierten Endes in dem Quellstrings.
- Erreichen des dimensionierten Endes in der letzten Zielvariablen.

Beispiel: Der Inhalt des Strings M$ wird 1:1 in die Variablen A,B,C,A$,B$,C$ übertragen.

```
.
100 DIM A$(10),B$(10),C$(10)   /*ZIELVARIABLE
105 DIM 2%,A,B,C               /*ZIELVARIABLE
110 DIM M$(42)                 /*QUELLVARIABLE
  .
1000 CALL 73,M$,A,B,C,A$,B$,C$ /*ÜBERTRAGUNG
  .
```

Anhang-CALL-Unterprogramme

11.21 CALL 80

Schnittstelle zwischen "RUN" und "$CAS"

CALL 80 repräsentiert ein physikalisches IOCS und bildet die Schnittstelle zwischen dem BASIC-Interpreter "RUN" und dem Driver "$CAS".
Auf einer Cassette können lediglich sequentielle Dateien abgestellt werden. Die Verwaltung der Dateien ist vom Anwenderprogramm zu steuern.

Syntax:

Syntaktisch wird zwischen drei Formaten unterschieden:

Format 1: CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>

Funktion:

CALL = Anweisung.

<N-EXPR> = Nummer des aufzurufenden Unterprogrammes. In diesem Fall = 80.

<N-VAR1> = Numerische Variable (1-4%), in der die Nummer der anzusprechenden MBC-Einheit vorgegeben ist. Muß immer "0" sein.

<N-VAR2> = Numerische Variable (1-4%), in der die Nummer der durchzuführenden Funktion vorzugeben ist.

<N-VAR3> = Numerische Variable, (1-4%) in der, nachdem die Funktion durchgeführt ist, ein Statuscode übergeben wird.

Anhang-CALL-Unterprogramme

Format 2: CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,
<N-VAR4>,<N-VAR5>,<S-VAR>

Funktion:

CALL = Anweisung.

<N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3> (siehe Format 1)

<N-VAR4> = Numerische Variable (1-4%) in der, abhängig von der Funktion, die Anzahl zu lesender / zu schreibender Bytes vorzugeben ist (maximal = 512 Byte).

<N-VAR5> = Numerische Variable (1-4%), in der, nachdem die Funktion durchgeführt ist, die aktuelle Anzahl der gelesenen oder geschriebenen Bytes Übergeben wird.

<S-VAR> = String-Variable zur Aufnahme der gelesenen oder geschriebenen Datenblöcke.

Anhang-CALL-Unterprogramme

Format 3: CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3>,
<S-VAR1>,<S-VAR2>

Funktion:

CALL = Anweisung.

<N-EXPR>,<N-VAR1>,<N-VAR2>,<N-VAR3> (wie Format 1 u. 2)

<S-VAR1> = String-Variable (256 Byte), zur Aufnahme der Eingabe-Codetabelle. Sie muß, bevor die MBC eröffnet wird, vom Anwenderprogramm geladen werden.

<S-VAR2> = String-Variable (265 Byte), zur Aufnahme der Ausgabe-Codetabelle. Sie muß, bevor die MBC eröffnet wird, vom Anwenderprogramm geladen sein.

Das zu wählende Format ist von der durchzuführenden Funktion abhängig.

Anhang-CALL-Unterprogramme

Format 1

F	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 25 Sekunden zum Einlegen der Cassette und Schließen des Deckels durch den Bediener. Danach wird das Band auf die BOT-Marke (Beginn of tape) zurückgespult. Wird innerhalb der Wartezeit (25 sec) keine Cassette eingelegt und der Deckel nicht geschloßen, wird der Status "Time out" gemeldet und der Inhalt von <N-VAR3> ist 5.
13	Die Verarbeitung der MBC schließen, die Cassette auf 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 hinter dem aktuellen Datenblock positioniert. Folgt auf den aktuellen Block kein weiterer, wird nach 25 Sekunden "Time out" gemeldet und der Inhalt von <N-VAR3> auf 5 gesetzt.
4	Cassetten-Band um einen Block zurücksetzen, d.h., das Band wird auf das nächste GAP vor dem aktuellen Datenblock positioniert. Ist dort kein Block vorhanden (aktueller Block = erster Block), wird nach 25 Sekunden "Time out" gemeldet und der Inhalt von <N-VAR3> auf 5 gesetzt.
1	Cassetten-Band auf BOT (Beginn of tape) zurückspulen.
2	Deckel der MBC-Einheit öffnen und Cassette auswerfen.

Anhang-CALL-Unterprogramme

Format 2

F	Funktion
0	Datenblock lesen und in <S-VAR> übertragen. Wenn die Cassette mit Format 3 eröffnet ist, wird eine entsprechende Codeumwandlung durchgeführt. Die Übertragung wird beendet, sobald die von <S-VAR> dimensionierte Länge oder die in <N-VAR4> vorgegebene Anzahl Byte erreicht ist. Nachdem die Funktion durchgeführt ist, wird in <N-VAR5> die Anzahl der tatsächlich übertragenen Byte übergeben. Der in <N-VAR4> angegebene Wert darf die maximale Blocklänge von 512 Byte nicht überschreiten. Nachdem die Funktion durchgeführt ist, ist das Band auf dem GAP hinter dem zuletzt gelesenen Block positioniert. Treten Fehler beim Lesen auf, wird der Vorgang bis zu acht mal wiederholt. Sind die Leseversuche ohne Ergebnis, wird der Status "bad tape" gemeldet und in <N-VAR3> der Wert 4 abgestellt.
5	Block schreiben, d.h., der Inhalt des mit <S-VAR> bezeichneten Strings wird ab der aktuellen Position auf das Band geschrieben. Ist die MBC mit Format 3 eröffnet worden, wird eine Codeumwandlung über eine Tabelle durchgeführt. Die Übertragung wird beendet sobald die dimensionierte Länge von <S-VAR>, oder die in <N-VAR4> vorgegebene Anzahl Byte erreicht ist. Nachdem die Funktion durchgeführt ist, wird in <N-VAR5> die Anzahl der tatsächlich geschriebenen Byte übergeben und das Band auf dem GAP hinter dem zuletzt geschriebenen Block positioniert. Der in <N-VAR4> vorgegebene Wert darf die maximale Blocklänge von 512 Byte nicht überschreiten. Bei Schreibfehlern wird der Vorgang bis zu acht mal wiederholt. Bleibt die Funktion trotzdem ohne Ergebnis, wird der Status "bad tape" gemeldet und <N-VAR3> der Wert 4 eingetragen.

Anhang-CALL-Unterprogramme

Format 3

F	Funktion
11	Die MBC-Einheit wird zur Verarbeitung mit Codeumwandlung eröffnet.

Anmerkung:

Die Codetabellen müssen vom entsprechenden Basic-Programm in <S-VAR1> und <S-VAR2> zur Verfügung gestellt werden. Nach dem Eröffnen darf der Variableninhalt nicht mehr verändert werden.

Die folgenden Standard-Codetabellen stehen auf der Magnetplatte 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.

Anhang-CALL-Unterprogramme

Statuscodes

Jeweils, nachdem eine MBC-Funktion durchgeführt ist, ist der in <N-VAR3> abgestellte Status abzufragen.

Status	Bedeutung
0	Operation war erfolgreich (alle Funktionen).
1	Die MBC-Einheit ist nicht verfügbar (von anderem Teilnehmer bereits eröffnet).
2	Die MBC-Einheit ist nicht bereit (alle Funkt.)
3	Die Cassette ist schreibgeschützt (Funkt. 5).
4	Schreib-/Lesefehler (Funktionen 0 und 5).
5	Time out (Funktionen 0,1,2,3,4,11,13).
6	End of tape (Funktionen 0,3,4,5).
7	Unzulässige Operation (alle Funktionen).

11.22 CALL 90

Unterstützung der Bearbeitung von String-Variablen.

Syntaktisch werden drei Formate unterschieden.

Format 1: CALL <N-EXPR>,<N-VAR1>,<S-VAR1>,
<N-VAR2> ,[<N-VAR3>]$_{1}^{9}$

Funktion:

CALL = Anweisung.

<N-EXPR> = Die Nummer des aufzurufenden Unterprogrammes. In diesem Fall = 90.

<N-VAR1> = Numerische Variable (1-4%), die den Funktionscode enthält.

Funktionscode:

0= Transport von 1-9 numerischen Variablen (Quelle) in eine String-Variable (Ziel).
1= Transport des Inhaltes einer String-Variable (Quelle) in eine bis neun numerische Variablen (Ziel).

<S-VAR1> = String-Variable, die den Ziel- oder Quell-String enthält. Ob der String das Ziel oder die Quelle ist, ist von dem in <N-VAR1> vorgegebenen Funktionscode abhängig.

<N-VAR2> = Numerische Variable (1-4%). Sie bezeichnet das erste zu bearbeitende Byte in <S-VAR1> und muß einen ungeraden Wert enthalten. Dieser Wert darf nicht größer als die dimensionierte Länge von <S-VAR1> sein.

<N-VAR3> = Eine bis neun numerische Variablen, die entweder als Quell- oder Ziel-Variablen definiert sind. Ob Quell- oder Ziel-Variable, ist von dem in <N-VAR1> vorgegebenen Funktionscode abhängig.

Anmerkung:

Die Übertragung wird beendet, wenn die dimensionierte Länge von <S-VAR1> erreicht, oder der Inhalt der zuletzt

angegebenen numerischen Variable übertragen ist.

Format 2: CALL <N-EXPR>,<N-VAR1>,<S-VAR1>,
<N-VAR2>,<S-VAR2>

<N-EXPR> = Nummer des Unterprogrammes = 90.

<N-VAR1> = Numerische Variable (1-4%), die den Funktionscode enthält.
Er muß in diesem Fall = 2 sein.

Dadurch wird der Inhalt eines Strings, ohne Berücksichtigung von Grenzzeichen, in einen Ziel-String übertragen. Die Übertragung wird beendet wenn die dimensionierte Länge des Quell- oder Ziel-Strings erreicht ist.

<S-VAR1> = String-Variable, die als Ziel-String dient.

<N-VAR2> = Numerische Variable. Sie enthält die relative Byte-Adresse im Ziel-String, ab der die zu übertragenden Zeichen abgestellt werden.
Der angegebene Wert darf nicht größer als die dimensionierte Länge von <S-VAR1> sein.

<S-VAR2> = String-Variable, die als Quell-String dient.

Anhang-CALL-Unterprogramme

Format 3: CALL <N-EXPR>,<N-VAR1>,<S-VAR1>,
<N-VAR2>, <N-VAR3>

<N-EXPR> = Nummer des Unterprogrammes = 90.

<N-VAR1> = Numerische Variable (1-4%), die den Funktionscode enhält. Er muß hier = 3 sein.

Dadurch wird der Ziel-String von VON einer zu definierenden Byte-Adresse, BIS zu einer zu definierenden Byte-Adresse, bzw., bis zum dimensionierten Ende, mit Grenzzeichen gefüllt.

<S-VAR1> = String-Variable, die als Ziel-String dient.

<N-VAR2> = Numerische Variable (1-4%). Sie enthält die Adresse des ersten Bytes in <S-VAR1> (Zielstring) das mit Grenzzeichen gefüllt wird. Der Wert darf nicht größer sein, als die dimensionierte Länge von <S-VAR1>.

<N-VAR3> = Numerische Variable (1-4%). Sie enthält die Adresse des letzten Bytes in <S-VAR1> das mit Grenzzeichen gefüllt wird.
Die Adresse muß größer, gleich dem in <N-VAR2> angegebenen Wert sein.

Ist <N-VAR3> nicht angegeben, wird der String bis zum dimensionierten Ende mit Grenzzeichen gefüllt.
Ebenso, wenn <N-VAR3> größer als die dimensionierte Länge von <S-VAR1> ist.

Fehler 38	Bedeutung
	- Falscher Variablentyp wurde vorgegeben.
(Format 1)	- Ist <N-VAR1> = 0 oder 1, enthält <N-VAR2> einen geradzahligen Wert oder ist größer als die dimensionierte Länge von <S-VAR1>.
(Format 2 und 3)	- Ist <N-VAR1> = 2, ist der Wert von <N-VAR2> größer als die dimensionierte Länge von <S-VAR2>.
	- Wenn <N-VAR1> = 3, darf der Wert von <N-VAR2> nicht größer als die dimensio-

nierte Länge von <S-VAR2> sein, oder
der Wert von <N-VAR3> ist größer als der
Wert von <N-VAR2>.

Anhang-CALL-Unterprogramme

11.23 CALL 91

Inversion von Sub-Strings in eine beliebige String-Variable.

Syntax: CALL <N-EXPR>,<N-VAR1>,<S-VAR1>,<N-VAR2>,
<N-VAR3>,<N-VAR4>

Funktion:

CALL = Anweisung.

<N-EXPR> = Nummer des aufzurufenden Unterprogrammes. Sie muß in diesem Fall = 91 sein.

<N-VAR1> = Numerische Variable (1-4%), die den Funktionscode enthält. Dieser muß hier = 0 sein.

<S-VAR1> = String-Variable deren Inhalt den Substring darstellt, oder in die der Inhalt von Sub-Strings invertiert wird.

<N-VAR2> = Numerische Variable (1-4%). Sie enthält die relative Adresse des betreffenden Bytes in <S-VAR1> ab der invertiert werden soll.

<N-VAR3> = Numerische Variable (1-4%), die die Anzahl Bytes enthält deren Inhalt invertiert werden soll.

<N-VAR4> = Numerische Variable (1-4%), die den Skip-Wert enthält.

Der Inhalt von <N-VAR4>, auf die jeweils aktuelle Anfangsadresse eines Sub-Strings addiert, ergibt die Anfangsadresse des nächsten Sub-Strings in <S-VAR1>.

Der Wert in <S-VAR4> muß größer, gleich dem Wert in <N-VAR3> sein.

Fehler 38	Bedeutung
	- Falscher Variablentyp wurde vorgegeben.
	- Der Wert von <N-VAR4> ist kleiner als der Wert in <N-VAR3>.

Anhang-CALL-Unterprogramme

11.24 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
  - Privilegstufe
  - Kontogruppennummer
  - Kontobenutzernummer
- Dateiart
  - Dateiart
  - Dateischutz
- Derzeitige Dateigröße in Blöcken (Sektoren)
- Dateistatus
- Dateikosten
- Kumulierter Betrag, der anderen Teilnehmern für die Benutzung der Datei bisher berechnet wurde.
- Zeitpunkt der Dateierstellung (abgelaufene Stunden, bezogen auf den 1.1.1973).
- Zeitpunkt der letzten Dateieröffnung (abgelaufene Stunden seit dem 1.1.1973).
- Plattenadresse des Dateikennsatzes.

Der Zugriff auf die Datei "INDEX" erfolgt durch die Vorgabe der relativen Satznummer innerhalb der Datei "INDEX" beginnend mit 0.

Syntax:
<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<N-VAR2>,<S-VAR>,
<N-VAR3>,<N-VAR4>,<N-VAR5>,<N-VAR6>,
<N-VAR7>,<N-VAR8>,<N-VAR9>,<N-VAR10>,
<N-VAR11>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterpro-

grammes.
<N-EXPR> muß in diesem Fall = 97 sein.

<N-VAR1> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen ist die logische Gerätenummer der Platte vorzugeben, von der die Dateiinformation gelesen werden soll.

<N-VAR2> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen ist die relative Satznummer des zu lesenden Satzes in der Datei "INDEX" vorzugeben.

Alle folgenden Variablen werden durch CALL 97 aufbereitet und beinhalten nach der Durchführung des CALL die Dateiinformationen.

<S-VAR> = String-Variable, die für mindestens 15 Byte dimensioniert sein muß. In diesem String wird der Dateiname übergeben.

<N-VAR3> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen werden die Angaben:
- Privilegstufe,
- Kontogruppennummer und
- Kontobenutzernummer
an das Anwenderprogramm übergeben. Wie der Inhalt von <N-VAR3> aufgeschlüsselt werden kann, ist im Beispiel unter den Zeilennummern 590 bis 610 zu sehen.

<N-VAR4> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen werden die Angaben:
- Dateiart und
- Dateischutz
an das Anwenderprogramm übergeben. Wie der Inhalt von <N-VAR4> aufgeschlüsselt werden kann, ist im Beispiel unter den Zeilennummern 500 bis 570 zu sehen.

<N-VAR5> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird die Dateigröße in Blöcken (Sektoren) an das Anwenderprogramm übergeben.

<N-VAR6> = Numerische Variable, Vektor- oder

Matrix-Element. In dieser Variablen wird der Dateistatus an das Anwenderprogramm übergeben. Ein Wert <> 0 bezeichnet eine formatierte Datei.

<N-VAR7> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird der Betrag, mit dem andere Teilnehmer für die Benutzung dieser Datei belastet werden, an das Anwenderprogramm übergeben.

<N-VAR8> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird der kumulierte Betrag, mit dem andere Teilnehmer für die bisherige Benutzung dieser Datei belastet werden, an das Anwenderprogramm übergeben.

<N-VAR9> = Numerische Variablen, Vektor- oder Matrix-Element. In dieser Variablen wird der Zeitpunkt der Dateierstellung (bezogen auf den 1.1.1973) übergeben.

<N-VAR10> = Numerische Variable , Vektor- oder Matrix-Element. In dieser Variablen wird der Zeitpunkt der letzten Dateieröffnung (bezogen auf den 1.1.73) übergeben.

<N-VAR11> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird die Plattenadresse des Dateikennsatzes dieser Datei übergeben.

Ist der über <N-VAR2> adressierte Satz in der Datei "INDEX" ein Leersatz, wird so lange sequentiell weitergelesen, bis ein Satz mit Dateiinformationen oder das Dateiende erreicht ist.
Ist das Dateiende erreicht, wird in <N-VAR2> der Wert -1 übergeben.
Nach der Durchführung des CALL enthält <N-VAR2> die relative Satznummer des zuletzt gelesenen Satzes +1.

Fehler:	Basic-Fehler	Bedeutung
	# 38	- <S-VAR> wurde für weniger als 15 Byte dimensioniert. - Angesprochene LU ist nicht aktiv.

- Numerische Variable wurde kleiner als 2% dimensioniert.
- Falscher Variablentyp vorgegeben.

Beispiel: Auflistung der Dateiinformationen der Magnetplatte mit der logischen Gerätenummer 0.

Erläuterung der im Beispiel ausgedruckten Werte:

FILENAME = Dateiname

PROT = Dateischutz

COST = Dateikosten

SIZE = Dateigröße in Blöcken (Sektoren)

ACCOUNT = Kontogruppennummer, Kontobenutzernummer

AGE = Vergangene Zeit in Stunden seit der Dateierstellung

HSLA = Vergangene Zeit in Stunden seit der letzten Dateieröffnung

TYPE = Dateiart, max. 3-stellig (oktal):
1. Stelle = Kennzeichen für lauffähigen Processor
2. und 3.Stelle = Dateiart

PRIV = Privilegstufe

HBA = Plattenadresse des Dateikennsatzes (oktal)

Anhang-CALL-Unterprogramme

```
10  REM  ********** BEISPIEL CALL 97 **********
20  REM
30  REM
40  REM  ********** DIMENSIONIERUNGEN **********
50  REM
60  REM
65  IF ERR 0 GOTO 1000
70  DIM 1%,J,F1,F9,2%
80  DIM A$(3)
90  DIM F$(15)
100 DIM H$(125),L$(125),H1$(125)
110 REM
120 REM  ********** ÜBERSCHRIFT AUFBEREITEN **********
130 REM
150 LET H$=" ",H$
160 LET H$(1,8)="FILENAME"
170 LET H$(16,19)="PROT"
180 LET H$(27,30)="COST"
190 LET H$(38,41)="SIZE"
200 LET H$(49,55)="ACCOUNT"
210 LET H$(63,65)="AGE"
220 LET H$(73,76)="HLSA"
230 LET H$(84,87)="TYPE"
240 LET H$(95,98)="PRIV"
250 LET H$(106,108)="HBA"
260 LET H1$="-",H1$
270 OPEN #2,"$LPT"
280 REM  ********** ÜBERSCHRIFT DRUCKEN **********
290 LET P9= P9+1
300 PRINT #2;"CALL 97 BEISPIEL";TAB(90);"SEITE: ";P9";"←215←"
310 PRINT #2;"ALLE VARIABLEN DIM 2%←215←←215←"
320 PRINT #2;H$
330 PRINT #2;H1$
340 LET Z1=0
350 CALL 97,U,R,F$,A,T,S,Q,C,I,D,L,H
370 IF R=-1 GOTO 900                    /* DATEIENDE?
380 REM  ********** KONVERTIERUNG DEZIMAL NACH OKTAL ****
390 FOR J=1 TO 10
400 IF H< 8*J GOTO 420
410 NEXT J
420 FOR J1=1 TO J
430 LET H1= INT (H/8 (J-J1))*10 (J-J1)
440 LET H2= H2+H1
450 LET H= H-( INT (H/8 (J-J1)))*8 (J-J1)
460 NEXT J1
470 LET H= H2
480 LET H2= 0
490 REM  ********** HOLEN DATEITYP AUS T **********
491 REM  ********** T1 = DATEITYP DEZIM. **********
492 REM  ********** T2 = DATEITYP OKTAL  **********
493 REM  ********** R1 = PROCESSOR-TYP   **********
```

Anhang-CALL-Unterprogramme

```
494 REM  ********** P = DATEISCHUTZ      **********
495 REM  ********** P1= 1. ZIFFER DATEISCHUTZ *****
496 REM  ********** P2= 2. ZIFFER DATEISCHUTZ *****
500 LET T1= T-32* INT(T/32)
510 LET T2= T1- (INT(T1/8)*8)+(INT(T1/8)*10)
520 REM  ********** HOLEN KONTROLLZIFFER (R,L,I) AUS T
530 LET R1= ((T-512*INT(T/512))-T1)/64
540 REM  ********** HOLEN DATEISCHUTZ AUS T *******
550 LET P= (T-64*R1-T1)/512
560 LET P1= INT (P/8)
570 LET P2= INT (P-8*P1)
580 REM  ********** TRENNEN PRIV,GROUP,USER AUS ACCOUNT
590 LET L1= INT (A/16384)
600 LET G= INT ((A-L1*16384)/64)
610 LET U1= A-L1*16384-G*64
620 REM  ******* STUNDEN SEIT ERSTELLUNG U. LETZTEM OPEN
630 LET D= SPC (2)-D
640 LET L= SPC (2)-L
650 REM  ********** DRUCKZEILE AUFBEREITEN **********
660 LET R1= R1*100
680 LET A$(1,3)= T2 USING "###"
690 LET L$= " ",L$
700 LET L$(1,15)= F$
710 LET L$(17,17)= P1 USING "#"
720 LET L$(18,18)= P2 USING "#"
730 LET L$(25,30)= C/10 USING "###.##"
740 LET L$(38,41)= S USING "####"
750 LET L$(49,51)= G USING "###"
760 LET L$(52,52)= ","
770 LET L$(53,55)= U1 USING "###"
780 LET L$(61,65)= D USING "#####"
790 LET L$(72,76)= L USING "#####"
800 LET L$(85,87)= A$
810 LET L$(97,97)= L1 USING "#"
820 LET L$(104,108)= H USING "#####"
830 PRINT #2;L$
840 REM  ********** NEUE SEITE ???? **********
850 LET Z1= Z1+1
860 IF Z1< 34 GOTO 350
870 PRINT #2;" 214 ";
880 GOTO 290
900 CLOSE #2
910 CHAIN ""
1000 PRINT
1010 PRINT "BASIC-FEHLER #";SPC8;"ZEILE =";SPC10
1020 CHAIN ""
```

Anhang-CALL-Unterprogramme

11.25 CALL 98

Senden eines beliebigen Systemkommandos an eine beliebige Task. Dadurch wird die Möglichkeit geboten, jeden Teilnehmer ab- und anzumelden und ein beliebiges Programm zu starten.

Syntax:

<CALL> :: = CALL <N-EXPR>,<N-VAR1>,<S-VAR>,<N-VAR2>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 98 sein.

<N-VAR1> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen ist die BA-Nummer des Teilnehmers (Task-Nummer) vorzugeben, an den das Kommando gesendet werden soll.

<S-VAR> = String-Variable. In diesem String ist das Systemkommando, das an den mit <N-VAR1> adressierten Teilnehmer gesendet werden soll, vorzugeben. Zugelassen sind alle Systemkommandos, die unter "SCOPE" über die Tastatur eingegeben werden können.

<N-VAR2> = Numerische Variable, Vektor- oder Matrix-Element. In dieser Variablen wird nach Ausführung des CALL eine Statusmeldung übergeben.

<N-VAR2>	Bedeutung
= 0	Funktion fehlerfrei durchgeführt.
<> 0	Funktion konnte nicht durchgeführt werden (Das Kommando wurde nicht an die adressierte Task übergeben).

Anhang-CALL-Unterprogramme

An/Abmelden einer Task:

Zum Starten eines Programmes mit der Anweisung "CALL 98" ist es erforderlich, die Anweisung "CALL 98" zweimal auszuführen. Bei der ersten Ausführung muß an die adressierte Task der Code 334 (oktal) gesendet werden. Man bewirkt damit, daß die adressierte Task definiert abgemeldet wird.
Im Anschluß daran kann durch Übersendung eines beliebigen Systemkommandos ein beliebiges Programm in der adressierten Task gestartet werden.

Anmerkung: Nachdem die Anweisung "CALL 98" abgesetzt ist, muß zwei Sekunden gewartet werden (Signal 3,20), bevor der Status in <N-VAR2> abgefragt wird.

Beispiel: Starten des Programmes "FAKTURA" in der Phantom-Task (BA-Nummer = 1).

```
.
1000 LET S= 0              /* <N-VAR2> AUF 0
1005 LET P= 1              /* BA-NUMMER = 1
1010 LET A$= "←334← "      /* CODE FÜR ABMELDEN
1015 CALL 98,P,A$,S        /* PHANTOM-TASK ABM.
1020 SIGNAL 3,20           /* 2 SEKUNDEN WARTEN
1025 IF S GOTO ...         /* STATUSABFRAGE
1030 LET A$= "FAKTURA"     /* PROGRAMMNAMEN LADE
1035 CALL 98,P,A$,S        /* PROGRAMM STARTEN
1040 SIGNAL 3,20           /* 2 SEKUNDEN WARTEN
1045 IF S GOTO ...         /* STATUSABFRAGE
.
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablentyp vorgegeben.

Anhang-CALL-Unterprogramme

11.26 CALL 99

Übernehmen oder ändern von Systemdatum und Systemzeit.

Syntax:

<CALL> :: = CALL <N-EXPR>,<S-VAR>

Funktion:

CALL = Anweisung.

<N-EXPR> = Beliebiger numerischer Ausdruck. Der Wert dieses Ausdrucks repräsentiert die Nummer des aufzurufenden Unterprogrammes.
<N-EXPR> muß in diesem Fall = 99 sein.

<S-VAR> = String-Variable. Dieser String muß für mindestens 25 Byte dimensioniert sein. <S-VAR> 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 die aktuelle Systemzeit an das Anwenderprogramm übergeben werden, muß <S-VAR> ein Leerstring sein.

Beispiel: Systemdatum und Systemzeit übernehmen.

```
.
1000 LET A$= ""        /* <S-VAR> = LEERSTRING
1005 CALL 99,A$        /* 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			
Byte #	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5

MMM = Monat: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.

TT = Tag

JJJJ = Jahr

HH = Stunde

MM = Minute

SS = Sekunde

Systemdatum und Systemzeit ändern:

Sollen Systemdatum und Systemzeit geändert werden, muß <S-VAR> wie folgt aufbereitet sein:

Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	J	J	J	J	,	M	M	,	T	T	,	H	H	,	M	I	,	S	S						

JJJJ = Jahr

MM = Monat

TT = Tag

HH = Stunde

MI = Minute

SS = Sekunde

Als Trennzeichen kann jedes beliebige, nicht-numerische Zeichen verwendet werden.

Beispiel: Ändern des aktuellen Systemdatums. Das neue Datum ist in der geforderten Form einzugeben.

```
.
1000 INPUT "DATUM/ZEIT: "A$ /*DATUM/ZEIT
                              EINGEBEN
1005 CALL 99,A$             /*DATUM/ZEIT
                              ÄNDERN
.
```

Fehler:

Basic-Fehler	Bedeutung
# 38	- Falscher Variablentyp vorgegeben. - Datum/Zeit in falschem Format vorgegeben. - Datum oder Zeit unmöglich.

Anhang - Schnittstellen

12.1 TAMOS - Schnittstellen

Alle in TAMOS-Selektoren eingebundenen Anwenderprogramme müssen eine Reihe von Schnittstellenbedingungen aufweisen, um eine korrekte Ablaufsteuerung zu gewährleisten.

12.1.1 Starten eines Anwenderprogrammes

Unter TAMOS laufende Anwenderprogramme sind aus den entsprechenden Selektoren mit:

Programm-Nr. [CR]

aufzurufen (s. Bedienerhandbuch).
Zwischen dem Aufruf und dem eigentlichen Programmstart führt TAMOS folgende Arbeitsgänge durch:

- In der Datei TF.PORT wird ein Datensatz mit den folgenden Informationen abgestellt:
  - CPU - Zeit
  - Ablauf Modus (Run mode)
  - Kontogruppen-Nr., Kontobenutzer-Nr. (Group/User)
  - Selektor-Name 1. Ebene
  - Selektor-Name 2. Ebene
  - Selektor-Name 3. Ebene
  - Programmname
  - Benutzte log. Geräte (LU-Nummern)

  Der Datensatz wird für jeden Teilnehmer getrennt angelegt.

- In der Common area (gemeinsamer Teilnehmerbereich) stellt TAMOS, dem Teilnehmer der das Programm gestartet hat, folgende Daten bereit:
  - Selektor-Name 3. Ebene
  - Selektor-Name 2. Ebene
  - Selektor-Name 1. Ebene
  - Datum
  - Der Text: NACHRICHT:
  - Selektor-Parameter
  - Programmname
  - Installierte log. Geräte (LU-Nummern)

Anhang - Schnittstellen

- In die "Log-Datei" (TF. LOGFILE) wird ein Datensatz mit folgenden Informationen geschrieben:
  - Teilnehmer-Nummer
  - Kontogruppen-Nr., Kontobenutzer-Nr. (Group/User)
  - Startzeit
  - Kopfzeile
  - Status = "START"

Die Kanäle "0" und "1" belegt TAMOS für die Dateien

- TF.LOGFILE (Kanal 0) und
- TF.PARAM (Kanal 1).

Sie sollten also von Anwenderprogrammen nicht geschlossen werden.
Eine Ausnahme besteht allerdings für Anwenderprogramme die weder TAMOS-Unterprogramme aufrufen, noch beim Beenden zu TAMOS verzweigen.

12.1.2 Beenden eines Anwenderprogrammes

Jedes, unter TAMOS gestartete, Anwenderprogramm muß mit einem Verzweig ("CHAIN") zu einem der Programme:

- TA.END
- TA.NCO
- TA.ABO

beendet werden.

TA.END

Dieses Programm ist aufzurufen, wenn ein Anwenderprogramm korrekt beendet ist. Es schließt alle eröffneten Kanäle und vermerkt die Beendigung in der Log-Datei (TF.LOGFILE). Darüberhinaus werden in der Archiv-Datei die Copy-Flags gemäß den Selektor-Einträgen des entsprechenden Programmes aktualisiert.

Anmerkung: Sind beim Aufruf von TA.END die Kanäle 0 und 1 nicht eröffnet, wird stattdessen automatisch TA.ABO aktiviert.

TA.NCO

Anwenderprogramme, die abgebrochen werden sollen, ohne allerdings die Copy-Flags in der Archiv-Datei zu aktualisieren, sind mit TA.NCO zu beenden.

Dies kann zum Beispiel dann sinnvoll sein, wenn ein Anwenderprogramm zu einem Zeitpunkt abgebrochen wird, bevor es Datenbestände auf Magnetplatten (im Programmselektor eingetragene LU´s) verändert hat.

Anhang - Schnittstellen

TA.ABO

Muß ein Anwenderprogramm abgebrochen werden, das bereits Daten auf einer Magnetplatte verändert hat (also die Copy-Flags in der Archiv-Datei aktualisiert werden müssen), ist das Programm TA.ABO aufzurufen.
TA.ABO schließt alle eröffneten Kanäle und vermerkt den Programmabbruch in der Log-Datei.

Am Arbeitsplatz wird der Abbruch in der Nachrichtenzeile angezeigt, sobald wieder der TAMOS-Selektor erscheint.

Nach einem Programm, das ohne TA.ABO-Ausgang abgebrochen wird, ist es nicht möglich weitere Programme zu starten, da TAMOS nur noch Programme zur Datensicherung zulässt (Meldung in der Nachrichtenzeile und Rekonstruktionslauf erforderlich).

Anhang - Schnittstellen

12.1.3 Schnittstelle - Kopfzeile ausgeben

Zu Beginn eines jeden unter TAMOS laufenden Anwenderprogrammes, sollte das Standard-Unterprogramm "Kopfzeile ausgeben" aufgerufen werden.
Dieses Unterprogramm löscht den Bildschirminhalt und gibt die Kopfzeile (Zeile 0) sowie die Statuszeile (Zeile 24) aus.
Bevor das Unterprogramm aufgerufen wird, sind in einem String die Angaben "Release" und "Level" in folgender Form aufzubereiten:

```
X X Y /
|  |  |
|  |  +------ Schrägstrich, dient als Trennzeichen
|  +--------- Release
+------------ Level 0 bis 99
```

Kopfzeile und Statuszeile werden als Hintergrundzeichen ausgegeben!

Anmerkung: Programme die nur in der Phantom-Task laufen, müssen dieses Unterprogramm nicht aufrufen.

Das Unterprogramm

- Benutzte Variablen:
```
H$ (122) = Kopfzeile
V$ (4)   = Release + Level
P   2%   = Teilnehmernummer
```

- Unterprogramm:

```
1080 LET P=SPC 6
1090 CALL 3,P,H$
1100 PRINT 'CS';'SB';V$;H$(5,66);TAB(71,0);
          H$(67,74);TAB(0,24);H$(75,89);
1110 RETURN
```

12.1.4 Schnittstelle - Nachricht ausgeben

Ausgeben von Nachrichten auf der Nachrichtenzeile (Zeile 24). Es können sowohl eigens aufbereitete Texte (Nachrichten) als auch Nachrichten aus der Nachrichtendatei (TF.PARAM) ausgegeben werden.

Das Unterprogramm

- Benutzte Variablen:

```
E   2%     = Nachrichtennr. in TF.PARAM
M$  (50)   = Nachricht
```

- Aufruf:

```
1000 LET E = XXX    /* NACHR.-NUMMER LADEN
1005 GOSUB  1120    /* SPRUNG INS UPRO
.
oder
.
1000 LET M$ = "NACHRICHT" /* NACHRICHT LADEN
1005 GOSUB  1130    /* SPRUNG INS UPRO
```

- Unterprogramm:

```
1120 READ # 1,INT(E/10)+15,FRA(E/10)*510;M$;
1130 PRINT TAB(0,24);'LD';'LI';'SB';H$(75,89);'SF';'BEL';M$;'SB';
1140 RETURN
```

12.1.5 Schnittstelle - Logging

Eintragen von beliebigen Programmereignissen in die Log-Datei. Ein Eintrag besteht aus:

- Teilnehmernummer
- Kontogruppen-Nr., Kontobenutzer-Nr.
- Uhrzeit
- Programmname
- Nachricht

Die Nachricht kann ein beliebiger Text, bestehend aus maximal 50 Zeichen, sein.
Programmstart, Programmende oder Programmabbruch brauchen nicht protokolliert werden, da dies von TAMOS durchgeführt wird.

Das Unterprogramm

- Benutzte Variablen:
  L$ (126) = Zu schreibender Log-Datei Satz
  H$ (122) = Kopfzeile
  M$ (50) = Nachricht

- Aufruf:
```
500 LET  M$ = "NACHRICHT" /* NACHRICHT LADEN
505 GOSUB 1150            /* SPRUNG INS UPRO
```

- Unterprogramm:
```
1150 LET L$ = "2", H$ (96, 113)
1160 LET L$ (20, 21) = SPC 6
1170 LET L$ (22, 27) = SPC 5 USING "######"
1180 LET L$ (28, 33) = (SPC 2-INT (SPC 2/24)*24)
          *3600 + INT (SPC 3/10) USING "######"
1190 LET L$ (34) = M$
1200 WRITE #0,CHF 0; L$;
1210 RETURN
```

Anmerkung: Der String M$ muß vor Aufruf dieses Unterprogrammes mit der Nachricht geladen sein.

12.1.6 Schnittstelle - Fehlerbehandlung

Behandlung schwerwiegender, zum Abbruch führender Fehler im Anwenderprogramm.
Dieses Unterprogramm enthält die folgenden Funktionen:

- Rücksprung -1, ins Anwenderprogramm, wenn eine Basic-Fehlernr. >97 auftritt.
- Eintrag der Fehlermeldung in die Log-Datei
- Aufruf des TAMOS - Programmes "TA.ABO"

Die Meldung, daß das Programm abgebrochen wurde, wird von "TA.ABO" am Bildschirm angezeigt.

Das Unterprogramm

- Benutzte Variablen:
  L$ (126) = Zu schreibender Log-Datei Satz
  H$ (122) = Kopfzeile
  M$ (50) = Text für Log-Eintrag

- Aufruf:
```
. . . . IF ERR 0 GOSUB 1000
.
```

- Unterprogramm:
```
1000 IF SPC 8>97 RETURN -1
1010 IF ERR 0 GOTO 1070
1020 LET L$ = "SEGMENT-NAME"
1030 LET M$ = "BASIC-ERROR #    AT     IN", L$
1040 LET M$ (14, 15) = SPC 8 USING "##"
1050 LET M$ (21, 24) = SPC 10 USING "####"
1060 GOSUB 1150    /* LOGGING
1070 CHAIN "TA.ABO"
```

12.1.7 Schnittstelle - Kommunikation zwischen Phantom-Task und Master-Platz

Diese Schnittstelle ermöglicht den Austausch von Meldungen/Anweisungen zwischen dem in der Phantom-Task laufenden Programm und dem Bediener am Master-Platz. Das Unterprogramm beinhaltet die Funktionen:

- Ausgabe einer Meldung in der Statuszeile des Masterplatzes.
- Warten auf eine Meldung (Eingabe vom Masterplatz), die im gemeinsamen Bereich der Phantom-Task übergeben wird.

Das Unterprogramm

- Benutzte Variablen:
  M$ (50) = Nachricht und Antwort
  P (25) = Task-Nummer
  E (2%) = Nummer der Nachricht in TF.PARAM

- Aufruf:

```
1000 LET E = XXX       /NACHR.-NUMMER LADEN
1005 GOSUB  1220       /* SPRUNG INS UPRO
.
oder
.
1000 LET M$ = "NACHRICHT"
1005 GOSUB 1230      /* SPRUNG INS UPRO
```

- Unterprogramm:

```
1220 READ #1,INT(E/10)+15,FRA(E/10)*510;M$;
1230 LET P = -SPC6    /* TASK-NUMMER NEGATIV
1240 CALL 2,P,P,M$    /* MELDUNG ÜBERGEBEN
1250 READ # 1,14,137;M$(38);
1260 LET M$="←207←←376←←211←←376←←221←
      B←230←",M$(38),"←376←←212←"
1270 IF ERR 0 GOTO 1330       /* FEHLERAUSWERTUNG
1280 CALL 4,M$                /* NACHRICHT AUSGEBEN
1290 SIGNAL 3,250             /* 25 SEKUNDEN WARTEN
1300 CALL 3,P,P,M$            /* ANTWORT ABHOLEN
1310 IF P = -SPC6 GOTO 1250   /* KEINE ANTWORT
1320 RETURN                   /* RÜCKSPRUNG
1330 IF SPC8 <> 38 GOTO 1010 /* FEHLERBEHANDLUNG
1340 GOTO 1290
```

Anhang - Schnittstellen

Anmerkung:

Dieses Unterprogramm zeigt im Abstand von 25 Sekunden auf dem Master-Bildschirm (Nachrichtenzeile) die Meldung "SPOOL-STATUS" an und wartet auf eine Antwort, die im gemeinsamen Bereich der Phantom-Task abgestellt wird, d.h., sobald am Masterbildschirm die TAMOS Programmwahl erscheint, und der Bediener [CR] drückt, wird die von der Phantom-Task übergebene Meldung in der Nachrichtenzeile angezeigt. TAMOS erwartet danach eine Eingabe und übergibt sie in den gemeinsamen Bereich der Phantom-Task.

Anhang - Schnittstellen

12.1.8 Schnittstelle - Spooling

Das Spooling-System in Tamos ist so ausgelegt, daß die Jobs vom Anwenderprogamm kreiert werden müssen. Dazu muß von dem kreierenden Programm eine Job-Beschreibung in die Spool-Datei (TF.SPOOLQUEUE) überstellt werden.

Kreieren eines Textdatei-Jobs

Folgende Informationen müssen in der Spool-Datei abgestellt werden:

- Job-Typ (0 = Textdatei-Job)
- Job-Beschreibung (beliebiger Job-Name, der am Master-Bildschirm erscheint, wenn die Spool-Datei angezeigt wird).
- LU/Textdatei-Name
- Löschcode: 0 = Textdatei nach Ausdruck nicht löschen
  1 = Textdatei nach Ausdruck löschen
- Job-Skip-Erlaubnis (Löschcode + 100 = Job-Skip nicht erlaubt).
- Anzahl Durchläufe (1 - 7999)
- Papiercode: 0 und 1 = Standardpapier
  2 bis n = kein Standardpapier
- Papierbeschreibung (beliebiger Text zur Bedienerinformation bei Papierwechsel).
- Papierformat (1. und letzte Druckposition, sowie Zeilenanzahl pro Seite).
- Seitennummer der erste druckbaren Seite.
- Parameter aus der Datei "TF.PORT":
  Job-Erstellungszeit
  Runmode
  Teilnehmernummer
  Programmselektor-Koordinaten

Diese Parameter sind Werte des Mutterprogrammes, das den Job kreiert. Soll ein Parameter eingegeben werden der von denen in TF.PORT abweicht, kann dieser entsprechend geändert werden, z.B.: Runmode (siehe auch Standard-Routine).

Anhang - Schnittstellen

Kreieren eines Basic-Programm-Jobs

Die folgenden Informationen müssen in die Spool-Datei überstellt werden:
- Job-Typ (1 für Basic-Programm-Job)
- Job-Beschreibung (siehe Textdatei-Job)
- LU=/Programmdatei-Name
- Abbruch-Kontroll Code:
  0 = Programm darf abgebrochen werden
  10 = Programmabbruch bewirkt Rekonstruktion
- Job-Skip-Erlaubnis (Abbruchcode + 100 = Keine Skip-Erlaubnis)
- Runmode-Kontroll-Code:
  Abbruchcode + 0 = Runmode des Mutterprogramms
  Abbruchcode + 1 = Runmode 0
- Anzahl Durchläufe
- Papiercode: 0 = Drucker wird nicht benötigt
  1 = Standardpapier
  2 bis n = Kein Standardpapier
- Papierbeschreibung (frei für Benutzerinformation) *
- Papierformat (frei für Benutzerinformation) *
- Seitennummer (frei für Benutzerinformation) *
- Parameter aus der Datei "TF.PORT":
  Job-Erstellungszeit
  Runmode
  Teilnehmer-Nummer
  Selektor-Koordinaten

Anmerkung:

Diese Parameter sind Werte des Mutterprogrammes, das den Job kreiert. Soll ein Parameter von den Angaben in der Datei "TF.PORT" abweichen, kann dieser entsprechend geändert werden, z.B.: Runmode (siehe auch Standard-Routine).

Der "Common-Bereich" des Phantom-Port wird vom Spooler benutzt, um dem Job vor dem Start Parameter aus der Spool-Datei zu übergeben. Die mit "*" gekennzeichneten Parameter werden ebenfalls übergeben und können zu beliebigen Programmsteuer-Informationen benutzt werden.

Anhang - Schnittstellen

Das Unterprogramm:

- Benutzte Variable:

H$	(122)	= Kopfzeile (siehe auch Unterprogramm "Kopfzeile ausgeben").
M$	(50)	= Nachricht (Hilfsfeld).
D$	(20)	= Job-Beschreibung.
P6$	(18)	= LU/Text- oder Programmdatei-Name
S$	(25)	= Papierbeschreibung.
F$	(25)	= Beliebige Benutzerinformation.
T	(1%)	= Job-Typ.
T5	(2%)	= Job-Erstellungszeit.
P8	(1%)	= Runmode.
G5	(2%)	= Teilnehmer-Nummer.
C5	(1%)	= Selektor-Koordinate 1. Level
S6	(1%)	= Selektor-Koordinate 2. Level
P7	(1%)	= Selektor-Koordinate 3. Level
S9	(1%)	= Löschcode und Job-Skip Erlaubnis bei Textdatei-Jobs, bzw. Abbruchcode, Runmode und Job-Skip-Erlaubnis bei Basic-Programmjobs.
S8	(1%)	= Anzahl der Durchläufe.
S7	(1%)	= Papiercode.
L7	(1%)	= 1. Druckposition.
L8	(1%)	= Letzte Druckposition.
L9	(1%)	= Zeilen pro Seite.
C8	(1%)	= Seitennummer der 1. zu druckenden Seite.
C9	(1%)	= Hilfsfeld.
E	(2%)	= Fehlernummer.

- Aufruf:

```
1000 GOSUB  705              /* SPRUNG INS UPRO
1005 IF E = 164 GOTO . . .   /* ..........
```

Anmerkung:

Alle oben aufgeführten Variablen sind vor dem Aufruf des Unterprogrammes zu dimensionieren.

Außerdem müssen die Variablen:
H$, D$, P6$, F$, T, S9, S8, S7, L7, L8, L9, und C8
aufbereitet sein.

- Das Unterprogramm:

```
705 REM ***JOB IN JOB-DATEI EINTRAGEN***
710 OPEN #2, "TF.PORT"
720 READ #2,0,22 + SPC 6*44; T5, P8, G5,  C5, S6, P7;
725 CLOSE #2
730 OPEN #2, "TF.SPOOLQUEUE"
750 FOR C9 = 0 TO CHF 2-1
760 READ #2, C9; M$
770 IF M$ (1,1) = " " GOTO 800
780 NEXT C9
785 LET E = 164
795 GOTO 845
800 LET E  = 0
810 LET C9 = 0
820 LET T5 = (SPC 2- INT (SPC2/24)*24)*3600+ INT
           (SPC3/10)
830 IF LEN D$< 3 LET D$ = H$ (5)
840 WRITE #2,-2; D$, T, T5, P8, G5, C5, S6, P7, P6$, S9,
                 S8, S7, S$, L7, L8, L9, C9, C8, F$;
845 CLOSE #2
850 RETURN
```

Anmerkung:

Einem Basic-Programm-Job wird vom Spooler im gemeinsamen Bereich mittels CALL 2 folgende Information übergeben:

```
LET P = 1
CALL 2,P,H$,F$,S$,L7,L8,L9,S7,S8,C8
```

Ein negativer Wert in der Variablen S7 besagt, daß ein Papiercode-Wechsel vorliegt, also, bei Drucker-Jobs entsprechende Einricht-Routinen abgewickelt werden können.

Ein Basic-Programm ist zu beenden, indem es eines der folgenden Programme aufruft:

- TA.END,
- TA.NCO,
- TA.ABO

Die zu benutzenden logischen Einheiten eines Basic-Programm-Jobs sind im Programmselektor des "Mutter-Programmes" definiert. Das Mutterprogramm sollte also, wenn es selbst keine Aktualisierung bedingt, mit einem Verzweig zu TA.NCO versehen sein.

12.1.9 Aufbau der Bildschirm - Kopfzeile

Position	Inhalt
0 - 1	Level
2	Release
4 - 24	Beschreibung 3. Ebene
25 - 45	Beschreibung 2. Ebene
46 - 65	Beschreibung 1. Ebene
71 - 78	Datum (JJ.MM.TT)

12.1.10 Aufbau der Nachrichtenzeile (Zeile 24)

Position	Inhalt
0 - 6	Text: "NACHRICHT"
15 - 64	Nachricht

12.1.11 Im gemeinsamen Teilnehmerbereich übergebene Parameter

Byte #	Inhalt
1 - 4	Release
5 - 25	Selektor-Name, 1. Ebene
26 - 46	Selektor-Name, 2. Ebene
47 - 66	Selektor-Name, 3. Ebene
67 - 74	Datum (JJ.MM.TT)
75 - 89	Text: "NACHRICHT:"
90 - 92	Firmennummer
93 - 95	Parameter-Tabellennummer
96 -110	Programmname
111-113	Programmnummer
114-121	Benutzte LU´s

Anhang - Schnittstellen

12.2 Betriebssystem - Schnittstellen

Basic ermöglicht mit der Anweisung "CHAIN" den Aufruf beliebiger Processoren.
Innerhalb der Processoren werden drei Gruppen unterschieden:

- Processoren, die durch das aufrufende Programm mit Parametern versorgt werden, ablaufen und daran anschließend die Steuerung an ein beliebiges Programm bzw. an einen beliebigen Processor übergeben.
- Processoren, die durch das aufrufende Programm mit Parametern versorgt werden, ablaufen und anschließend die Steuerung an "SCOPE" übergeben.
- Processoren, die nach dem Aufruf (durch ein Anwenderprogramm) über die Tastatur mit Parametern versorgt werden, ablaufen und danach die Steuerung an "SCOPE" übergeben.

Anhang - Schnittstellen

12.2.1 Processoren, die zum Anwenderprogramm verzweigen.

Der Aufruf dieser Processoren erfolgt mit der Anweisung "CHAIN" und die jeweils benötigten Parameter werden als String-Literal oder in einer String-Variablen übergeben. Mit einem Aufruf können mehrere Processoren aktiviert werden, die nacheinander ablaufen sollen. Der Abschluß eines Kommandos ist dem jeweiligen Processor durch eines der Trennzeichen:

> (größer als) bei dem COPY-Processor
/ (Backslash) bei dem INSTALL-Processor
<- (Pfeil links) bei allen anderen Processoren

mitzuteilen.
Sobald eines der Zeichen auftritt, übernimmt das im Anschluß daran spezifizierte Programm/Processor die weitere Ablaufsteuerung.
Folgende Processoren sind per Programm aufrufbar, sowie mit Parametern zu versorgen, und verzweigen ohne zusätzlichen Eingriff in ein beliebig definierbares Programm.

- BASIC
- COPY
- INSTALL
- REMOVE
- RUN
- SAVE

Anmerkung:

Nachdem der String, mit dem das Programm aufgerufen wurde, abgearbeitet ist, wird die weitere Verarbeitung mit der ersten Anweisung des aufgerufenen Programmes fortgesetzt.
Soll in einem Programm allerdings bei einer bestimmten Zeilennummer aufgesetzt werden, ist dieser Einsprung innerhalb des Anwenderprogrammes zu verwalten.

Fehlermeldungen der Processoren:

Alle diese Processoren übergeben beim Auftreten von Fehlern sog. Fehlerschlüssel im gemeinsamen Teilnehmerbereich. Diese Fehlerschlüssel werden dort im ersten Wort, bzw. in den ersten beiden Worten abgestellt.
Wird ein Processor aufgerufen, der sich nicht auf der angegebenen Magnetplatte befindet, erscheint die Meldung:

NO SUCH PROCESSOR

und das System verzweigt nach "SCOPE".

Anhang - Schnittstellen

12.2.1.1 BASIC - Processor

Im Basic-Processor können die Kommandos "LOAD" und "DUMP" per Kommandostring aufgerufen werden. Nach der Ausführung des Kommandos wird die Ablaufsteuerung an ein beliebiges Anwenderprogramm bzw. einen beliebigen Processor übergeben.

"LOAD" - Kommando im Kommandostring

Der Kommandostring in dem die erforderlichen Parameter an den Processor übergeben werden hat den Aufbau:

BASIC <PROG-NAME> LOAD <TEXTDAT>←337←<PROGR/PROC1>←337←<PROGR/PROC2>

Funktion:

BASIC = Name des aufzurufenden Processor (in diesem Fall BASIC)

<PROG-NAME> = Dummy-Programm. Das angegebene Basic-Programm <PROG-NAME> sollte so klein wie möglich sein. Diese Angabe ist aus syntaktischen Gründen erforderlich.

LOAD = Kommando, bewirkt das Laden eines in einer Textdatei abgestellten Programmes.

<TEXTDAT> = Name der Textdatei, in der das zu ladende Programm steht.

← oder ←337← = Trenncode. Dieser Code zeigt dem Basic-Processor, daß die Steuerung an das im Anschluß angegebene Programm bzw. Processor übergeben werden soll.

<PROG/PROC1> = Programm bzw. Processor, an den die Steuerung übergeben werden soll, sobald das LOAD-Kommando korrekt ausgeführt ist.

← oder ←337← = Trenncode. Im Anschluß daran muß ein Programm- oder Processorname angegeben sein. Andernfalls wird nach "SCOPE" verzweigt.

<PROGR/PROC2> = Programm oder Processor, an den die Steuerung übergeben werden soll, wenn bei LOAD ein Fehler auftritt.

Anhang - Schnittstellen

Anmerkung:

Vor dem Laden des Programmes aus der Textdatei, wird ein "NEW" -Kommando durchgeführt.

Ist kein Trenn-Code (←) angegeben, behält der Basic-Processor die Kontrolle bzw., folgt auf einen Trenn-Code keine weitere Angabe, geht die Steuerung an "SCOPE" über.

Fehler:

Tritt bei "LOAD" ein Fehler auf, wird die Fehlermeldung in den ersten beiden Worten (I% - Variable) des gemeinsamen Teilnehmerbereiches abgestellt. Das Laden wird abgebrochen und die im Kommandostring folgende Anweisung (Kommando) ausgeführt.
Die Bedeutung der im gemeinsamen Bereich übergebenen Fehlermeldung ist wie folgt:

1. Wort: Zeilennummer in welcher der gemeldete Fehler (Syntax, Format) beim Laden der Textdatei auftrat. Andernfalls ist der Inhalt dieses Wortes = 0.

2. Wort: Fehlernummer. Entweder Basic-Fehlernummer oder eine der im Anschluß an die Beschreibung der Basic-Kommandos angegebenen Fehlernummern.

Wird bei der Angabe <PROG-NAME> ein Fehler festgestellt, erscheint die entsprechende Fehlermeldung am Bildschirm und das System verzweigt nach SCOPE.

Beispiel:

Laden eines in einer Textdatei stehenden Programmes und anschließendes Starten dieses Programmes.

Als <PROG-NAME> ist "DUMMY" angegeben.

```
.
CHAIN "BASIC DUMMY LOAD TEXT ← RUN"
.
```

Anhang - Schnittstellen

"DUMP" - Kommando im Kommandostring

Der Kommandostring, in dem die erforderlichen Parameter an den Processor übergeben werden, hat folgenden Aufbau:

BASIC <PROG-NAME> DUMP <TEXTDAT> ←337← <PROGR/PROC>

Funktion:

BASIC = Name des aufzurufenden Processors (in diesem Fall BASIC).

<PROG-NAME> = Das Programm, das in die mit <TEXTDAT> bezeichnete Textdatei ausgegeben werden soll. Dieses Programm wird, nachdem das Kommando erteilt ist, geladen.

DUMP = DUMP-Kommando, bewirkt das Ausgeben eines BASIC-Programmes in eine Textdatei oder auf einem Drucker.

<TEXTDAT> = Name der Textdatei, in die das Programm ausgegeben werden soll.
Wird ein Drucker-Driver angegeben, erfolgt die Ausgabe über die entsprechende Peripherie.
Bei Ausgaben in eine Textdatei kann das Steuerzeichen "!" zum Erstzen der Datei verwendet werden.

<- oder<-337<- Trenncode. Dieser Code zeigt dem Basic-Processor, daß die Steuerung an das im Anschluß codierte Programm bzw. den angegebenen Processor übergeben werden soll.

<PROG/PROC> = Programm bzw. Processor, an den die Steuerung übergeben werden soll.

Anmerkung:

Ist kein Trenn-Code (←) angegeben, bleibt die Kontrolle beim Basic-Processor. Folgt auf einen Trenn-Code keine weitere Angabe, übernimmt "SCOPE" die weitere Kontrolle.

Anhang - Schnittstellen

Fehler:

Entsteht bei der Ausführung von "DUMP" ein Fehler, wird die Fehlermeldung im ersten Wort des gemeinsamen Teilnehmerbereiches abgestellt. Die Ausgabe wird abgebrochen und die im Kommandostring folgende Anweisung (Kommando) ausgeführt.

Die im gemeinsamen Bereich übergebene Fehlermeldung repräsentiert eine Basic-Fehlernummer von 1 bis 99 oder eine der im Anschluß an die Beschreibung der Basic-Kommandos angegebenen Fehlernummern.

Ist die Angabe von <PROG-NAME> fehlerhaft, erscheint am Bildschirm eine Fehlermeldung und das System verzweigt nach "SCOPE".

Beispiel:

Ausgabe des Programmes "PROG01" in eine Textdatei mit dem Namen "T.PROG01". Diese Datei ist bereits vorhanden und soll ersetzt werden.
Im Anschluß daran, übernimmt das Programm "GEN" die weitere Steuerung.

```
.
CHAIN "BASIC PROG01 DUMP T.PROG01! ← GEN"
.
```

Anhang - Schnittstellen

Sonstige Kommandos im Basic-Processor

Ebenso wie "LOAD" und "DUMP" können auch alle anderen verfügbaren Anweiungen in einem Kommandostring vorgegeben werden. Allerdings besteht, nachdem sie ausgeführt sind, nicht die Möglichkeit, die weitere Kontrolle an ein anderes Programm oder Processor zu übergeben, sondern sie verzweigen zum Kommandomodus des Basic-Processors.

Beispiel:

Laden eines Programmes und ausgeben der Zeilen 100 bis 200.

```
.
PRINT TAB (0,5)      /* CURSORPOSITIONIERUNG
CHAIN "BASIC PROG1   100LIST200"
.
```

Anmerkung:

Strings dürfen, sofern ein Kommando angegeben ist, das die Steuerung an den Basic-Processor übergibt, jeweils nur ein solches Kommando enthalten.

Zusätzliche Fehlernummern:

Fehler #	Bedeutung
103	Dateiname bereits für Datei anderen Typs vergeben.
104	Dateiname belegt (Datei wird angelegt/ ersetzt)
105	Dateiname belegt und kein "!" angegeben.
106	Dateiname unter einem anderen Konto belegt.
107	Dateiname durch permanente Systemdatei belegt
108	Nicht genügend freie Blöcke auf der LU.
109	Nicht genügend freie Blöcke auf dem Konto.
110	Syntaktischer Fehler bei der Angabe von COST und/oder PROTECTION.
111	Unzulässiger Dateiname.
112	Logische Einheit ist nicht bereit.

12.2.1.2 COPY - Processor

Der Aufruf des "COPY"- Processors aus einem Programm ist vom Aufbau her identisch mit dem Aufruf über die Arbeitsplatztastatur.

Aufbau des Kommandostrings:

COPY [?][(SA:SL)] <D-NAME1> [!]{←/=} <D-NAME2> <PROC/PROG>

Funktion:

COPY = Der Name des aufzurufenden Processors (in diesem Fall COPY).

? = Ist ein "?" angegeben, bedeutet das, daß die Zieldatei mit der Quelldatei verglichen wird.

(SA:SL) = Nur bei zusammenhängenden Dateien!
Es wird die Größe der Ausgabedatei angegeben. Diese Angabe ermöglicht der Zieldatei, eine größere Kapazität zuzuweisen als die Quelldatei benötigt.

<D-NAME1> = Der Name der Zieldatei. Liegt sie auf einer logischen Einheit <> 0, ist vor dem Dateiname die LU-Nummer anzugeben (LU/Dateiname).

! = Das Zeichen "!" ist dann anzugeben, wenn die Zieldatei bereits vorhanden ist und durch die Quelldatei ersetzt werden soll.

← oder = = Trennt den Zieldateinamen von dem der Quelldatei(en).

<D-NAME2> = Der Name der Quelldatei. Liegt sie auf einer logischen Einheit <> 0, ist vor dem Dateinamen die LU-Nummer anzugeben (LU/Dateiname)!
Ist die Zieldatei allerdings eine Textdatei, können mehrere Quelldateien angegeben werden. Diese werden dann in der Reihenfolge ihres Auftretens in die Zieldatei kopiert.

← oder ←276← = Trenncode. Der Code zeigt dem COPY-Processor, daß die Steuerung an das im Anschluß codierte Programm, bzw. den angegebenen Processor übergeben werden soll.

<PROC-PROG> = Programm bzw. Processor, an den die Steuerung übergeben werden soll.

Achtung!

Wird die Steuerung an ein Basic-Programm übergeben, muß vor dem Namen des Programmes unbedingt "RUN" codiert und durch ein Leerzeichen vom Namen getrennt sein.

Wurde mit dem COPY-Processor ein Vergleich durchgeführt und keine Unterschiede festgestellt, erscheint am Bildschirm die Meldung:

```
VERIFIED !
```

bzw. wenn Unterschiede festgestellt wurden, die Meldung:

```
VERIFIED WITH xxx ERRORS
```

(xxx = Anzahl festgestellter Unterschiede)

Kopiervorgänge mit dem COPY-Processor werden mit der Meldung:

```
COPIED !
```

beendet. Die Meldungen erscheinen am Bildschirm ab der jeweils aktuellen Cursorposition.

Beispiel:

Kopieren (Ausgeben) der Textdatei "T.PROGO" am Nadeldrucker ($LPT).

```
.
.
CHAIN "COPY $LPT = T.PROGO>RUN GEN"
.
```

Daran anschließend übernimmt das Programm "GEN" die weitere Steuerung.

Anhang - Schnittstellen

Fehler:

Treten während des Kopier/Vergleichsvorganges Fehler auf, wird die entsprechende Fehlermeldung am Bildschirm ab der aktuellen Cursorposition angezeigt. Anschließend verzweigt das System nach "SCOPE"

Anmerkung:

Nach Fehlern bei Vergleichsoperationen erscheint zusätzlich zu
"VERIFIED WITH xxx ERRORS"
die Meldung
"NO SUCH PROCESSOR"
und dann erst verzweigt das System nach "SCOPE".

12.2.1.3 INSTALL - Processor

Beim Aufruf des "INSTALL" - Processors aus einem Anwenderprogramm ist der Kommandostring in der gleichen Form aufzubereiten, wie die Parameter bei dem manuellen Aufruf unter "SCOPE" vorgegeben werden.

Aufbau des Kommandostrings:

INSTALL 0.<PLATTEN #>/<PROG-PROC>

Funktion:

INSTALL = Name des aufzurufenden Processors (in diesem Falle INSTALL).

0.<PLATTEN => = Controller-Nr. und physikalische Einheiten-Nr.

Als Controller-Nr. ist nur "0."zulässig.

<PLATTEN #>

0 = Festplatte, Laufwerk 0
1 = Wechselplatte, Laufwerk 0
2 = Festplatte, Laufwerk 1
3 = Wechselplatte, Laufwerk 1
4 = Festplatte, Laufwerk 2
5 = Wechselplatte, Laufwerk 2
6 = Festplatte, Laufwerk 3
7 = Wechselplatte, Laufwerk 3

/ oder ←374← = Trenncode. Dieser Code zeigt dem Install-Processor, daß die Steuerung an das im Anschluß codierte Programm bzw. den angegebenen Processor übergeben wird.

<PROG/PROC> = Programm- bzw. Processor-Name, an den die Steuerung übergeben werden soll.

Anmerkung:

Eine Magnetplatte die auf diese Art angemeldet wird, muß zu einem früheren Zeitpunkt schon einmal angemeldet gewesen sein (INSTALL unter SCOPE), da ihr bei diesem Verfahren die LU-Nr. der vorherigen Anmeldung erneut

zugewiesen wird.

Beispiel:

Installieren einer Wechselplatte auf Laufwerk 0 , und anschließendes Übergeben der Steuerung an das Programm "PROG01"

```
CHAIN "INSTALL 0.1/PROG01"
```

Fehler:

Tritt im Verlauf des INSTALL ein Fehler auf, wird die entsprechende Fehlermeldung im ersten Wort (1%-Variable) des gemeinsamen Teilnehmerbereich abgestellt, der Vorgang abgebrochen und das im Kommandostring angegebene Programm bzw. Processor aufgerufen.

Fehlermeldungen die im gemeinsamen Bereich übergeben werden können:

Fehler #	Bedeutung
1	"CONFIG"-Datei nicht im System.
2	Unzulässiges INSTALL/REMOVE Kommando erteilt.
3	Logische Einheit ist nicht angemeldet.
4	Logische Einheit kann vom aufrufenden Teilnehmer nicht abgemeldet werden.
5	Fehler im Dateikennsatz, (Verlust der Datei).
7	Zugriff aufgrund der Privilegierung nicht erlaubt.
8	Physikalische Einheit ist bereits belegt

12.2.1.4 REMOVE - Processor

Der Aufruf des "REMOVE" - Processors aus einem Anwenderprogamm ist entsprechend dem Aufruf unter "SCOPE", mit der abzumeldende LU-Nummer, zu parametrieren.

Der Aufbau des Kommandostrings:

REMOVE <LU#> <- <PROG/PROC>

Funktion:

REMOVE = Name des aufzurufenden Processors (in diesem Fall REMOVE)

<LU#> = Die abzumeldende logische Einheiten-Nr. (0 bis 32767).

← oder ←337← = Trenncode. Dieser Code bewirkt, daß die Steuerung an das im Anschluß codierte Programm, bzw. den angegebenen Processor übergeben wird.

<PROG/PROC> = Programm bzw. Processor, an den die Steuerung übergeben wird.

Beispiel:

Abmelden (REMOVE) der log. Einheit 1

CHAIN "REMOVE 1<-PROG"

Nach durchgeführter Abmeldung geht die Steuerung an das Programm "PROG" über.

Fehler:

Stellt der Remove-Processor einen Fehler fest, wird eine Fehlermeldung im ersten Wort (1%-Variable) des gemeinsamen Bereichs des Teilnehmers abgestellt. REMOVE wird abgebrochen und das angegebene Programm bzw. Processor wird aufgerufen.

Die Fehlerschlüssel die übergeben werden sind identisch mit dem des INSTALL.

Anhang - Schnittstellen

Fehler #	Bedeutung
2	Unzulässiges INSTALL/REMOVE-Kommando erteilt.
3	Logische Einheit ist nicht angemeldet.
4	Logische Einheit kann vom aufrufenden Teilnehmer nicht abgemeldet werden.

12.2.1.5 RUN - Processor

Der "RUN" - Processor wird automatisch beim Aufruf eines beliebigen Basic-Programmes aktiviert. Findet der Aufruf innerhalb eines laufenden Anwenderprogrammes statt, wird der initiierende Kommandostrings beendet, und ein Laden weiterer Processoren bzw. Anwenderprogramme ist vom gestarteten Programm zu verwalten.

Aufbau des Kommandostrings:

[RUN] <PROGRAMMNAME>

Funktion:

RUN = Name des aufzurufenden Processors (in diesem Falle RUN). Diese Angabe kann entfallen, da bei Angabe eines Basic-Programmes unter <PROGRAMMNAME> "RUN" grundsätzlich aktiviert wird.

<PROGRAMMNAME> = Zu startendes Basic-Programm.

Beispiel:

Aufruf eines Basic-Programmes mit dem Namen "PROG01"

```
CHAIN "PROG01"
```

Fehler:

Ist das zu ladende Programm nicht auf der Magnetplatte gespeichert oder ist als <PROGRAMMNAME> kein Basic-Programm angegeben, wird dies durch:

"NO SUCH PROCESSOR"

gemeldet. Anschließend verzweigt das System nach "SCOPE".

12.2.1.6 SAVE - Processor

Beim Aufruf des "SAVE"-Processors aus einem Anwenderprogramm, ist der Kommandostring für "SAVE" in der gleichen Form aufzubereiten, wie die Parameter bei einem manuellen Aufruf unter "SCOPE" .
Der Aufruf von "SAVE" ohne zusätzliche Processor-Aufrufe in demselben Kommandostring (z.B. BASIC) ist allerdings nicht sinnvoll.

Aufbau des Kommandostrings:

SAVE [#<PART>#] [<<PP>>] [<COST>] [[<LU>/] <DATEINAME> [!]] ←<PROG/PROC>

Funktion:

SAVE = Name des aufzurufenden Processors (in diesem Fall SAVE).

#<PART># = Partitiongröße in KB.
Das Partitionkonzept unterstützt die Verwaltung von Partitions unterschiedlicher Größe. Dazu ist es erforderlich, die für den Ablauf eines Basic-Programmes benötigte Partitiongröße im Kennsatz der Basic-Programmdatei abzustellen.
Ist die Angabe #<PART># nicht vorhanden, so setzt der "SAVE"-Processor in den Datei-Kennsatz eine Kennung, daß dieses Programm nur in einer Partition ablaufen kann, die so groß ist wie die größte konfigurierte Partition (Active-File). Die Active-File-Größe, die zum Zeitpunkt des "SAVE" konfiguriert ist, kann unterschiedlich zu der Active-File-Größe sein, die zum Zeitpunkt des Programmablaufes konfiguriert ist. Letztere Größe muß jedoch ausreichen, um das Programm ausführen zu können.
Existiert im Dateikennsatz bereits eine Angabe zur Partition-Größe aufgrund eines vorangegangenen "SAVE" mit der Option #<PART>#, wird diese Angabe übernommen, wenn sie nicht anders im "SAVE" angegeben wird.
Wird eine kleinere Partitiongröße als erforderlich angegeben, wird beim späteren Laden des Programmes TRAP #0 oder TRAP #30 am jeweiligen Arbeitsplatz ausgegeben.

Anhang - Schnittstellen

<<PP>> = Schutzstufen - Zuordnung

Achtung:

Die beiden inneren, spitzen Klammern sind unbedingt anzugeben.

<PP>
- 1. P: Benutzer derselben Privilegebene
- 2. P: Benutzer niedriger Privilegebene

Eine Schutzmöglichkeit gegen Benutzer höherer Privilegebenen besteht nicht.

Folgende Möglichkeiten des Schutzes bestehen:

P = 0 = kein Schutz
- 1 = Kopierschutz. Andere Benutzer werden gehindert, das Programm aufzulisten oder es unter anderem Namen sicherzustellen.
- 2 = Schreibschutz. Andere Benutzer werden gehindert, die Basic-Programmdatei zu löschen oder Attribute zu ändern.
- 3 = Kopier- und Schreibschutz
- 4 = Leseschutz. Andere Benutzer werden gehindert, das Programm zu benutzen.
- 5 = Lese- und Kopierschutz
- 6 = Lese- und Schreibschutz
- 7 = Lese-, Schreib- und Kopierschutz.

Wird keine Schutzstufe angegeben, bleibt der aktuelle Schutz wirksam. Beim erstmaligen Sichern eines Programmes und keiner Schutzstufenangabe wird <77> eingesetzt.

<COST> = Der Betrag, der dem Konto eines anderen Benutzers für den Zugriff auf diese Programmdatei angerechnet wird. Form: $DDD.PP

DDD = Betrag vor dem Komma von 0 bis 999
PP = Betrag nach dem Komma von 0 bis 99

Der eingegebene Nachkomma-Betrag wird immer auf einen vollen 10er Wert abgerundet.
Fehlt die Angabe, bleiben die aktuellen Werte gültig, bzw. bei der ersten Sicherung werden die Kosten auf 0 gesetzt.

Anhang - Schnittstellen

<LU> = Die Nummer der Logischen Einheit, auf die das Programm ausgelagert werden soll. Ist LU = 0 kann diese Angabe entfallen.

<DATEINAME> = Der Name, unter dem das Basic-Programm auf der Magnetplatte ausgelagert werden soll. Diese Angabe entfällt, wenn das Programm bereits ausgelagert ist und z.B. nach Programmänderungen unter dem gleichen Namen erneut gesichert werden soll.

! = Muß angegeben werden, wenn eine bereits bestehende Programmdatei ersetzt werden soll, z.B. ein nicht mehr benötigtes Basic-Programm wird durch ein neu erstelltes ersetzt.

← oder ←337← = Trenncode. Dieser Code zeigt dem SAVE-Processor, daß die Steuerung an das im Anschluß codierte Programm bzw. den angegebenen Processor übergeben werden soll. Diese Angabe ist bündig zu codieren.

<PROG/PROC> = Programm bzw. Processor, an den die Steuerung übergeben werden soll.

Beispiel:

Laden eines Programmes aus einer Textdatei mit dem Namen "T.PROG01" und Sichern dieses Programmes mit SAVE unter dem Namen "PROG01"

```
.
    CHAIN "BASIC DUMMY LOAD T.PROG01<-SAVE PROG01<-PROG"
.
```

Anschließend wird das Programm "PROG" gestartet.

Anhang - Schnittstellen

Fehler:

Stellt "SAVE" einen Fehler fest, wird eine entsprechende Fehlermeldung im ersten Wort (1% - Variable) des gemeinsamen Teilnehmerbereiches abgestellt. Das Sichern wird abgebrochen, und die Steuerung geht an das Programm bzw. an den Processor über, der im Anschluß an den folgenden Trenn-Code (←337←) angegeben ist.
Der übergebene Fehlerschlüssel ist eine um 1000 erhöhte Basic-Fehlernummer.
Zusätzlich dazu kann eine der beiden Fehlernummern übergeben werden:

Fehler #	Bedeutung
1500	Active-File leer
1501	Unzulässige Operanden

Wird kein Fehler erkannt, bleibt der gemeinsame Bereich unverändert.

12.2.2 Processoren, die von Anwenderprogrammen parametriert werden können.

- OPINDEX
- CLEANUP
- REMOVE
- FORMAT
- LIBR
- QUERY
- KILL
- MAIL

Die Parameter, die zu übergeben sind, entsprechen im Aufbau der Parametereingabe per Tastatur unter "SCOPE". Sie sind im Bedienerhandbuch in Kapitel 9 für jeden Processor beschrieben.

Nach dem Ablauf übergibt der entsprechende Processor die Steuerung an "SCOPE".

Beispiel:

Aufruf des Processors "LIBR", um alle auf LU 0 befindlichen Basic-Programme auf dem Drucker auszugeben.

```
CHAIN "LIBR a*B($LPT)"
```

Anhang - Schnittstellen

12.2.3 Processoren, die manuell mit Parametern versorgt werden.

- D2DUTIL
- FORMATTER
- INSTALL AND CLEAR
- EDIT
- BUILDXF
- CHANGE
- KILLALL
- COPYALL
- UTILITY
- ACCOUNTLIST

Der Aufbau des jeweiligen Kommandostrings entspricht dem, der beim Aufruf unter "SCOPE" vorgegeben ist.
Nach dem Laden des Processors erfolgt die Bedienung wie sie im Bedienerhandbuch, Kapitel 9, für jeden Processor beschrieben ist.

Beispiel:

Aufruf des Processors "CHANGE" zum Ändern von Parametern in der Datei "DAT01"

```
CHAIN "CHANGE DAT01"
```

Anhang - Fehlermeldungen/Tabellen

13 Fehlermeldungen/Tabellen

13.1 Liste der Basic-Fehler

Fehler #	Bedeutung
1	Syntaktischer Fehler
2	Unzulässige String-Operation
3	Speicherüberlauf (Programm zu groß)
4	Formatfehler
5	Unzulässiges Zeichen
6	Zeilennummer nicht existent
7	Renumber-Abbruch durch Drücken der Taste "ESC", Programmverlust.
8	Mehr als 93 Variablennamen definiert
9	Unzulässiges Wort
10	RUN-Kommando mit Zeilennummer nicht erlaubt
11	Fehlerhafter Klammerausdruck
12	Programm ist gegen Listen/Kopieren geschützt
13	Numerischer Wert > 9.9999999999999E + 62
14	Keine weiteren Konstanten definiert
15	Arithmetischer Überlauf
16	Zu tiefe Unterprogramm-Schachtelung
17	"RETURN" auf Hauptprogramm-Ebene
18	Zu tiefe "FOR/NEXT" Schachtelung
19	"FOR" ohne zugehörige "NEXT" - Anweisung
20	"NEXT" ohne zugehörige "FOR" - Anweisung
21	Ausdruck zu komplex
22	Nicht genug Plattenblöcke für "SWAP-OUT" vorhanden
23	Matrixgröße überschreitet die Größe der ersten Dimensionierung
24	Strings dürfen nur einmal dimensioniert werden
25	String oder Matrix nicht dimensioniert
26	Logische Einheit ist nicht bereit
27	Syntaktischer Fehler in Anwenderfunktion
28	Unzulässiger Wert für Index, Kanalnummer, SIGNAL-Parameter
29	Unzulässiger Funktions-Aufruf
30	Anwenderfunktion nicht definiert
31	Anwenderfunktion zu tief geschachtelt
32	Matrizen haben ungeleiche Dimensionen
33	Operand ist keine Matrix
34	Dimensionen sind nicht verträglich
35	Matrix ist nicht quadratisch
36	CALL-Unterprogramm ist nicht vorhanden
37	Ausdruck als Operand in einer "CALL"-Anweisung

Anhang - Fehlermeldungen/Tabellen

Fehler #	Bedeutung
38	Fehler von aufgerufenem "CALL" -Unterprogramm erkannt
39	Formatierte Ausgabe überschreitet Puffergröße
40	Kanal bereits belegt
41	Unzulässiger Dateiname
42	Datei nicht gefunden
43	Syntaktischer Fehler in "COST/PROTECTION" - Angabe
44	Unzulässige Dateiart
45	Datei ist gegen Lesen geschützt
46	Datei ist gegen Schreiben geschützt
47	Nicht genügend freie Blöcke auf der angegebenen logischen Einheit zur Verfügung
48	Nicht genügend freie Blöcke auf dem Konto
49	Kanal nicht eröffnet
50	Datei ist bereits eröffnet
51	Unzulässige Satznummer
52	Satz nicht vorhanden
53	Unzulässige Feldnummer
54	Unverträglicher Feldtyp
55	Direktausführung der Anweisung nicht möglich
56	Kein Programm zur Ausgabe vorhanden
57	Stringvariable bereits definiert
58	Fehler in Aufbereitungs-Maske
59	Processor "RUNMAT" nicht im System
60	Zu viele Werte eingegeben
61	Matrizen haben unterschiedliche Element-Formate
62	Signalpuffer voll/Teilnehmernummer nicht vorhanden
63	Kommando ist im Modus "LOAD" unzulässig
64	Fehlende Zeilennummer im Modus "LOAD"
65	Dateiname bereits von einer Datei anderen Typs belegt
66	Dateiname belegt (Datei wird angelegt/ersetzt)
67	Dateiname belegt und kein "!" angegeben
68	Dateiname unter einem anderen Konto belegt
69	Datei ist ein Processor oder Driver
70	Lesefehler
71	Datei ist kein Peripheriegerät
72	Kommando-String in "CHAIN" zu lang

Anhang - Fehlermeldungen/Tabellen

Fehler #	Bedeutung
73	frei
74	frei
75	frei
76	frei
77	frei
78	frei
79	frei
80	frei
81	frei
82	frei
83	frei
84	frei
85	frei
86	frei
87	frei
88	frei
89	frei
90	frei
91	frei
92	frei
93	frei
94	frei
95	frei
96	frei
97	"$DEC" nicht im System
98	Eingabe ist nicht numerisch
99	Taste "ESC" oder "CTL", "C" gedrückt

Anhang - Fehlermeldungen/Tabellen

13.2 Fehlermeldungen des BA

Die Fehlermeldungen des BA (siehe auch Pkt.: 7.2.4) haben folgende Bedeutung:

Fehler #	Bedeutung
01	Codefehler
11	SAS-Zeitfehler
12	SAS-Parityfehler
21	Netzausfall Tastatur
22	Parityfehler Tastatur
23	Netzausfall und Parityfehler Tastatur
41	Parityfehler Datenübertragung
42	Overrun-Error
43	Parityfehler Datenübertragung und Overrun-Error
44	Framing-Error
45	Parityfehler Datenübertragung und Framing-Error
46	Overrun-Error und Framing-Error
47	Parityfehler Datenübertragung, Overrun-Error und Framing-Error
48	Empfangspuffer ist voll

Anmerkung: Der einzige Fehler der durch das Anwenderprogramm verursacht werden kann, ist: ERROR: 01. Dieser Fehler besagt, daß vom BA ein "Lead-in-Code" ohne darauffolgenden Funktionscode empfangen wurde.

13.3 Umrechnungstabelle: oktal ⟷ dezimal

	0	1	2	3	4	5	6	7
0.0	0	1	2	3	4	5	6	7
0.1	8	9	10	11	12	13	14	15
0.2	16	17	18	19	20	21	22	23
0.3	24	25	26	27	28	29	30	31
0.4	32	33	34	35	36	37	38	39
0.5	40	41	42	43	44	45	46	47
0.6	48	49	50	51	52	53	54	55
0.7	56	57	58	59	60	61	62	63
1.0	64	65	66	67	68	69	70	71
1.1	72	73	74	75	76	77	78	79
1.2	80	81	82	83	84	85	86	87
1.3	88	89	90	91	92	93	94	95
1.4	96	97	98	99	100	101	102	103
1.5	104	105	106	107	108	109	110	111
1.6	112	113	114	115	116	117	118	119
1.7	120	121	122	123	124	125	126	127
2.0	128	129	130	131	132	133	134	135
2.1	136	137	138	139	140	141	142	143
2.2	144	145	146	147	148	149	150	151
2.3	152	153	154	155	156	157	158	159
2.4	160	161	162	163	164	165	166	167
2.5	168	169	170	171	172	173	174	175
2.6	176	177	178	179	180	181	182	183
2.7	184	185	186	187	188	189	190	191
3.0	192	193	194	195	196	197	198	199
3.1	200	201	202	203	204	205	206	207
3.2	208	209	210	211	212	213	214	215
3.3	216	217	218	219	220	221	222	223
3.4	224	225	226	227	228	229	230	231
3.5	232	233	234	235	236	237	238	239
3.6	240	241	242	243	244	245	246	247
3.7	248	249	250	251	252	253	254	255

Anhang - Fehlermeldungen/Tabellen

13.4 Umrechnungstabelle: hexadezimal ⟷ dezimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2.	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3.	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4.	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5.	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6.	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7.	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8.	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9.	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A.	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B.	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C.	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D.	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E.	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F.	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Anhang - Fehlermeldungen/Tabellen

13.5 ASCII-Code-Tabelle

Verwendet wird ein 7 Bit ASCII-Code. Das 8. Bit eines Bytes ist intern immer auf 1 gesetzt.

Zeichen	Oktal	Hexadezimal	Bit-Kombination	Taste
NUL	0.0.0	0.0	0 000 000	CS,@
SOH	0.0.1	0.1	0 000 001	CTL,A
STX	0.0.2	0.2	0 000 010	CTL,B
ETX	0.0.3	0.3	0 000 011	CTL,C
EOT	0.0.4	0.4	0 000 100	CTL,D
ENQ	0.0.5	0.5	0 000 101	CTL,E
ACK	0.0.6	0.6	0 000 110	CTL,F
BEL	0.0.7	0.7	0 000 111	CTL,G
BS	0.1.0	0.8	0 001 000	CTL,H
HT	0.1.1	0.9	0 001 001	CTL,I
LF	0.1.2	0.A	0 001 010	CTL,J
VT	0.1.3	0.B	0 001 011	CTL,K
FF	0.1.4	0.C	0 001 100	CTL,L
CR	0.1.5	0.D	0 001 101	CTL,M
SO	0.1.6	0.E	0 001 110	CTL,N
SI	0.1.7	0.F	0 001 111	CTL,O
DLE	0.2.0	1.0	0 010 000	CTL,P
DC1	0.2.1	1.1	0 010 001	CTL,Q
DC2	0.2.2	1.2	0 010 010	CTL,R
DC3	0.2.3	1.3	0 010 011	CTL,S
DC4	0.2.4	1.4	0 010 100	STL,T
NAK	0.2.5	1.5	0 010 101	CTL,U
SYN	0.2.6	1.6	0 010 110	CTL,V
ETB	0.2.7	1.7	0 010 111	CTL,W
CAN	0.3.0	1.8	0 011 000	CTL,X
EM	0.3.1	1.9	0 011 001	CTL,Y
SUB	0.3.2	1.A	0 011 010	CTL,Z
ESC	0.3.3	1.B	0 011 011	CS,K
FS	0.3.4	1.C	0 011 100	CS,L
GS	0.3.5	1.D	0 011 101	CS,M
RS	0.3.6	1.E	0 011 110	CS,N
US	0.3.7	1.F	0 011 111	CS,O

Anhang - Fehlermeldungen/Tabellen

Anmerkungen:

Ist zum Erzeugen eines Zeichens an der Tastatur eine Tastenkombination erforderlich, so wird dies durch eine Mehrfachangabe in der Spalte "Taste" gekennzeichnet. Die einzelnen Tastenangaben werden durch "," voneinander getrennt.

Im einzelnen bedeuten:

- CTL = Control-Taste festhalten, und zusätzliche Taste betätigen
- CS = Control-Taste und Shift-Taste festhalen und zusätzliche Taste betätigen.

Ist das gewünschte Zeichen in der oberen Reihe der angegebenen Taste graviert, muß die Shifttaste vor Zeicheneingabe gedrückt und festgehalten werden. Sollen Kleinbuchstaben eingegeben werden, muß ebenfalls die Shift-Taste gedrückt, festgehalten und dann der entsprechende Großbuchstabe eingegeben werden.

Anhang - Fehlermeldungen/Tabellen

Zeichen	Oktal	Hexadezimal	Bit-Kombination	Taste
SP	0.4.0	2.0	0 100 000	Space
!	0.4.1	2.1	0 100 001	!
"	0.4.2	2.2	0 100 010	"
#	0.4.3	2.3	0 100 011	#
$	0.4.4	2.4	0 100 100	$
%	0.4.5	2.5	0 100 101	%
&	0.4.6	2.6	0 100 110	&
´	0.4.7	2.7	0 100 111	´
(	0.5.0	2.8	0 101 000	(
)	0.5.1	2.9	0 101 001	)
*	0.5.2	2.A	0 101 010	*
+	0.5.3	2.B	0 101 011	+
,	0.5.4	2.C	0 101 100	,
-	0.5.5	2.D	0 101 101	-
.	0.5.6	2.E	0 101 110	.
/	0.5.7	2.F	0 101 111	/
0	0.6.0	3.0	0 110 000	0
1	0.6.1	3.1	0 110 001	1
2	0.6.2	3.2	0 110 010	2
3	0.6.3	3.3	0 110 011	3
4	0.6.2	3.4	0 110 100	4
5	0.6.5	3.5	0 110 101	5
6	0.6.6	3.6	0 110 110	6
7	0.6.7	3.7	0 110 111	7
8	0.7.0	3.8	0 111 000	8
9	0.7.1	3.9	0 111 001	9
:	0.7.2	3.A	0 111 010	:
;	0.7.3	3.B	0 111 011	;
<	0.7.4	3.C	0 111 100	<
=	0.7.5	3.D	0 111 101	=
>	0.7.6	3.E	0 111 110	>
?	0.7.7	3.F	0 111 111	?
@	1.0.0	4.0	1 000 000	@
A	1.0.1	4.1	1 000 001	A
B	1.0.2	4.2	1 000 010	B
C	1.0.3	4.3	1 000 011	C
D	1.0.4	4.4	1 000 100	D
E	1.0.5	4.5	1 000 101	E
F	1.0.6	4.6	1 000 110	F
G	1.0.7	4.7	1 000 111	G
H	1.1.0	4.8	1 001 000	H
I	1.1.1	4.9	1 001 001	I
J	1.1.2	4.A	1 001 010	J
K	1.1.3	4.B	1 001 011	K
L	1.1.4	4.C	1 001 100	L
M	1.1.5	4.D	1 001 101	M
N	1.1.6	4.E	1 001 110	N
O	1.1.7	4.F	1 001 111	O

Anhang - Fehlermeldungen/Tabellen

Zeichen	Oktal	Hexadezimal	Bit-Kombination	Taste
P	1.2.0	5.0	1 010 000	P
Q	1.2.1	5.1	1 010 001	Q
R	1.2.2	5.2	1 010 010	R
S	1.2.3	5.3	1 010 011	S
T	1.2.4	5.4	1 010 100	T
U	1.2.5	5.5	1 010 101	U
V	1.2.6	5.6	1 010 110	V
W	1.2.7	5.7	1 010 111	W
X	1.3.0	5.8	1 011 000	X
Y	1.3.1	5.9	1 011 001	Y
Z	1.3.2	5.A	1 011 010	Z
Ä	1.3.3	5.B	1 011 011	Ä
Ö	1.3.4	5.C	1 011 100	Ö
Ü	1.3.5	5.D	1 011 101	Ü
↑	1.3.6	5.E	1 011 110	↑
←	1.3.7	5.F	1 100 000	←
ß	1.4.0	6.0	1 100 000	ß
a	1.4.1	6.1	1 100 001	a
b	1.4.2	6.2	1 100 010	b
c	1.4.3	6.3	1 100 011	c
d	1.4.4	6.4	1 100 100	d
e	1.4.5	6.5	1 100 101	e
f	1.4.6	6.6	1 100 110	f
g	1.4.7	6.7	1 100 111	g
h	1.5.0	6.8	1 101 000	h
i	1.5.1	6.9	1 101 001	i
j	1.5.2	6.A	1 101 010	j
k	1.5.3	6.B	1 101 011	k
l	1.5.4	6.C	1 101 100	l
m	1.5.5	6.D	1 101 101	m
n	1.5.6	6.E	1 101 110	n
o	1.5.7	6.F	1 101 111	o
p	1.6.0	7.0	1 110 000	p
q	1.6.1	7.1	1 110 001	q
r	1.6.2	7.2	1 110 010	r
s	1.6.3	7.3	1 110 011	s
t	1.6.4	7.4	1 110 100	t
u	1.6.5	7.5	1 110 101	u
v	1.6.6	7.6	1 110 110	v
w	1.6.7	7.7	1 110 111	w
x	1.7.0	7.8	1 111 000	x
y	1.7.1	7.9	1 111 001	y
z	1.7.2	7.A	1 111 010	z
ä	1.7.3	7.B	1 111 011	ä
ö	1.7.4	7.C	1 111 100	ö
ü	1.7.5	7.D	1 111 101	ü
Lead in	1.7.6	7.E	1 111 110	CS, .
DEL	1.7.7	7.F	1 111 111	CS, /

Stichwortverzeichnis

14 Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis