14 Lochstreifenleser/Lochstreifenstanzer

Lochstreifen können zum Datenaustausch mit anderen Systemen oder zur Datensicherung eingesetzt werden.
Das System erlaubt den Anschluß je eines Lochstreifenlesers und eines Lochstreifenstanzers zur Bearbeitung sowohl von 5-Kanal- als auch von 8-Kanal-Lochstreifen.

14.1 Eingabe über den Lochstreifenleser

Für den Lochstreifenleser bildet ein logisches IOCS (Input-Output-Control-System) die Schnittstelle zur Sprache Business-BASIC. Das IOCS übernimmt die Verwaltung der Datenbestände. Alle über den Lochstreifenleser eingelesenen Daten sind als Dateien organisiert. Die Bearbeitung einer Lochstreifendatei erfolgt über einen Datenkanal.

Bevor Daten über den Lochstreifenleser eingelesen werden können, muß der Lochstreifenleser auf einem beliebigen Kanal eröffnet werden:

```
OPEN    # <NAusdr> [;<String1>] ,<String2>
      [[,# <NAusdr> [;<String1>]] ,<String2>]n
```

<NAusdr> : Kanalnummer, auf der der Lochstreifenleser eröffnet werden soll.

<String1> : Soll mit Codewandlung gearbeitet werden, ist hier die Code-Tabelle zu übergeben. Zweckmäßigerweise wird die Code-Tabelle vorher in einer String-Variablen abgestellt, die hier angegeben wird.

Zusätzlich zu den Informationen für Codewandlung enthält diese Tabelle Steuerinformationen für die Verarbeitung.
Aufbau von Code-Tabellen siehe Kap. "Code-Tabellen für Lochstreifenleser/-Stanzer".

<String2> : Als Geräte-Dateiname für den Lochstreifenleser ist

"$PTRA"

anzugeben.

Lochstreifenleser/Lochstreifenstanzer

Lesen von Daten

Die Eingabe von Daten über den Lochstreifenleser erfolgt sequentiell mit der Anweisung

```
READ #
```

Achtung!

Es dürfen nur Strings als Ziel-Variable angegeben werden. Wird eine numerische Variable als Ziel-Variable angegeben, wird

```
ERROR # 54  (Unverträglicher Feldtyp)
```

gemeldet.

Grundsätzlich sollte bei der Eingabe über den Lochstreifenleser nur in eine Zielvariable angegeben werden.

Die Übertragung wird beendet bei

- Auftreten eines Grenzzeichens,
- Erreichen der dimensionierten bzw. durch Indizes angegebenen Länge der String-Variablen.

Angaben von Satznummer und Satz-Displacement in der READ #-Anweisung werden ignoriert, ebenso liefert die CHF-Funktion keine definierte Aussage.

Schließen des Lochstreifenlesers

Das Schließen des Datenkanals, auf dem der Lochstreifenleser eröffnet wurde, erfolgt mit der Anweisung

```
CLOSE #
```

Achtung:

Durch die Grenzzeichenverarbeitung von "RUN" kann das letzte in den String gelesene Zeichen verloren gehen, wenn nicht mit einen Teilstring gearbeitet wird.

```
Statt READ#; A$
also READ#; A$(1,LENA$-1)
```

Lochstreifenleser/Lochstreifenstanzer

14.2 Ausgabe über den Lochstreifenstanzer

Für den Lochstreifenstanzer bildet ebenso wie für den Lochstreifenleser ein logisches IOCS (Input-Output-Control-System) die Schnittstelle zur Sprache Business-BASIC. Das IOCS übernimmt die Verwaltung der zu übertragenden Daten. Die über den Lochstreifenstanzer auszugebenden Daten sind als Dateien organisiert. Die Bearbeitung erfolgt über einen Datenkanal.

- Eröffnen des Lochstreifenstanzers

  Bevor Daten über den Lochstreifenstanzer ausgegeben werden können, muß er auf einem beliebigen freien Datenkanal eröffnet werden:

  OPEN # ⟨NAusdr⟩ [;⟨String1⟩] ,⟨String2⟩
  [[,# ⟨NAusdr⟩ [;⟨String1⟩]] ,⟨String2⟩]^{n}

  ⟨NAusdr⟩ : Kanalnummer, auf der der Lochstreifenstanzer eröffnet werden soll.

  ⟨String1⟩ : Soll mit Codewandlung gearbeitet werden, ist hier die Code-Tabelle zu übergeben. Zweckmäßigerweise wird die Code-Tabelle vorher in einer String-Variablen abgestellt, die hier angegeben wird.

  Zusätzlich zu den Informationen für Codewandlung enthält diese Tabelle Steuerinformationen für die Verarbeitung.
  Aufbau von Codetabellen siehe Kap. "Code-Tabellen für Lochstreifenleser/-Stanzer".

  ⟨String2⟩ : Als Geräte-Dateiname für den Lochstreifenstanzer ist

  "$PTPA"

  anzugeben.

- Ausgeben von Daten

  Die Ausgabe von Daten über den Lochstreifenstanzer erfolgt sequentiell mit den Anweisungen

  - PRINT # und
  - WRITE #

Dabei ist darauf zu achten, daß in der WRITE #-Anweisung nur String-Variablen und String-Literale angegeben werden dürfen.
Wird in einer WRITE #-Anweisung eine numerische Variable oder ein numerischer Ausdruck ausgegeben, führt dies zu

ERROR # 54 (Unverträglicher Feldtyp)

Die Anweisung PRINT # wandelt alle auszugebenden Daten automatisch in alphanumerische Daten um.

Grundsätzlich sollte bei der Ausgabe über den Lochstreifenstanzer nur eine String-Variable angegeben werden.

Die Angabe von Satznummer und Satz-Displacement bleibt ohne Wirkung, die Funktion CHF liefert keine definierten Werte.

Die Übertragungslänge wird bestimmt durch

- das Auftreten eines Grenzzeichens im Quellstring,
- Erreichen von String-Ende in einer String-Variablen oder einem String-Literal.

Die Grenzzeichenbehandlung kann allerdings durch Vorgabe von Steuerinformationen in der Code-Tabelle beeinflußt werden (siehe Kap. "Code-Tabellen für Lochstreifenleser/-Stanzer").

- Schließen des Lochstreifenstanzers

Das Schließen des Datenkanals, auf dem der Lochstreifenstanzer eröffnet wurde, erfolgt mit der Anweisung

CLOSE #

Das bewirkt in Abhängigkeit von der Codetabelle das Stanzen von Leerlochungen. Standardmäßig (ohne Codetabelle oder Bit 1 von Byte 1 der Codetabelle = 0) wird ein Nachlauf von 256 Leerzeichen (nur Führungslöcher) gestanzt.

14.3 Code-Tabellen für Lochstreifenleser/-Stanzer

Soll mit Codeumwandlung gearbeitet werden, ist in der OPEN #-Anweisung eine String-Variable als ⟨String1⟩ anzugeben. In dieser Variablen ist vor der Durchführung der OPEN #-Anweisung die Umwandlungstabelle abzustellen.

Lochstreifentabellen haben immer den folgenden Aufbau:

- Byte 1 bis 4, Steuerinformationen
- Byte 5 bis 260, Tabelle

Aufbau Eingabetabelle (Lochstreifenleser)

Byte	Bedeutung
1	= 0, 8-Kanal Lochstreifen = 1, 5-Kanal Lochstreifen
2	= 0, Vorlauf überlesen = 1, Vorlauf nicht überlesen
3	Umschaltzeichen für Buchstaben (nur bei 5-Kanal Lochstreifen)
4	Umschaltzeichen für Ziffern (nur bei 5-Kanal Lochstreifen)
5-260	Tabelle

Ist keine Tabelle in der OPEN #-Anweisung angegeben, werden die Bytes 1 und 2 als 0 angenommen und der Lochstreifencode direkt übernommen.

Bei 5-Kanal-Lochstreifen (Fernschreibcode) müssen in Byte 3 und 4 die Umschaltzeichen eingetragen sein. Die Tabelle ist in diesem Fall in Gruppen zu je 64 Byte eingeteilt:

- Byte 5 - 68 = Buchstabengruppe
- Byte 69 - 132 = Zifferngruppe
- Byte 133 - 260 = frei

Wird eines der Umschaltzeichen gelesen, wird vom Kanalprogramm auf die entsprechende Gruppe umgeschaltet. Die Umschaltzeichen werden der Variablen nicht übergeben!

Lochstreifenleser/Lochstreifenstanzer

Aufbau Ausgabetabelle (Lochstreifenstanzer)

Byte	Bedeutung
1	= 0, 8-Kanal Lochstreifen = 1, 5-Kanal Lochstreifen
2	= 0, Endekennzeichen stanzen = 1, Endekennzeichen nicht stanzen jeweils mit Vor- und Nachlauf = 2, Endekennzeichen stanzen = 3, Endekennzeichen nicht stanzen jeweils ohne Vor- und Nachlauf
3	Umschaltzeichen für Buchstaben (nur bei 5-Kanal Lochstreifen)
4	Umschaltzeichen für Ziffern (nur bei 5-Kanal Lochstreifen)
5-260	Tabelle

Ist keine Tabelle in der OPEN #-Anweisung angegeben, werden die Bytes 1 und 2 als 0 angenommen und der ASCII-Code direkt gestanzt.

Bei 5-Kanal-Lochstreifen muß auf jedem Tabellenplatz (Byte 5-260) in Bit 7 und 8 eine Gruppenkennung angegeben sein.

Aufbau eines Tabellenplatzes bei 5-Kanal:

Bit	Bedeutung
1-5	Zu stanzender Code
6	frei
7-8	= 00, nicht erlaubt = 01, Zeichen der Buchstabengruppe = 10, Zeichen der Zifferngruppe = 11, Zeichen kommt in beiden Gruppen vor

Beim Wechsel von einer zur anderen Gruppe wird vom Kanalprogramm das entsprechende Umschaltzeichen (Byte 3 und 4) ausgegeben.

Lochstreifenleser/Lochstreifenstanzer

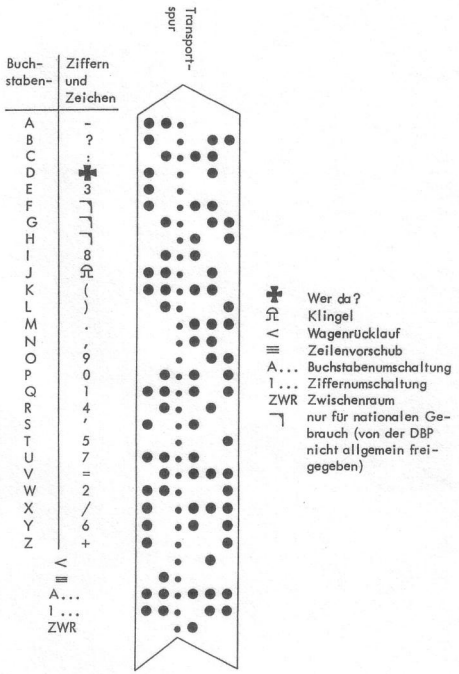
Standardtabellen

Als Standardtabellen werden zur Verfügung gestellt:

TT5.PUNCHTABLE: Codetabelle zur Erstellung von 5-Kanal-Lochstreifen in Fernschreibcode.
Standardeinträge Steuerinformation:
- 5-Kanal-Kennung in Byte 1
- Kein Endekennzeichen stanzen in Byte 2
- Vor- und Nachlauf stanzen in Byte 2

TT5.READTABLE: Codetabelle zum Einlesen von 5-Kanal-Lochstreifen im Fernschreibcode.
Standardeinträge Steuerinformation:
- 5-Kanal-Kennung in Byte 1
- Vorlauf überlesen in Byte 2

Fernschreibcode:



15 Das Magnetband

Das Magnetband kann zum Datenaustausch mit anderen Systemen und zur Datensicherung eingesetzt werden.

Bei Anschluß von SMT-Laufwerken (nur über den neuen MB-Controller 1895.01 möglich) sind 1600 bpi bzw. 3200 bpi möglich, nicht aber 800 bpi. 3200 bpi wird vor allem zur Datensicherung und -rekonstruktion unter TAMOS genutzt.

Ab REL 5.1 gibt es, zusätzlich zu der CALL70-Schnittstelle, eine neue Magnetband-Schnittstelle (Peripheriedriver-Schnittstelle) mit den Anweisungen OPEN #, ENV #, READ #, WRITE # und CLOSE #. Zulässige Drivernamen sind $MT0, $MT1, $MT2 und $MT3. Neu programmierte Anwendersoftware sollte diese neue Peripheriedriver-Schnittstelle benutzen. Die Bitdichte (1600/3200 bpi) wird bei der OPEN-Anweisung vorgegeben.
Programme, die auf einem Betriebssystemstand vor Release 5.1 laufen sollen, müssen als Schnittstelle zur Magnetbandstation weiterhin auf den CALL70 zurückgreifen.

15.1 Organisation der Daten auf einem Magnetband

Alle auf einem Magnetband gespeicherten Daten werden im Rahmen von Dateien organisiert. Die einzig mögliche Organisationsform ist sequentiell. Die Verwaltung der Dateien muß vollständig vom Anwenderprogramm durchgeführt werden.

Das Band kann zwischen den Reflektormarken für Bandanfang und Bandende in nahezu beliebiger Weise benutzt werden. Ein Update von Blöcken innerhalb des logischen Bandanfangs und -endes ist jedoch nicht erlaubt. Physikalisch gesehen wird in diesem Bereich lediglich zwischen Datenblöcken, die beliebig lang sein können, und Blocklücken (Gap) unterschieden. Das Gap dient zur Trennung zweier Blöcke und zur Überbrückung kleiner defekter Bandstellen. Es hat für die Verarbeitung der Information keinerlei Bedeutung.

Ein Block ist eine zusammenhängende Informationseinheit beliebiger Länge. Umfang und Inhalt dieser Information werden durch das bearbeitende Programm festgelegt.

Das Magnetband

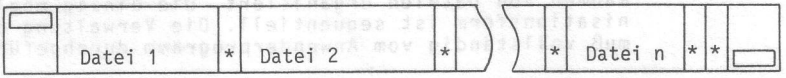
Sollen Magnetbänder mit anderen Systemen ausgetauscht werden, müssen bestimmte Vorschriften über den logischen Datenträgeraufbau eingehalten werden.
Logisch gliedert sich die Datenkapazität des Magnetbandes in eine oder mehrere Dateien. Die Dateien wiederum bestehen aus beliebig vielen Datenblöcken.

Zur logischen Gliederung des Bandes werden sogenannte Bandmarken benutzt. Dies sind 1-Zeichen-Blöcke mit einem beliebigen Code bei 800 bpi, ab 1600 bpi wird der Bandmarkenblock durch die Hardware verwaltet. Das Schreiben von 1-Zeichen-Blöcken durch das Anwenderprogramm ist deshalb zu vermeiden.

Jedes Band endet logisch mit einer Doppelbandmarke.
Zwischen Bandanfang und dieser Doppelbandmarke liegen die Dateien, die jeweils durch eine Bandmarke getrennt sind.
Bei Dateien mit Kennsätzen ist der Kennsatz-Bereich der Datei vom Datenbereich ebenfalls durch eine Bandmarke abgetrennt.

Bänder ohne Kennsätze

Bei einem Band ohne Kennsätze ergibt sich folgende Bandstruktur:



* Bandmarke

Das Magnetband

Bänder mit Kennsätzen

Ein Magnetband mit Kennsätzen muß mindestens folgende Kennsätze enthalten:

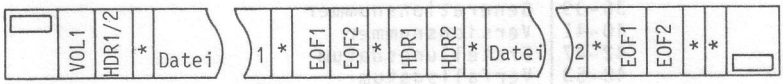
- VOL1 = Bandanfangskennsatz
- HDR1 = Dateianfangskennsatz 1
- EOF1 = Dateiendekennsatz 1 und ggf.
- EOV1 = Bandendekennsatz 1

In höheren Kennsatzstufen können auch die Kennsätze HDR2, EOF2/EOV2 obligatorisch sein.
Falls Kennsätze HDR2-9, EOF2-9 oder EOV2-9 benutzt werden, müssen diese direkt hinter den entsprechenden Kennsätzen 1 stehen. Wahlweise können weitere Benutzer-Kennsätze folgen.

- UVL1 bis UVLn = Benutzer-Bandanfangskennsätze (n=2-9) (schließen unmittelbar an VOL1 an)
- UHLa bis UHLa = Benutzer-Dateianfangskennsätze (schließen unmittelbar an HDR1 bzw. 2-9 an)
- UTLa bis UTLa = Benutzer-Bandendekennsätze (schließen unmittelbar an EOF2 bzw. 3-9 oder an EOV2 bzw. 3-9 an)

a = beliebiges Alphazeichen

Es ergibt sich folgende Bandstruktur:



Aufbau der Kennsätze

Jeder Kennsatz ist ein Block von 80 Zeichen. Entspricht der Inhalt der Kennsätze der DIN-Norm 66029, ist Kompatibilität zu anderen Systemen gegeben.
Der Code der Daten innerhalb einer Datei muß nicht mit dem Code der Kennsätze identisch sein.

Das Magnetband

VOL1

Der Kennsatz hat die Aufgabe, das Band zu identifizieren.

Byte	Inhalt	Wert
1-3	Kennsatzname	VOL
4	Kennsatznummer	1
5-10	Bandkennzeichen (Archivnummer)	
11	Zugriffsvermerk	Ø
12-37	Reserviert für spätere Normung	Ø
38-51	Eigentümerkennzeichen	
52-79	Reserviert für spätere Normung	Ø
80	Normvermerk	1/2/3

HDR1 / HDR2

Die Kennsätze HDR1 und HDR2 dienen zur Identifikation der folgenden Datei.

Byte	Inhalt	Wert
1-3	Kennsatzname	HDR
4	Kennsatznummer	1
5-21	Dateiname	
22-27	Dateimengenkennzeichen	
28-31	Dateiabschnittsnummer	
32-35	Dateifolgenummer	
36-39	Generationsnummer	
40-41	Versionsnummer	00
42-47	Erstellungsdatum	ØJJTTT
48-53	Verfallsdatum	ØJJTTT
54	Zugriffsvermerk	Ø
55-60	Blockzähler	000000
61-73	Systemkennzeichen (Rest Ø)	
74-80	Reserviert für spätere Normung	Ø

Byte	Inhalt	Wert
1-3	Kennsatzname	HDR
4	Kennsatznummer	2
5	Satzformat	F/D/S
6-10	Blocklänge	
11-15	Satzlänge	
16-50	Ø	
51-52	Länge eines zusätzlichen Blockfeldes, das am Anfang jedes Datenblocks eingefügt ist.	
53-80	Reserviert für spätere Normung	Ø

Das Magnetband

EOF1 / EOF2

Diese Kennsätze kennzeichnen das Ende Ende einer Datei.

Byte	Inhalt	Wert
1-3	Kennsatzname	EOF
4	Kennsatznummer	1
5-54	entspricht HDR1	
55-60	Blockzähler: Anzahl der Datenblöcke seit der letzten Kennsatzgruppe	
61-80	entspricht HDR1	

Byte	Inhalt	Wert
1-3	Kennsatzname	EOF
4	Kennsatznummer	2
5-80	entspricht HDR2	

EOV1 / EOV2

Diese Kennsätze kennzeichnen das logische Bandende bei Mehrband-Dateien. Sie sind bis auf den Kennsatznamen und die Kennsatznummer völlig mit den Kennsätzen EOF1 und EOF2 identisch.

UVL / UHL / UTL

Wahlweise können die obligatorischen Kennsätze durch Benutzerkennsätze ergänzt werden. Diese Kennsätze sind ebenfalls 80 Zeichen lang, ihr Inhalt unterliegt jedoch ebenso wie der Inhalt von HDR3-9, EOF3-9 und EOV3-9 keiner besonderen Vorschrift und kann von Anwender frei belegt werden. Als einzige Bedingung wird ein zulässiger Kennsatzname verlangt.

Das Magnetband

15.2 Zugriff auf Magnetbanddateien

Eine Magnetbandstation kann nur von einem Teilnehmer zu einer Zeit eröffnet werden.

15.2.1 Eröffnen des Magnetbandgerätes mit der OPEN-Anweisung

Mit dem Aufruf der OPEN-Anweisung wird das Magnetbandgerät dem eröffnenden Arbeitsplatz zugeordnet und für andere Teilnehmer gesperrt.

Syntax: OPEN #[<NAusdr1>];[<NAusdr2>],<SLit>

<NAusdr1> Kanalnummer, auf der die Bandstation eröffnet werden soll.

<NAusdr2>: Kennung für das Einstellen der Bitdichte. Besitzt der Numerische Ausdruck den Wert 4, wird die Bitdichte auf 3200 bpi eingestellt. Ist <NAusdr2> nicht angegeben, wird auf 1600 bpi eingestellt. (Hinweis: die Einstellung erfolgt erst dann, wenn das Band auf BOT steht)

<SLit> Name des Drivers. Zulässige Drivernamen sind $MT0, $MT1, $MT2 und $MT3.

15.2.2 Die READ-Anweisung

Mit dieser Anweisung werden Daten von Band gelesen.

Syntax: READ #⟨NAusdr⟩;⟨SVar⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨SVar⟩ String-Variable zur Aufnahme der empfangenen Daten.

Die Länge der String-Variablen, die durch die Dimensionierung oder Indizierung festgelegt wird, gibt die Anzahl der zu lesenden Zeichen an. Ausnahme: Ist bei einer indizierten Variablen die Stringlänge kleiner, als durch die Dimensionierung vorgegeben, wird ein Zeichen weniger übertragen, als die Indizierung angibt.

Die Abweichung zur tatsächlich gelesenen Anzahl Zeichen kann mit der ENV-Anweisung (Funktion 1) abgefragt werden.

Wichtig:
Die gelesenen Daten stehen erst nach der Statusabfrage zur Verfügung.

15.2.3 Die WRITE-Anweisung

Mit dieser Anweisung werden Daten auf Band geschrieben.

Syntax: WRITE #⟨NAusdr⟩;⟨SVar⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨SVar⟩ String-Variable mit den zu sendenden Daten

Die Länge der String-Variablen, die durch die Dimensionierung oder Indizierung festgelegt wird, gibt die Anzahl der zu schreibenden Zeichen an. Ausnahme: Ist bei einer indizierten Variablen die Stringlänge kleiner als durch die Dimensionierung vorgegeben, so wird ein Zeichen weniger übertragen, als die Indizierung angibt.

Das Magnetband

15.2.4 Die ENV-Anweisung

Die ENV-Anweisung übergibt Angaben, die für den Betrieb einer Datenübertragungsleitung notwendig sind, und sie ermittelt den Status der Leitung nach Ausführung einer OPEN#-, WRITE#- und READ#-Anweisung.

15.2.4.1 Übergabe der Codetabellen

Mit der ENV-Anweisung, Funktionsnummer 0, können Codetabellen an den Driver übergeben werden.

Syntax: ENV #⟨NAusdr⟩;⟨NVar⟩[,⟨SVar1⟩,⟨SVar2⟩]

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨NVar⟩ 0 (Funktionsnummer für die Codeumschaltung)

⟨SVar1⟩ Codetabelle für das Lesen von Band (256 Bytes)

⟨SVar2⟩ Codetabelle für das Schreiben auf Band (256 Bytes)

Falls ⟨SVar1⟩ und ⟨SVar2⟩ nicht angegeben werden, wird auf Lesen/Schreiben ohne Codetabelle umgeschaltet.

15.2.4.2 Die Status-Abfrage

Nach jeder OPEN#-, READ#- und WRITE#-Anweisung muß mit der ENV-Anweisung, Funktion 1, der Status überprüft werden.

Syntax: ENV #⟨NAusdr⟩;⟨NVar1⟩,⟨NVar2⟩,⟨NVar3⟩,⟨NVar4⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨NVar1⟩ 1 (Funktionsnummer für die Statusabfrage)

⟨NVar2⟩ Fehlerschlüssel mit folgender Bedeutung:

0 Operation erfolgreich
1 nicht betriebsbereit
2 Schreibschutz
3 max. Gaplänge überschritten
4 Operation nicht vollständig ausgeführt (vorzeitig FM, LDP, EOT, 2 FM's erkannt; s. ⟨NVar3⟩)
5 nicht behebbarer Bandfehler
6 Controller defekt
7 Time out, Parity-Fehler, Lost Data
8 Netzausfall
9 Band ist mit anderer Bitdichte beschrieben

⟨NVar3⟩ Status der Bandstation

0 kein Status
1 EOT (Bandende)
2 LDP (Bandanfang)
3 RWD (Band spult zurück)
4 FM (Bandmarke gefunden)
5 FM und EOT (Bandmarke und Bandende gefunden)

⟨NVar4⟩)= 2%-Variable. Abweichung zwischen der gewünschten Anzahl Zeichen, Blöcke bzw. Bandmarken und der tatsächlichen Anzahl nach Ausführung der Anweisung. (Abweichung = tatsächlich - gewünscht). Positive Abweichungen werden nur vom PSR-Controller erkannt.

Das Magnetband

15.2.4.3 Rückspulen

Mit der ENV-Anweisung, Funktion 2, wird das Band bis BOT zurückgespult.

Syntax: ENV #⟨NAusdr⟩;⟨NVar1⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨NVar1⟩ Funktionscode: 2

15.2.4.4 Rückspulen und Abschalten

Mit der ENV-Anweisung, Funktion 2, wird das Magnetband zurückgespult und das Bandgerät "Offline" geschaltet.

Syntax: ENV #⟨NAusdr⟩;⟨NVar1⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨NVar1⟩ Funktionscode: 3

15.2.4.5 Bandmarke schreiben

Mit der ENV-Anweisung, Funktion 4, wird eine Bandmarke geschrieben.

Syntax: ENV #⟨NAusdr⟩;⟨NVar1⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨NVar1⟩ Funktionscode: 4

15.2.4.6 Blöcke vorsetzen

Mit der ENV-Anweisung, Funktion 5, kann eine vorgegebene Anzahl Blöcke auf dem Magnetband vorgesetzt werden. Die tatsächliche Anzahl Blöcke, um die vorgesetzt wurde, muß über die Statusabfrage bestimmt werden.

Syntax: ENV #⟨NAusdr⟩;⟨NVar1⟩,⟨NVar2⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde

⟨NVar1⟩ Funktionscode: 5

⟨NVar2⟩ Anzahl Blöcke, um die vorgesetzt werden soll

15.2.4.7 Blöcke zurücksetzen

Mit der ENV-Anweisung, Funktion 6, kann um eine vorgegebene Anzahl Blöcke zurückgesetzt werden. Die tatsächliche Anzahl Blöcke, um die zurückgesetzt wurde, wird bei der Statusabfrage angegeben.

Syntax: ENV #⟨NAusdr⟩;⟨NVar1⟩,⟨NVar2⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet wurde.

⟨NVar1⟩ Funktionscode: 6

⟨NVar2⟩ Anzahl Blöcke, um die zurückgesetzt werden soll.

15.2.5 Die CLOSE-Anweisung

Diese Anweisung schließt den mit OPEN eröffneten Kanal.

Syntax: CLOSE #⟨NAusdr⟩

⟨NAusdr⟩ Kanalnummer, auf der die Bandstation eröffnet worden ist.

15.2.6 Eröffnen des Magnetbandgeräts mit dem CALL 70

Für die Programmierung des Magnetbandes auf älteren Betriebssystemständen als Release 5.1 steht weiterhin die Anweisung

```
CALL 70
```

zur Verfügung. CALL 70 bildet die Schnittstelle zischen dem BASIC-Interpreter RUN und dem Kanalprogramm $MTX.

Die Funktionen zum Eröffnen des Magnetbandgeräts beinhalten gegebenenfalls ein Zurückspulen auf BOT und legen fest, ob mit oder ohne Codewandlung gearbeitet wird.

Syntax:

```
CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩,
                                    [,⟨SVar1⟩, ⟨SVar2⟩]
```

⟨NAusdr⟩ : 70

⟨NumPar1⟩ : 0 bis 3 (Nummer der Bandstation).

⟨NumPar2⟩ : Funktionsvariable:
11: Eröffnen des Bandgerätes
12: wie 11 und Zurückspulen auf BOT

⟨NVar3⟩ : Statusvariable.

⟨SVar1⟩ : Eingabe-Codetabelle (256 Byte).

⟨SVar2⟩ : Ausgabe-Codetabelle (256 Byte).

Soll mit Codewandlung gearbeitet werden, müssen ⟨SVar1⟩ und ⟨SVar2⟩ codiert und vor Eröffnung geladen sein.

Das Magnetband

15.2.7 Verarbeitung

15.2.7.1 Verwaltung des Datenträgers

Mit diesen Funktionen kann das Band über schnellen Rücklauf auf BOT (Bandanfangsmarke) zurückgespult, eine Bandmarke oder ein Gap geschrieben werden.
Das Schreiben eines Gaps wird notwendig bei permanenten Bandfehlern. Ab Release 5.1 wird die Fehlerbehandlung allerdings vom Kanalprogramm bzw. vom PSR-Controller durchgeführt; die Anweisung "Gap schreiben" hat dann keine Wirkung und wird vom Controller nicht mehr angeboten.

Syntax:

```
CALL <NAusdr>, <NumPar1>, <NumPar2>, <NVar3>
```

<NAusdr> : 70

<NumPar1> : 0 bis 3 (Bandstation)

<NumPar2> : Funktionsvariable:
1: Band auf BOT zurückspulen
6: Bandmarke schreiben
7: Gap schreiben (2,5 Zoll Bandlücke)
10: Bandmarke schreiben und Band auf BOT spulen

<NVar3> : Statusvariable.

15.2.7.2 Band vor- oder zurücksetzen

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩, ⟨NumPar4⟩, ⟨NVar5⟩

⟨NAusdr⟩ : 70

⟨NumPar1⟩ : 0 bis 3 (Bandstation)

⟨NumPar2⟩ : Funktionsvariable:
3: Band um ⟨NumPar4⟩ Blöcke vorsetzen
4: Band um ⟨NumPar4⟩ Blöcke zurücksetzen
8: Band um ⟨NumPar4⟩ Bandmarken vorsetzen
9: Band um ⟨NumPar4⟩ Bandmarken zurücksetzen

⟨NVar3⟩ : Statusvariable.

⟨NVar5⟩ : Tatsächlich verarbeitete Anzahl Blöcke oder Bandmarken.

Das Magnetband

15.2.7.3 Block lesen/schreiben

Lesen oder Schreiben eines Datenblocks, Blocklänge: $\geq$ 2 und $\leq$ Magnetbandpuffer des Systems. Die Anzahl tatsächlich verarbeiteter Zeichen, die in ⟨NVar5⟩ abgestellt wird, sollte in jedem Fall überprüft werden.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩, ⟨NumPar4⟩, ⟨NVar5⟩, ⟨SVar⟩

⟨NAusdr⟩ : 70

⟨NumPar1⟩ : 0 bis 3 (Bandstation)

⟨NumPar2⟩ : Funktionsvariable:
0: Block lesen. Die Übertragung wird beendet bei Erreichen der dimensionierten Länge von ⟨SVar⟩ oder der in ⟨NumPar4⟩ vorgegebenen Anzahl Zeichen.
5: Block schreiben. Übertragungsende wie Block lesen.

⟨NVar3⟩ : Statusvariable.

⟨NumPar4⟩ : Anzahl zu verarbeitender Zeichen.

⟨NVar5⟩ : Anzahl tatsächlich verabeiteter Zeichen.

⟨SVar⟩ : Stringvariable zur Aufnahme des gelesenen oder zu schreibenden Blocks.

15.2.8 Schließen des Bandgerätes

Die Funktionen zum Schließen des Bandgerätes erlauben zusätzlich ein "off-line"-Schalten der Magnetbandstation.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩

⟨NAusdr⟩ : 70

⟨NumPar1⟩ : 0 bis 3 (Bandstation)

⟨NumPar2⟩ : Funktionsvariable:
2: Bandgerät "off-line" schalten
13: Bandgerät schließen
14: Bandgerät schließen und "off-line" schalten

⟨NVar3⟩ : Statusvariable.

15.2.9 Ermitteln der Größe des Magnetbandpuffers

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩, ⟨NVar4⟩

⟨NAusdr⟩ : 70

⟨NumPar1⟩ : 0 bis 3 (Bandstation)

⟨NumPar2⟩ : 15 (Funktionsvariable)

⟨NVar3⟩ : Statusvariable.

⟨NVar4⟩ : Nach Ausführung Größe des Magnetbandpuffers.

Da die neuen Magnetbanddriver (ab Release 5.1) direkt auf die Anwenderpartition zugreifen, kann die Konfigurierung eines Magnetbandpuffers entfallen. In diesem Fall wird die Funktion 15 (Ermitteln der Größe des Magnetbandpuffers) nicht mehr benötigt.

Das Magnetband

15.3 Fehlerbehandlung

15.3.1 Fehlerbehandlung der Magnetband-Schnittstelle

Bei fehlerhaftem Aufruf (falsche Konfiguration der Magnetbandsoftware, falsche Funktionsnummer bei ENV-Aufrufen, fehlende Statusabfrage nach einem fehlerhaften Auftrag) erfolgt BASIC-Error 73.

Ansonsten sind die Fehlerschlüssel zu beachten, die die ENV-Anweisung (Funktion 1 - Statusabfrage) liefert.

Das Magnetband

15.3.2 Fehlerstatuscodes bei Anwendung des CALL 70

Nach jeder Magnetbandoperation mit dem CALL 70 wird in der Statusvariablen ein Statuscode übergeben. Grundsätzlich sollte auf jeden dieser Statuscodes gezielt reagiert werden.

In der folgenden Liste wird eine Übersicht über die möglichen Statuscodes mit Bedeutung und weiteren Hinweisen gegeben:

Status	Bedeutung	Erläuterung und Reaktion
0	Operation war erfolgreich	-
1	Bandgerät nicht verfügbar	Bandgerät von anderem Teilnehmer eröffnet. Später erneut eröffnen.
2	Bandgerät nicht bereit	Bandgerät betriebsbereit ("online") setzen.
3	Band ist schreibgeschützt	Schreibring fehlt, einsetzen und Operation wiederholen.
4	Schreib-/Lesefehler	Vorgang wiederholen, gegebenenfalls Gap schreiben.
5	EOF (Bandmarke gelesen)	Lesen abbrechen.
6	EOT (Bandende-Spiegel erreicht)	Beim Lesen: Lesen abbrechen Beim Schreiben: Durch ⟨NVar4⟩ prüfen, ob der letzte Block vollständig geschrieben wurde. Datei schließen.
7	Unzulässige Operation	Verursachende Operation überprüfen (⟨NumPar1⟩ ⟨⟩ 0? Einheit nicht eröffnet?).

16 Die Magnetbandcassette

Das System 8870 erlaubt den Anschluß eines Magnetbandcassettengerätes für die Systempflege per Cassette, zur Datensicherung und zum Datenaustausch mit anderen Systemen.

16.1 Organisation der Daten auf der Cassette

Die auf der Magnetbandcassette gespeicherten Daten sind im Rahmen von Dateien organisiert. Die einzig mögliche Organisationsform ist sequentiell. Die Verwaltung der Dateien einschließlich eventuell vorhandener Kennsätze muß vollständig vom Anwenderprogramm durchgeführt werden.

Das Cassettenband kann zwischen der Bandanfang- (BOT) und Bandendemarke (EOT) in nahezu beliebiger Weise benutzt werden; ein Update von Blöcken innerhalb des logischen Bandanfangs und -endes ist jedoch nicht erlaubt! Physikalisch gesehen wird in diesem Bereich lediglich zwischen Datenblöcken, die bis zu 512 Bytes lang sein können, und Blocklücken (Gap) unterschieden. Das Gap dient zur Trennung zweier Blöcke, es hat für die Verarbeitung der Information keinerlei Bedeutung.

Ein Block ist eine zusammenhängende Informationseinheit, Umfang und Inhalt dieser Information wird durch das bearbeitende Programm festgelegt.

Sollen Cassetten mit anderen Systemen ausgetauscht werden, müssen bestimmte Vorschriften über den logischen Datenträgeraufbau eingehalten werden.
Logisch gliedert sich die Datenkapazität einer Cassettenseite in eine oder mehrere Dateien. Die Dateien wiederum bestehen aus beliebig vielen Datenblöcken.

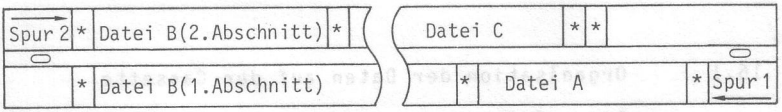
Zur logischen Gliederung des Bandes werden sogenannte Bandmarken benutzt. Dies sind 1-Zeichen-Blöcke mit dem Oktal-Code 377.

Jedes Band endet mit einer Doppelbandmarke. Zwischen Bandanfang und dieser Doppelbandmarke liegen die Dateien, die jeweils durch eine Bandmarke getrennt sind.
Bei Dateien mit Kennsätzen ist der Kennsatz-Bereich der Datei vom Datenbereich ebenfalls durch eine Bandmarke abgetrennt.

Die Magnetbandcassette

Bänder ohne Kennsätze

Bei einem Band ohne Kennsätzen ergibt sich folgende Bandstruktur:



* Bandmarke

Bänder mit Kennsätzen

Es gibt zwei genormte Stufen von Kennsatzanordnungen auf Cassetten:

1. Die Kompaktstufe, deren Kennsätze 32 Zeichen lang sind und mit einer numerischen Tastatur erstellt werden können.

2. Die Ausbaustufe, deren Kennsätze 80 Zeichen lang sind.

Eine genaue Beschreibung von Aufbau und Anordnung der Kennsätze ist in den Normen DIN 66229 und DIN 66029 enthalten.

16.2 Zugriff auf Cassettendateien

Die Cassetten-Einheit kann nur von einem Teilnehmer zu einer Zeit eröffnet werden. Zur Programmierung der Cassette steht die Anweisung

```
CALL 80
```

zur Verfügung. CALL 80 bildet die Schnittstelle zwischen dem BASIC-Interpreter RUN und dem Kanalprogramm $CAS.

16.2.1 Eröffnen des Magnetbandcassettengeräts

Mit dem Eröffnen wird das Magnetbandcassettengrät gegen den Zugriff anderer Teilnehmer gesperrt. Der Deckel der Cassetten-Einheit wird geöffnet und das Cassetten-Kanalprogramm wartet 105 Sekunden auf das Einlegen der Cassette und Schließen des Deckels. Die eingelegte Cassette wird auf BOT zurückgespult.

Syntax:

```
CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩
                                [,⟨SVar1⟩, ⟨SVar2⟩]
```

⟨NAusdr⟩ : 80

⟨NumPar1⟩ : 0 (MBC-Einheit)

⟨NumPar2⟩ : Funktionsvariable:
11: MBC zur Verarbeitung eröffnen

⟨NVar3⟩ : Statusvariable.

⟨SVar1⟩ : Eingabe-Codetabelle (256 Byte).

⟨SVar2⟩ : Ausgabe-Codetabelle (256 Byte).

Soll mit Codewandlung gearbeitet werden, müssen ⟨SVar1⟩ und ⟨SVar2⟩ codiert und vor Eröffnung geladen sein.

Die Magnetbandcassette

16.2.2 Verarbeitung

16.2.2.1 Verwaltung des Datenträgers

Mit diesen Funktionen wird das Zurückspulen des Cassettenbandes sowie das Auswerfen einer Cassette ermöglicht.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩

⟨NAusdr⟩ : 80

⟨NumPar1⟩ : 0 (MBC-Einheit)

⟨NumPar2⟩ : Funktionsvariable:
1: Cassettenband auf BOT zurückspulen
2: Deckel öffnen und Cassette auswerfen

⟨NVar3⟩ : Statusvariable.

16.2.2.2 Cassettenband vor- oder zurücksetzen

Das Band wird blockweise, d.h. von Gap zu Gap positioniert.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩

⟨NAusdr⟩ : 80

⟨NumPar1⟩ : 0 (MBC-Einheit)

⟨NumPar2⟩ : Funktionsvariable:
3: Band um einen Block vorsetzen
4: Band um einen Block zurücksetzen

⟨NVar3⟩ : Statusvariable.

16.2.2.3 Block lesen/schreiben

Übertragen eines Datenblocks, Länge $\geq$ 1 und $\leq$ 512 Bytes. Die Übertragung eines Datenblocks wird beendet, wenn die dimensionierte Länge von ⟨SVar⟩ oder die in ⟨NVar4⟩ vorgegebene Anzahl Byte erreicht ist. Die tatsächlich übertragene Anzahl Zeichen, die in ⟨NVar5⟩ abgestellt wird, ist zu überprüfen.
Nach der Übertragung ist das Band auf dem Gap hinter dem übertragenen Block positioniert.

Bei Lese-/Schreibfehlern wird der Vorgang bis zu achtmal wiederholt.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩, ⟨NVar4⟩, ⟨NVar5⟩, ⟨SVar⟩

⟨NAusdr⟩ : 80

⟨NumPar1⟩ : 0 (MBC-Einheit)

⟨NumPar2⟩ : Funktionsvariable:
0: Block lesen und nach ⟨SVar⟩ übertragen
1: Inhalt von ⟨SVar⟩ als Block schreiben

⟨NVar3⟩ : Statusvariable.

16.2.3 Schließen des Magnetbandcassettengeräts

Mit dieser Funktion wird die Verarbeitung abgeschlossen, die Cassette zurückgespult und ausgeworfen.

Syntax:

CALL ⟨NAusdr⟩, ⟨NumPar1⟩, ⟨NumPar2⟩, ⟨NVar3⟩

⟨NAusdr⟩ : 80

⟨NumPar1⟩ : 0 (MBC-Einheit)

⟨NumPar2⟩ : Funktionsvariable: 13 (Schließen)

⟨NVar3⟩ : Statusvariable.

16.3 Fehlerbehandlung

Nach jeder Cassetten-Operation wird in der Statusvariablen ein Statuscode übergeben. Grundsätzlich sollte auf jeden dieser Statuscodes gezielt reagiert werden.

In der folgenden Liste wird eine Übersicht über die möglichen Statuscodes mit Bedeutung und weiteren Hinweisen gegeben:

Status	Bedeutung	Erläuterung und Reaktion
0	Operation war erfolgreich	-
1	MBC-Einheit nicht verfügbar	MBC-Einheit von anderem Teilnehmer eröffnet. Später erneut eröffnen.
2	MBC-Einheit nicht bereit	MBC-Einheit betriebsbereit setzen (Deckel schließen).
3	Cassette ist schreibgeschützt	Zunge an der Rückseite ausgebrochen.
4	Schreib-/Lesefehler	Vorgang wiederholen, gegebenenfalls Cassette wechseln.
5	Zeit überschritten	Beim Eröffnen: Cassette einlegen und Deckel schließen. Sonst: Cassette wechseln.
6	EOT (Bandendemarke erreicht)	Beim Lesen: Lesen abbrechen Beim Schreiben: Durch ⟨NVar4⟩ prüfen, ob der letzte Block vollständig geschrieben wurde. Datei schließen.
7	Unzulässige Operation	Verursachende Operation überprüfen (⟨NumPar1⟩ ≠ 0? Einheit nicht eröffnet?).

Programmstruktur

17 Programmstruktur

17.1 Die Ausführung eines Programms

Ein BASIC-Programm, das ausgeführt werden soll, muß in einen Anwenderbereich des Hauptspeichers (Partition) geladen sein.

Hierzu bestehen zwei Möglichkeiten:

- Das Programm wurde zuvor eingegeben oder geändert und wird durch das BASIC-Kommando RUN gestartet.

- Ein gesichertes Programm wird aus der Programmdatei durch Eingabe des Dateinamens nach der entsprechenden Anwahl im EXPERT-Selektor in die Partition geladen und gestartet.

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

Stacks = Arbeitsbereich des BASIC-Interpreters RUN für: - Anwenderfunktionen-Liste - Unterprogramm-Keller - FOR-NEXT-Keller - Variablen-Liste
Tabelle der Zeilennummern (in aufsteigender Folge) mit Verweis auf die Adresse der zugehörigen Anweisung.
Anweisungen in komprimierter Form (Zwischencode).
Variablen-Datenbereich. Der Bereich wird erst zur Laufzeit des Programmes beim Durchlaufen der DIM-Anweisungen und bei impliziter Deklaration angelegt.

Programmstruktur

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:

- GOTO
- GOSUB
- RETURN
- CHAIN
- LINK
- IF
- ON
- END
- STOP
- NEXT

und unter bestimmten Umständen auch durch FOR unterbrochen.

Bei Sprungbefehlen wird die Zeilennummer der nächsten auszuführenden Anweisung nach der Methode des binären Suchens in der Tabelle der Zeilennummern ermittelt. Die Zeilennummer ist entweder in der Anweisung (GOTO, GOSUB) vorhanden oder wird aus den entsprechenden Stacks (FOR-NEXT-Keller, Unterprogramm-Keller) geholt.

Die Anweisungen STOP und END halten die Ausführung des Programmes an bzw. beenden die Programmausführung.

Während die GOTO-Anweisung das Fortsetzen der sequentiellen Ausführung der Anweisungen ab der angegebenen Zeilennummer veranlaßt, sind die etwas komplizierteren Vorgänge bei FOR-NEXT-, GOSUB-RETURN-, CALL-, CHAIN- und LINK-Anweisungen unter den folgenden Punkten beschrieben.

17.2 Die FOR-NEXT-Schleife

Die FOR-Anweisung kennzeichnet den Beginn und die NEXT-Anweisung das Ende einer Programmschleife.

In einer FOR-Anweisung wird angegeben:

1. die Laufvariable
2. der Startwert der Laufvariablen
3. der Endwert
4. die Schrittweite (optional, Standard: 1)

Bei der NEXT-Anweisung wird lediglich noch einmal die Laufvariable angegeben zur Kennzeichnung der Zusammengehörigkeit mit der entsprechenden FOR-Anweisung.

Kommt der Interpreter bei der Programmausführung zu der FOR-Anweisung, wird der Laufvariablen der Startwert zugewiesen und geprüft, ob der Endwert bereits über- bzw. unterschritten (bei negativer Schrittweite) ist. Wenn nicht, wird mit der Abarbeitung der Anweisungen fortgefahren. Beim Erreichen der zugehörigen NEXT-Anweisung wird zum aktuellen Wert der Laufvariablen die Schrittweite addiert und geprüft, ob der Endwert über-/unterschritten wurde. In diesem Fall wird mit der auf die NEXT-Anweisung folgenden Anweisung fortgefahren. Andernfalls wird mit dem geänderten Wert der Laufvariablen die Programmschleife erneut durchlaufen.

- Beispiel:

```
40 FOR I=1 TO 3
50 PRINT "WERT:";I
60 NEXT I
70 END
```

Abarbeitung:

```
1. I=1; I>3? Nein, fortfahren          (Anweisung 40)
2. WERT: 1 anzeigen                    (Anweisung 50)
3. I=I+1; I>3? Nein, Sprung nach 50    (Anweisung 60)
4. WERT: 2 anzeigen                    (Anweisung 50)
5. I=I+1; I>3? Nein, Sprung nach 50    (Anweisung 60)
6. WERT: 3 anzeigen                    (Anweisung 50)
7. I=I+1; I>3? Ja, Sprung nach 70      (Anweisung 60)
8. Programmende                        (Anweisung 70)
```

Programmstruktur

Der Wert der Laufvariablen darf in der Schleife nicht unkontrolliert verändert werden, weil sich damit auch die Anzahl Schleifenläufe ändert.

Start- und Endwert der Laufvariablen dürfen maximal sechs signifikante Stellen haben, da diese beiden Werte in 2%-Format gespeichert werden.

Programmschleifen können ineinandergeschachtelt werden (bis zu einer Tiefe von 8 Schleifen), sie dürfen sich aber nicht überschneiden.

- Beispiel:

```
  :                          :
┌─40 FOR I=1 TO 10         ┌─40 FOR I=1 TO 10
│ :                        │ :
│ ┌─65 FOR J=1 TO 10       │ ┌─65 FOR J=1 TO 10
│ │ :                      │ │ :
│ └─80 NEXT J              └─┼─80 NEXT I
│ :                          │ :
└─95 NEXT I                  └─95 NEXT J
  :                          :
zulässig                   unzulässig
```

17.3 Die Unterprogrammtechnik

Soll an verschiedenen Stellen des Programmes die gleiche Anweisungsfolge durchlaufen werden, so ist die Unterprogrammtechnik anzuwenden.

Den Sprung in das Unterprogramm bewirkt die Anweisung

GOSUB ⟨Zlnr⟩

wobei ⟨Zlnr⟩ die Zeilennummer der ersten Anweisung des Unterprogramms ist. Gleichzeitig wird in den Unterprogramm-Keller die Zeilennummer der auf die GOSUB-Anweisung folgenden Anweisung eingetragen. Die GOSUB-Anweisung sollte deshalb nicht am Programmende stehen.

Programmstruktur

Das Unterprogramm wird nun sequentiell abgearbeitet bis zum Auftreten einer RETURN-Anweisung. Diese veranlaßt den Sprung zu der im Unterprogramm-Keller abgestellten Zeilennummer.

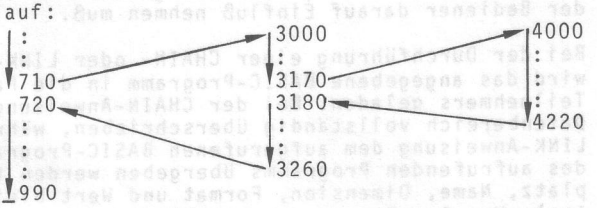
Durch Angabe eines ⟨NAusdr⟩ nach dem Schlüsselwort RETURN kann die Rückkehradresse modifiziert werden (siehe RETURN-Anweisung).

Innerhalb eines Unterprogrammes können weitere Unterprogramme aufgerufen werden. Die Unterprogrammtiefe darf 32 Stufen nicht überschreiten.

- Beispiel:

```
  :
710 GOSUB 3000
720 PRINT ...                    Hauptprogramm
  :
990 END oder CHAIN ...
3000 ...
  :
3170 GOSUB 4000
3180 LET ...                     Unterprogramm 1
  :
3260 RETURN
4000 ...
  :                              Unterprogramm 2
  :
4220 RETURN
```

Ablauf:



Programmstruktur

17.4 Unterprogramme in Maschinencode

Den Unterprogrammaufruf der Bibliotheks-Unterprogramme in Maschinencode bildet die Anweisung CALL.

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 Kapitel "CALL-Anweisungen" vorhanden.

17.5 Segmentierung/Verkettung von Programmen

Mit der Anweisung CHAIN kann ein Programm beliebige BASIC-Programme oder Processoren starten. Mit der LINK-Anweisung können ebenfalls BASIC-Programme gestartet werden. Darüberhinaus hat die LINK-Anweisung die Eigenschaft, Variablen dem neuen BASIC-Programm übergeben zu können.

Durch diese beiden Anweisungen ist die Möglichkeit gegeben, 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 Programmfolgen. In Abhängigkeit von Programmzuständen können bestimmte Programme nachgeladen werden, ohne daß der Bediener darauf Einfluß nehmen muß.

Bei der Durchführung einer CHAIN- oder LINK-Anweisung wird das angegebene BASIC-Programm in die Partition des Teilnehmers geladen. Bei der CHAIN-Anweisung wird der Datenbereich vollständig überschrieben, während bei der LINK-Anweisung dem aufgerufenen BASIC-Programm Variablen des aufrufenden Programms übergeben werden (Speicherplatz, Name, Dimension, Format und Wert bleiben erhalten). Von Programmen ohne GLOBAL-Anweisung werden alle dimensionierten und nicht mit DEALLO freigegebenen Variablen übergeben, von Programmen mit GLOBAL-Anweisungen alle in diesen Anweisungen genannten, soweit sie nicht mit der LINK-Anweisung von der Übergabe ausgeschlossen werden.

17.6 Hinweise für laufzeitoptimale Programmierung

17.6.1 BASIC-Optimizer

Es besteht immer der Wunsch, möglichst schnell ablaufende Programme zu haben. Im folgenden Kapitel wird beschrieben, wie BASIC-Programme programmiert bzw. umgestellt werden, um eine möglichst tiefe Compilation zu erhalten.

Erläuterung:

> Unter Compilation wird hier verstanden: Erzeugung eines Zwischencodes, der linear interpretiert werden kann und Variablenadressen enthält.

Diese Compilation wird mit dem Optimizer, der im SAVE-Kommando mit Parameter ",D," implizit aufgerufen wird, erreicht. Je mehrman sich an die hier beschriebenen Regeln hält, desto besser wird der Zwischencode und somit die Laufzeit.

Man kann 3 Qualitäten von Zwischencode erzeugen:

- B-File SAVE...
- D-File SAVE,D,...
- D-File mit Variablenadressen SAVE,D,...

Variablenadressen werden erzeugt, wenn die Variablen wie im folgenden beschrieben zu Anfang des Programms deklariert werden.

Für D-Files besteht ein spezieller Interpreter; je nachdem, welches File vorliegt, wird der alte oder der neue Interpreter aufgerufen.

17.6.2 Programm-Gliederung

Einzelprogramm

Wenn ein Programm (Programmcode und Variablenbereich) kleiner als die maximal freigegebene Partitiongröße ist, erübrigt sich eine Aufteilung des Programms in Teilprogramme. Alle Variablen sind lokal und werden mit DIM-Anweisung am Anfang des Programms geschrieben.

Programmstruktur

Teilprogramme im Programmpaket

Wenn sich eine Anwendung aus Platzgründen nicht in einem Programm realisieren läßt, verwendet man Teilprogramme, die ein "Programmpaket" bilden. Teilprogramme können über CHAIN-Anweisungen miteinander verbunden sein. Da beim Laden eines Teilprogramms der Variablenbereich gelöscht wird, ist eine Übergabe von Variablenwerten nur über Dateien oder den Commonbereich möglich. In derartigen Teilprogrammen sind die Variablen ebenfalls lokal und sind mit DIM-Anweisungen am Anfang jedes Teilprogramms zu schreiben. Jedes Teilprogramm nutzt zu seiner Laufzeit den gesammten Datenbereich.

Globaler- und lokaler Datenbereich

Da die Übergabe von Variablen mittels Dateien laufzeitungünstig ist, wird oft der Variablenbereich in einen globalen und einen lokalen Datenbereich unterteilt. Diese Aufteilung sollte man zu Anfang der Programmpaketentwicklung festlegen.

Variable, die über mehrere Teilprogramme referiert werden (Globale Variable), sind am Anfang eines jeden Teilprogramms mit der neuen GLOBAL-Anweisung zu schreiben. Vor einer GLOBAL-Anweisung darf als einzige Ausnahme die REM-Anweisung stehen. Wichtig ist weiterhin, daß in jedem Teilprogramm des Programmpakets die gleiche Global-Reihenfolge und -Größe bestehen muß. Es ist jedoch zulässig die Anzahl der Globals von einem zum nächsten Teilprogramm zu erhöhen oder zu vermindern. Zusätzliche Globals müssen jedoch in eine neue GLOBAL-Anweisung geschrieben werden.

Die übrigen Variablen gelten ausschließlich für das jeweilige Teilprogramm (lokale Variable) und sollten unmittelbar hinter den GLOBAL-Anweisungen als DIM-Anweisungen geschrieben werden.

Die Teilprogramme werden mittels der LINK-Anweisung verbunden. Im Gegensatz zu CHAIN-Anweisungen sorgt die LINK-Anweisung dafür, daß der globale Datenbereich beim Laden des nächsten Teilprogramms nicht gelöscht wird. Die LINK-Anweisung bietet die Möglichkeit zur Verkleinerung des globalen Variablen-Bereiches, der dem folgenden Teilprogramm übergeben werden soll (siehe spätere Beispiele).

IF ESC- und IF ERR 0-Anweisung

Die Anweisungen IF ESC und IF ERR 0 sollen hinter den globalen und lokalen Variablen mit Konstanten als Index stehen. D.h. Fehler, die bei diesen GLOBAL- oder DIM-Anweisungen auftreten können, müssen zur Testzeit behoben werden. Das Programm kann während des Durchlaufens der GLOBAL- und DIM-Anweisungen nicht mit der ESCAPE-Taste unterbrochen werden.

17.6.3 Umstellen alter Programme

Für die Umstellung der Programme kann das ANALYSE-Kommando benutzt werden. Hiermit werden kritische Fälle in der Programmliste markiert (siehe Kap. "BASIC-Kommandos" und "Programmwartung").

Globale Variable

- Alle Globalen Variablen, die zwischen 2 Teilprogrammen eines Programmpakets übernommen bzw. übergeben werden, suchen. Global sind die Variablen, die nicht in der DEALLO-Anweisung vor der LINK-Anweisung stehen.

- Die Globalen Variablen als GLOBAL-Anweisungen an den Anfang der Teilprogramme stellen. Auf gleiche Reihenfolge und gleiche Dimensionierung der Globalen Variablen achten. Bei zunehmender Anzahl von Globalen Variablen müssen die neuen Globalen Variablen in einer neuen GLOBAL-Anweisung stehen. Die GLOBAL-Anweisungen werden bei D-Files zur Laufzeit nicht durchlaufen, sondern es wird nur der Variablenbereich entsprechend initialisiert. Bei B-Files werden sie nur durchlaufen, wenn das Programm nicht mit einem "LINK" aufgerufen wurde.

- DEALLO-Anweisung vor LINK-Anweisung streichen.

- IF ESC- und IF ERR 0-Anweisungen hinter GLOBAL- bzw. DIM-Anweisungen legen.

- GLOBAL-Anweisungen mit variablem Index sind nicht zulässig (also GLOBAL S$(7) und nicht GLOBAL S$(I*J)). Hier muß die Maximallänge benutzt werden. Ist dies nicht möglich, kann das Programmpaket nicht umgestellt werden.

Programmstruktur

- Teilprogramme alter Struktur mit DEALLO/LINK sind mit umgestellten Teilprogrammen mit GLOBAL-Anweisungen innerhalb eines Programmpakets unverträglich!

Lokale Variablen

- Alle explizit und implizit dimensionierten Lokalen Variablen in den jeweiligen Teilprogrammen suchen. Die Crossreference-Liste des ANALYSE-Kommandos ist dabei hilfreich.

- Aus diesen Lokalen Variablen die nicht indizierten und diejenigen mit einer Konstanten als Index heraussuchen und die DIM-Anweisungen hinter die GLOBAL-Anweisungen des jeweiligen Teilprogramms legen bzw. neu erfassen. Übereinstimmung in der %-Angabe der Dimensionierung beachten!

- Alle DIM-Anweisungen und implizite Dimensionierungen mit variablem Index suchen.

- Mehrfach-Dimensionierungen der gleichen Variablen suchen.

- Alle DEALLO-Anweisungen, die nicht vor der LINK-Anweisung stehen, suchen.

- Für die obigen 3 Punkte prüfen, ob diese Fälle vermeidbar sind und gegebenenfalls umstellen! Falls dies nicht geht, können diese DIM-Anweisungen mitten im Programm stehen bleiben, und es werden hierfür keine Adressen erzeugt.
  Wichtig: Tritt in einer DIM-Anweisung eine Dimensionierung mit variablem Index auf, werden für alle Variablen dieser Anweisung keine Adressen erzeugt. Darum Dimensionierungen mit variablen Indizes in eine getrennte DIM-Anweisung schreiben.

- Indizierte Variable werden oft aus Laufzeitgründen, anstelle einer Zuweisung von "0" mittels einer FOR/NEXT-Schleife, durch die Konstruktion DEALLO "Variablen" und DIM "Variablen" gelöscht. Hier sollte eine CLEAR-Anweisung benutzt werden (siehe Beispiele).

Programmstruktur

o Erläuternde Beispiele:

- 10 REM BESCHREIBUNG DATENBEREICH:
  20 GLOBAL 1%,A,B,C,2%,D,E(8),F(6,9) /* GLOBALE VAR.
  30 GLOBAL A$(12),B$(50)
  40 GLOBAL usw.
  50 DIM 1%,I,J,K(5),3%,L(8,6) /* LOKALE VARIABLE
  60 DIM usw.
  --- bis hier Adressenerzeugung möglich ---
  70 IF ESC GOSUB ...
  80 IF ERR 0 GOSUB ...
  :
  100 INPUT A
  :
  170 DIM 2%,A1(A) /* VARIABLE ALS INDEX!
  180 DIM A2(7,6)
  :
  200 LET A=A8 /* IMPLIZITE DIMENSIONIERUNG FÜR A8
  UNGÜLTIG
  :
  250 DEALLO A2 /* HIER "CLEAR A2" NUTZEN UND
  260 DIM A2(7,6) /* ANWEISUNG 180 IN LOCALTEIL LEGEN

- Globalbereich mit wechselnden Längen:

```
10 REM BESCHREIBUNG DATENBEREICH:
20 GLOBAL 3%, A,B,C,D   /* GLOBALBEREICH
30 DIM 1%,X   /* LOCALBEREICH
:
50 ON X GOTO 100,200,300
:
100 LINK "PROG1"   /* GLOBALBEREICH BIS D ÜBERGEBEN
:
200 LINK;B;"PROG2"   /* GLOBALBEREICH BIS D VERKÜRZT
:
200 LINK;C;"PROG3"   /* GLOBALBEREICH BIS C VERKÜRZT

10 REM PROG1
20 GLOBAL 3%, A,B,C,D   /* GLEICHER GLOBALBEREICH

10 REM PROG2
20 GLOBAL 3%, A,B   /* VERKÜRZTER GLOBALBEREICH

10 REM PROG3
20 GLOBAL 3%, A,B,C   /* VERKÜRZTER UND ERWEITERTER
                         GLOBALBEREICH
25 GLOBAL 3%, E,F   /* NEUE ANWEISUNG VERWENDEN!
```

Programmstruktur

17.6.4 Weitere wichtige Hinweise

Compilieren und Decompilieren von Einzelprogrammen

Der Compiler (zur Erzeugung des Zwischencodes) und der Decompiler sind an das SAVE-Kommando angebunden (siehe Kap. "BASIC-Kommandos"):

Compilieren : SAVE ,D, [#<PartG>#] [<ProgName> [!]]
Syntaxfehler werden im Dialog gemeldet.

Decompilieren : SAVE ,B, [#<PartG>#] [<ProgName> [!]]

Filetyp bleibt: SAVE [#<PartG>#] [<ProgName> [!]]

Programmoptimierung (Batch-Compiler und -Decompiler)

Die Programmoptimierung (Compilieren von Programmen, die in einer LIBR-Liste aufgeführt sind) wird über den TAMOS-Selektor aufgerufen (siehe Kap. "Programmwartung"). Der Batch-Compiler benötigt eine Textdatei für die Programmnamenprotokollierung, Syntaxfehlerausgabe bzw. Ausgabe der Programmlängenunterschiede zwischen B- und D-File.

Ein Syntaxcheck wird in folgenden Fällen durchgeführt:

- Erfassen und Korrigieren eines BASIC-Programms
- Compilieren eines BASIC-Programms (SAVE ,D, ...)
- Laden eines Programms (LOAD <ProgName>), wenn in der Anwender-Partition ein BASIC-Programm in Form eines D-Files liegt.

Bei SAVE ohne <PartG>-Angabe und eingetragener Size im Dateikennsatz (Header) ist folgendes zu beachten:

Compilieren: Wird das D-File größer als das B-File, wird die Codelänge auf volle KB gerundet und die alte Size im Header um diesen Wert erhöht.
Wird das D-File kleiner als das B-File, bleibt die alte Size im Header erhalten.

Programmstruktur

Decompilieren: Wird das B-File größer als das D-File, bleibt die alte Size im Header erhalten. Wird das B-File kleiner als das D-File, wird die Codelänge auf volle KB gerundet und die alte Size im Header um diesen Wert verkleinert.

Deshalb ist zu empfehlen, die Size generell beim SAVE mit B-Files anzugeben. Wird durch das Compilieren die Size größer als die Active-File-Size, wird Error 81 gemeldet.

Codeexpansion und Programmgröße

Umgestellte BASIC-Programme mit über Compilation erzeugten Variablenadressen haben im Mittel die gleiche Codegröße wie in B-File-Darstellung. Dennoch müssen angesichts der maximalen Partitiongröße Ausreißer nach oben kontrolliert werden. Hilfreich sind folgende Ausgaben:

- Die bisherige maximale Programmgröße in Worten wird bei QUERRY ausgegeben unter der Spalte: LAUFZEITGROESSE.

  Wichtig: Die Programmgröße wird jeweils nach RUN, wenn die gerade entstandene Programmgröße größer als die vorherige ist (abhängig vom Programmweg), eingetragen. Bei SAVE wird die Programmgröße auf 0 gesetzt.

Die Differenz zwischen B- und D-File wird beim Batch-Optimizer im Textfile-Listing ausgegeben: DIF.: +-xxxx WORDS.

- In der Programmliste werden bei D-Files die Codegrößen von B- und D-Files sowie PROGRAM D-FILE SIZE xxxx WORDS angelistet.

Quellprogramm-Kompatibilität

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

Programmstruktur

Der Optimizer optimiert zwecks Code- und Laufzeiterspar-nis alle überflüssigen Klammern in den Ausdrücken heraus:

- A(0,0) = A und A(0) = A; ebenso ist A$(0) = A$ und A$(1) = A$.

- Bei Matrixfeldern ist A(5,0) = A(5), ebenso ist bei Strings A$(1,5) = A$, A$(0,5) = A$ und A$(3,5) = A$(3), wenn die dimensionierte Stringlänge dem 2. Index ent-spricht.

- Bei GLOBAL-und DIM-Anweisungen, die ursprünglich ohne vollständige Genauigkeitsangabe erfolgten und für deren Variablen Adressen erzeugt wurden, wird beim Listen der Anweisung stets die Genauigkeit vor die 1. numerische Variablen-Dimensionierung abgestellt. Die Maximallänge der Anweisung kann dadurch überschritten werden!

Ausnahme:

- Tritt der monadische (nur auf den ersten Operanden wirkende) Operator "-" auf, wird dieser Ausdruck stets geklammert, z.B. aus: -1+B wird (-1)+B.

Mischsysteme

BASIC-Programme können gemischt mit B- und D-Files lau-fen. Da zur Laufzeit dann 2 Interpreter referiert werden, muß in diesem Fall eine Partition mehr generiert sein, da andernfalls der Durchsatz schlechter werden kann.

- Keine Einschränkung ergibt sich bei Programmen ohne GLOBAL-Anweisungen (B-File -> LINK -> D-File -> LINK -> B-File).

- Sind GLOBAL-Anweisungen vorhanden, müssen "alle Teil-programme" eines Programmpakets umgestellt sein (anson-sten erhält man Fehler 85 zur Laufzeit).

- Es ist möglich, verschiedene "Programmpakete aus D-Files" mit "Programmpaketen aus B-Files" gemischt zu verwenden.

18 Testhilfen

Das BASIC-Programmiersystem stellt dem Anwender eine Reihe von Hilfen zur Verfügung, neue oder geänderte Programme auf syntaktische Richtigkeit und ordnungsgemäßen Ablauf zu testen.

Im einzelnen sind dies:

- BASIC-Fehleranzeige bei der Eingabe von Anweisungen
- BASIC-Fehleranzeige bei der Ausführung des Programms
- Einzelschrittausführung des Programms und Ausführungsprotokoll
- Die STOP-Anweisung
- Die IF ERR 0-Anweisung
- Die IF ESC-Anweisung
- Testlauf des Programms (wahlweise ab einer bestimmten Zeilennummer bis zu einer Zeilennummer)
- Direktausführung von Anweisungen
- TRACE-Kommando

18.1 Die BASIC-Fehleranzeige

Der BASIC-Processor ist in der Lage, bereits bei der Eingabe von Anweisungen bestimmte Fehler, vor allem lexikalische und syntaktische, zu erkennen. Weiterhin werden bei der Ausführung sowohl die Anweisungen einzeln als auch in ihrer Gesamtheit auf syntaktische und semantische Fehler überprüft. Wird hierbei oder schon beim Eingeben der Anweisungen ein Fehler festgestellt, wird er dem Anwender in der Form

```
ERROR # xx  AT <Zlnr>  IN <Programmname>
```

am Bildschirm angezeigt. Bei Fehlern nach der Eingabe einer Anweisung entfällt "AT <Zlnr> IN <Programmname>". Die Bedeutung der Fehlernummer xx kann nun der Liste der BASIC-Fehler entnommen oder mit dem BASIC-Kommando

```
HELP [xx]
```

am Bildschirm sichtbar gemacht werden.

Testhilfen

Auch die Fehleranzeige während der Programmausführung bricht das Programm nur in schwerwiegenden Fällen ab. Ansonsten dient die Anzeige nur der Anwender-Information, und das Programm wird weiter ausgeführt. Durch das Anzeigen der Nummer der fehlerverursachenden Anweisung kann der Fehler leicht lokalisiert werden. Die standardmäßige BASIC-Fehleranzeige kann mit Hilfe der IF-ERR 0-Anweisung durch eine eigene Fehlerbehandlungsroutine ersetzt werden.

- Beispiel:

  Auf die Eingabe:

  10 OPEN #3,"$LPT

  also ohne die abschließenden Anführungsstriche, erscheint am Bildschirm zusammen mit einem akustischen Signal in der Tastatur:

  ERROR # 4

  Auf Eingabe des Kommandos:

  HELP 4 oder nur HELP

  wird folgender Fehlertext angezeigt:

  FORMATFEHLER

18.2 Die IF ERR 0-Anweisung

Soll die standardmäßige BASIC-Fehleranzeige durch eine gezielte Fehlerreaktion ersetzt werden, ist die IF ERR 0-Anweisung anzuwenden.

Syntax:

IF ERR 0 [<Anweisung>]

Das Auftreten eines BASIC-Fehlers an einer beliebigen Stelle nach dieser Anweisung erzeugt die Ausführung von <Anweisung>.
In der Regel wird dies ein Sprung in ein Fehlerbehandlungs-Unterprogramm oder in der Testphase auf STOP sein.

Testhilfen

Während der Programmausführung kann nur eine IF ERR 0-Anweisung aktiv sein. Das bedeutet, daß jede IF ERR 0-Anweisung eine eventuell vorhergehende aufhebt.

Sollen BASIC-Fehler wieder in der Form

ERROR # xx AT ⟨Zlnr⟩ IN ⟨Programmname⟩

angezeigt werden, kann die gezielte Fehlerreaktion durch die Anweisung IF ERR 0 (ohne ⟨Anweisung⟩) wieder aufgehoben werden.

Anmerkung:

Die Fehlernummern:

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

können mit IF ERR 0 nicht abgefangen werden.

Alle Kanäle werden geschlossen und es wird in den BASIC-Processor verzweigt.
Ist keine IF ERR 0-Anweisung wirksam, werden Fehler mit:

ERROR # xx AT ⟨Zlnr⟩ IN ⟨Programmname⟩

gemeldet und das Programm wird weiter ausgeführt.

- Beispiele:

Fehlerbehandlung in der Testphase:

```
10 IF ERR 0 STOP
 :
```

Bei Auftreten eines Fehlers wird das Programm in jedem Fall unterbrochen und

STOP AT ⟨Zlnr⟩ ⟨Anweisung⟩

angezeigt, wobei ⟨Zlnr⟩ die Zeilennummer der fehlerverursachenden Anweisung ist.
Der Anwender kann sich nun im Direktausführungsmodus Variableninhalte anzeigen lassen, die möglicherweise den Fehler verursacht haben.

- 10 IF ERR 0 GOSUB ...
  :

  Unterprogrammsprung in eine Fehlerbehandlungsroutine. In dem Unterprogramm können Fehlernummer sowie fehlerverursachende Anweisung mit SPC 8 bzw. SPC 10 ausgewertet werden.

Speziell für die Fehlerbehandlung mit IF ERR 0 stehen die Sonderfunktionen CHN, FLN und PRN zur Verfügung.

18.3 Die IF ESC - Anweisung

Selbstbestimmte Reaktion auf die Betätigung der ESC-Taste

Syntax:

```
IF ESC [<Anweisung>]
```

Bei Betätigung der ESC-Taste wird die Anweisung ⟨Anweisung⟩ ausgeführt.
Ohne die IF ESC - Anweisung würde bei Auslösung der ESC-Taste das Programm unterbrochen und in den DEBUG - Modus umgeschaltet (s. "Die ESC - Taste").
IF ESC REM macht die ESC-Taste unwirksam.
IF ESC ohne ⟨Anweisung⟩ hebt die ESC-Tasten-Behandlung wieder auf.
Bei mehreren IF ESC - Anweisungen in einem Programm ist nur die aktuelle wirksam. Bei mehrmaligem Drücken der ESC-Taste wird der Merker "ESC gewesen" nur einmal gesetzt und nach Ausführung von ⟨Anweisung⟩ wieder gelöscht. Nach Aufruf eines Folgeprogramms durch CHAIN oder LINK wird der Merker ebenfalls gelöscht. In dem Folgeprogramm muß dann die Anweisung IF ESC ⟨Anweisung⟩ neu codiert sein, wenn die ESC-Tasten-Behandlung wieder genutzt werden soll.

- Beispiel:

```
:
100 IF ESC GOSUB 5000
:
:
5000 PRINT TAB(0,24);"ESC-TASTE GEDRUECKT";
5100 RETURN
```

18

18.4 Die Direktausführung von Anweisungen

Zur Unterstützung des Programmtestes per Dialog können einzelne BASIC-Anweisungen auch direkt nach der Eingabe ausgeführt werden. Die Direktausführung wird dadurch erzeugt, daß die Anweisung ohne Zeilennummer eingegeben wird und somit die Anweisung auch nicht in das Programm eingeordnet werden kann.

Wie bei der Eingabe jeder anderen Anweisung wird auch hier nach der Eingabe eine syntaktische Prüfung vorgenommen.

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

Davon ausgenommen, weil nicht direkt ausführbar, sind:

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

und in D-Files die Statements, in denen eine Anwenderfunktion referiert wird.

Wird eine dieser Anweisungen eingegeben, erscheint die Meldung:

ERROR # 55

(Direktausführung der Anweisung nicht möglich).

Testhilfen

- Beispiel:

Das Programm

```
 5 IF ERR 0 STOP
10 DIM A(10,10)
20 FOR I=1 TO 10
30   FOR J=1 TO 11
40     LET A(I,J)=I*J+I
50   NEXT J
60 NEXT I
```

laufe auf einen BASIC-Fehler mit Anzeige

```
STOP AT 40  LET A(I,J) = I*J+T
```

In diesem Fall ist es sinnvoll, sich die Variableninhalte von I, J und A(I,J) durch die Eingabe von

```
PRINT I,J, A(I,J)
```

am Bildschirm anzeigen zu lassen.

18

18.5 Der Einzelschritt-Testlauf und das Ausführungsprotokoll

Erkennt das BASIC-System keinen Fehler und läuft das Programm trotzdem nicht wie gewünscht ab, wird eine weitere Möglichkeit zum Auffinden eines Fehlers angeboten.

Liegt ein für das System nicht erkennbarer logischer Fehler vor, der vom Anwender auch nur an der Ausgabe falscher Ergebnisse erkannt wird, ist es bei normaler Programmausführung sehr schwer herauszufinden, welche Anweisung den Fehler verursacht hat bzw. bei welcher Anweisung er zuerst aufgetreten ist.
In diesem Fall kann man das Programm Anweisung für Anweisung ausführen lassen, d.h. nach Ausführung einer Anweisung muß die Ausführung der nächsten Anweisung erst mit der "CR"-Taste bestätigt werden.

Die Einzelschrittausführung aktiviert das BASIC-Kommando

[<Zlnr>] DEBUG

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

Die Angabe <Zlnr> ist nur dann sinnvoll, wenn bereits ein initialisierendes RUN- oder DEBUG-Kommando durchgeführt wurde.
DEBUG initialisiert den Einzelschritt-Test (DEBUG-Modus), gibt die erste bzw. die unter <Zlnr> angegebene Anweisung aus :

STOP AT xxx <Anweisung>

wobei xxx die Zeilennummer ist.

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

Außerdem wird die Einzelschrittausführung durch eine STOP-Anweisung oder ein RUN<Zlnr>-Kommando initialisiert.

Testhilfen

Ein Ausführungsprotokoll erhalten Sie mit dem BASIC-Kommando

⟨Zlnr⟩ DEBUG ⟨DatName⟩

Hierbei wird das Programm ausgeführt und die durchlaufenen Anweisungen werden in der durch ⟨DatName⟩ angegebenen Datei bzw. auf dem angegebenen Drucker protokolliert.

Die Eingabe jedes anderen Kommandos ungleich DEBUG oder die Eingabe einer BASIC-Anweisung mit Zeilennummer löschen den DEBUG-Modus.

Direktausführung von Anweisungen oder Betätigung der ESC-Taste löscht den DEBUG-Modus nicht.

- Beispiel:

  300 DEBUG

  Beginnen des DEBUG-Modus bei der Zeilennummer 300 . Das Ausführen dieser Anweisung geschieht erst nach Betätigung der CR-Taste. Anschließend wird die Anweisung mit der darauffolgenden Zeilennummer angezeigt.

- Beispiel:

  DEBUG $LPT

  Der DEBUG-Modus beginnt mit der ersten Anweisung des Programms. Auf dem Drucker $LPT (s. Gerätezuordnungstabelle im Systembedienungsselektor) wird das Ausführungsprotokoll ausgegeben.

18.6 Testlauf des Programms

Für eine testweise Ausführung eines neu eingegebenen oder geänderten Programms braucht der BASIC-Processor nicht verlassen zu werden. Es genügt die Eingabe des BASIC-Kommandos

```
RUN
```

aus dem BASIC-Kommandomodus. In diesem Fall wird das BASIC-Programm in der Form, in der es sich im Hauptspeicher befindet, von Programmanfang bis -ende ausgeführt. Eine Modifikation des Kommandos ist durch die Angabe einer Start- und/oder End-Zeilennummer möglich:

```
[<Zlnr1>] RUN [<Zlnr2> [,<Zlnr2>][4]]
```

<Zlnr1> : Zeilennummer der Anweisung, ab der die Ausführung gestartet werden soll.

Diese Angabe ist nur zulässig, wenn

- bereits ein "initialisierendes" RUN-Kommando durchgeführt oder
- das Programm bis <Zlnr1> im DEBUG-Modus ausgeführt wurde

und das Programm anschließend noch nicht geändert wurde. Fehlt diese Angabe, wird das Programm mit der ersten Anweisung gestartet ("initialisierendes" RUN).

<Zlnr2> : Zeilennummern, bei deren Erreichen das Programm angehalten werden (STOP-Bedingung) und in den DEBUG-Modus gehen soll.

Ist eine der angegebenen Zeilennummern erreicht, wird die Meldung:

```
STOP AT <Zlnr2> <Anweisung>
```

für die entsprechende <Zlnr2> angezeigt. Zu dieser Zeit sind alle angegebenen Stop-Bedingungen gelöscht. Diese Anweisung ist noch nicht ausgeführt.

An dieser Stelle kann im DEBUG-Modus weitergetestet oder es können im Direktausführungsmodus Variableninhalte angezeigt werden, um die ordnungsgemäße Durchführung des Programms zu überprüfen.

Testhilfen

- Beispiel:

```
RUN 300
```

Starten eines Programmes am Programmanfang. Das Programm wird bei Erreichen der Zeilennummer 300 angehalten.

```
PRINT X,Y,A(X,Y)
```

Überprüfen der Variableninhalte von X, Y und A(X,Y).

```
300 RUN
```

Weiterführung bis zum Programmende.

18.7 Die STOP-Anweisung

Soll in der Testphase des Programms die Ausführung oftmals an den gleichen Stellen unterbrochen werden, empfiehlt es sich, an diesen Stellen die Anweisung

```
STOP
```

einzuschieben. Diese Anweisung hat zur Folge, daß das Programm an dieser Stelle mit der Anzeige

```
STOP AT <Zlnr>  STOP
```

unterbrochen wird und sich im DEBUG-Modus befindet. <Zlnr> ist die Zeilennummer der STOP-Anweisung. Eröffnete Dateien bleiben eröffnet.
Mit der STOP-Anweisung ist die Ausführung des Programmes also genauso wie ohne STOP-Anweisung mit dem Kommando:

```
RUN<Zlnr>
```

Nach der Programmunterbrechung besteht ebenfalls die Möglichkeit, im DEBUG-Modus weiterzutesten und sich Variableninhalte ausgeben zu lassen, um die korrekte Durchführung zu überwachen.

Anschließend kann das Programm mit

```
<Zlnr>RUN
```

zu Ende geführt werden.

Nach der Testphase sollte die STOP-Anweisung wieder aus dem Programm entfernt werden.

Testhilfen

18.8 Die "ESC"-Taste

Eine weitere Möglichkeit, die Ausführung eines BASIC-Programms zu unterbrechen, bietet die "ESC-Taste".

Nach Drücken der "ESC"-Taste wird die Programmausführung angehalten, der DEBUG-Modus eingeleitet und die Zeilennummer der nächsten auszuführenden Anweisung durch

STOP AT <Zlnr> <Anweisung>

angezeigt. Es besteht die Möglichkeit, im DEBUG-Modus weiterzutesten und sich Variableninhalte im Direktausführungsmodus anzeigen zu lassen.

Zu beachten ist, daß die Wirkung der "ESC"-Taste aufgehoben sein kann. Eine Programmunterbrechung ist dann nur durch Drücken von "CTL-Y" und "ESC" möglich, und falls der Anwender SCOPE-Berechtigung besitzt.

Soll mit der "ESC"-Taste während der Programmtestphase die Ausführung unterbrochen werden, müssen IF ERR 0 - und IF ESC - Anweisungen vorübergehend mit REM versehen werden.
Das Erzeugen von BASIC-Fehler 99 durch Drücken der "ESC"-Taste oder die Benutzung der IF ESC - Anweisung bietet dem Anwender die Gelegenheit, die Programmausführung gezielt mit der "ESC"-Taste zu beeinflussen.

- Beispiele:

```
10 IF ESC GOSUB...      Sprung in die ESC-Routine bei
                        betätigter ESC-Taste


10 IF ESC GOSUB...      Sprung in die ESC-Routine ...
 :
230 IF ESC              Zeile 10 unwirksam - erneut Stan-
                        dardreaktion bei betätigter ESC-
                        Taste


10 IF ESC REM           Die ESC-Taste ist unwirksam
```

18.9 Der Variablen-Dump

Nach Programmablauf, STOP oder ESCAPE können alle im Programm dimensionierten Variablen mit ihrem Inhalt auf Bildschirm, Drucker oder in Textdatei ausgegeben werden.

Syntax:

```
TRACE [<Datei>]
```

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

<Datei> : Angabe des Druckers oder der Textdatei
Fehlt diese Angabe, wird auf dem Bildschirm ausgegeben.

Testhilfen

18.10 Der Variablen-Stop

Durch 2 weitere Formate des TRACE-Kommandos läßt sich ein Programmstop bei Zuweisungen an bestimmte Variablen bzw. bei Zuweisung bestimmter Werte an bestimmte Variablen erreichen. Das Kommando wird vor einem Programmstart (RUN) gegeben.

Das Kommando

[⟨Zlnr1⟩] TRACE [⟨Zlnr2⟩] {,⟨VarName⟩}^{5}

bewirkt einen Programmhalt und Übergang in den DEBUG-Modus bei der ersten Anweisung im Programmablauf, in der eine Zuweisung an eine der Variablen ⟨VarName⟩ geschieht. Diese Anweisung ist schon ausgeführt und wird angezeigt. Auch Name und Wert der verursachenden Variablen werden ausgegeben. Das Programm kann mit ⟨Zlnr⟩ RUN fortgesetzt werden, wobei ⟨Zlnr⟩ die auf die Anweisung folgende (!) Zeilennummer ist.

Das Kommando

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

bewirkt ebenso einen Programmstop sowie die Ausgabe der Anweisung und der entsprechenden Variablen nach Ausführung einer Anweisung, durch die ⟨Bedingung⟩ erfüllt wird.

⟨Bedingung⟩::= { ⟨NVarName1⟩⟨VglOper⟩ {⟨NVarName1⟩ / ⟨Dezzahl⟩} / ⟨NVarName⟩ { =|⟨⟩} ⟨Dezzahl1⟩:⟨Dezzahl2⟩ / ⟨SVarName1⟩⟨VglOper⟩ {⟨SVarName2⟩ / ⟨SLit⟩} }

⟨NVarName⟩=⟨Dezzahl1⟩:⟨Dezzahl2⟩ ist erfüllt, wenn der Wert (der entsprechenden Komponente) von ⟨NVarName⟩ zwischen ⟨Dezzahl1⟩ und ⟨Dezzahl2⟩ (einschließlich) liegt. ⟨NVarName⟩ ⟨⟩ ⟨Dezzahl1⟩:⟨Dezzahl2⟩ ist erfüllt, wenn der Wert außerhalb des Bereichs liegt.

Die Länge von ⟨SLit⟩ ist auf 16 Zeichen begrenzt.
Sind ⟨Zlnr1⟩ oder ⟨Zlnr2⟩ angegeben, wirkt das TRACE-Kommando nur ab dem ersten Erreichen von ⟨Zlnr1⟩ (bzw. bis zum ersten Erreichen von ⟨Zlnr2⟩) im Programmablauf.

Weitere Hinweise siehe unter "BASIC-Kommandos" unter "TRACE".

TAMOS - Schnittstellen

19 TAMOS - Schnittstellen

19.1 Vorbereitung von Programmen zur Einbindung in Selektoren

Alle in TAMOS-Selektoren eingebundenen Anwenderprogramme müssen eine Reihe von Schnittstellenbedingungen erfüllen, um eine korrekte Ablaufsteuerung zu gewährleisten.

Die Schnittstellen werden durch Standardunterprogramme gebildet, die der BASIC-Programmierer mit dem BASIC-Kommando DUMP in Textdateien abstellen und in jedes Programm über das BASIC-Kommando LOAD einbinden sollte, sowie durch den Aufruf bestimmter TAMOS-Programme am Programmende.

Da die Standardunterprogramme eine Reihe von BASIC-Variablen benötigen, sollten folgende Variablen im Anwenderteil der Programme nicht benutzt werden:

- String-Variablen mit Ziffer 7 (z.B: H7$, L7$)

- die Vektoren S und T
  (S nur beim Arbeiten mit Job-Spooler)

Die Variablen werden von den Standardunterprogrammen wie folgt belegt:

Ohne Spooling:

Name	Beschreibung
H7$(122)	Kopfzeile
L7$(126)	Logdatei-Satz
M7$(55)	Nachricht
2%:	
T(0)	Arbeitsplatznummer
T(1)	Nachrichtennummer in TF.PARAM

Beim Arbeiten ohne Job-Spooler ist also zu Programmanfang folgende DIM-Anweisung zusätzlich erforderlich:

```
DIM H7$(122),L7$(126),M7$(55),2%,T(1)
```

TAMOS - Schnittstellen

Mit Job-Spooler:

Name	Beschreibung
H7$(122)	Kopfzeile
L7$(126)	Logdatei-Satz
M7$(55)	Nachricht
D7$(20)	Job-Beschreibung
F7$(25)	Benutzerinformation
P7$(18)	LU/Textdatei- oder Programmname
S7$(25)	Papierbeschreibung
Z7$(160)	Benutzerinformation
2%:	
T(0)	Arbeitsplatznummer
T(1)	Nachrichtennummer in TF.PARAM
T(2)	Job-Erstellungszeit
T(3)	Teilnehmernummer (group/user)
1%:	
S(0)	Job-Typ
S(1)	RUN-mode
S(2)	Selektorkoordinate der 1 Ebene
S(3)	Selektorkoordinate der 2. Ebene
S(4)	Selektorkoordinate der 3. Ebene
S(5)	Lösch-, Job-Skip-, Abbruch-, Neustart-Code
S(6)	Anzahl Durchläufe
S(7)	Papiercode
S(8)	Erste Druckposition
S(9)	Letzte Druckposition
S(10)	Zeilen pro Seite
S(11)	Hilfsfeld
S(12)	Erste Seitennummer

Beim Arbeiten mit Spooling müssen also folgende DIM-Anweisungen zusätzlich aufgenommen werden:

```
DIM H7$(122),L7$(126),M7$(55),V7$(4),D7$(20),F7$(25)
DIM P7$(18),Z7$(160),S7$(25),1%,S(12),2%,T(3)
```

TAMOS - Schnittstellen

19.2 Starten eines Anwenderprogrammes

Unter TAMOS laufende Anwenderprogramme sind aus den entsprechenden Selektoren anzuwählen (s."Bedienerhandbuch"). Zwischen der Anwahl und dem eigentlichen Programmstart führt TAMOS folgende Arbeitsgänge durch:

- In der Datei TF.PORT wird eine Eintragung mit folgenden Informationen vorgenommen:

  - CPU-Zeit
  - RUN-Mode
  - Anwender-Nr.
  - Selektorkoordinate der 1. Ebene
  - Selektorkoordinate der 2.Ebene
  - Selektorkoordinate der 3.Ebene
  - Programmname (eventuell bei LU ≠ 0 mit LU-Nummer)
  - Benutzte logische Einheiten

  Diese Eintragung wird für jeden Arbeitsplatz durchgeführt.

- In der Common-Area (gemeinsamer Teilnehmerbereich) stellt TAMOS dem Teilnehmer, der das Programm gestartet hat, folgende Daten bereit:

  - Beschreibung der 3.Ebene
  - Beschreibung der 2.Ebene
  - Beschreibung der 1.Ebene
  - Datum
  - den Text: NACHRICHT:
  - die beiden Selektor-Parameter
  - Programmname und -nummer
  - Benutze logische Einheiten

TAMOS - Schnittstellen

- In die Log-Datei(TF.LOGFILE) wird ein Datensatz mit folgenden Informationen geschrieben:

  - Programm-Typ
  - Spezifikation
  - Arbeitsplatz-Nummer
  - Startzeit
  - Kopfzeile
  - Status: START

  Die Datenkanäle 0 und 1 werden von TAMOS mit den Dateien

  - TF.LOGFILE (Kanal 0) und
  - TF.PARAM (Kanal 1)

  eröffnet übergeben. Sie sollten von Anwenderprogrammen nicht geschlossen werden.

TAMOS - Schnittstellen

19.3 Kopfzeile und Nachrichtenzeile anzeigen

Zu Beginn jedes unter TAMOS laufenden Anwenderprogramms sollte das Standard-Unterprogramm "Kopfzeile ausgeben" aufgerufen werden.

Dieses Unterprogramm löscht den Bildschirminhalt und gibt die Kopfzeile (Zeile 0) sowie die Nachrichtenzeile (Zeile 24) aus.

Kopf- und Nachrichtenzeile werden als Hintergrundzeichen ausgegeben.

Anmerkung:

Programme, die nur am Phantom Port laufen, brauchen dieses Unterprogramm nicht aufzurufen.

- Benutzte Variablen: H7$(122) = Kopfzeile
  T(0) = Arbeitsplatznummer (2%)

- Unterprogramm:

```
1100 LET T(0)= SPC 6
1110 CALL 3, T(0), H7$
1120 PRINT USING " ## ";'CS';'SB';H7$(5,66); TAB(67,0);
       "#";T(0);H7$(67,74); TAB(0,24);H7$(75,89);'SF';
1130 RETURN
```

Der Aufbau der Kopf- und Nachrichtenzeile ist im Abschnitt "Übergabe von Standardparametern durch TAMOS" beschrieben.

19

TAMOS - Schnittstellen

19.4 Beenden eines Anwenderprogramms

Ein unter TAMOS gestartetes Anwenderprogramm soll nach seiner Beendigung in den Anwahlselektor zurückkehren. Außerdem soll auf aufgetretene Fehler standardmäßig reagiert werden. Deshalb muß jedes unter TAMOS gestartete Anwenderprogramm mit einem Aufruf (CHAIN) eines der Standard-Programme:

- TA.END
- TA.NCO
- TA.ABO

beendet werden. Weiterhin ist die Stellung des PAF (Program Abort Flag) zu berücksichtigen.

PAF (Program Abort Flag)

Da Programme aus verschieden kritischen Verarbeitungsabschnitten bestehen, ist in BASIC eine unterschiedliche Programmabbruchbehandlung einzelner Abschnitte mit dem CALL 29 möglich. Durch eine Veränderung des PAF (Programm Abort Flag) mit diesem CALL kann der Eintrag im SELEKTOR (ABBRUCH Y/N) übersteuert werden. Das PAF kann vier verschiedene Werte mit folgender Bedeutung annehmen:

0: Rücksprung in den Anwahlselektor, keine Fehlerbehandlung
1: Neustart des Programms (mit Bedienerentscheidung)
2: Aufruf eines Reparaturprogramms
3: Rekonstruktion 2. Generation

ab Rel. 5.1 zusätzlich

4: Neustart des Programms (ohne Bedienerentscheidung)
Das Flag wird von TAMOS bei Neustart auf 1 zurückgesetzt. Durch geeignete Abfragen ist sicherzustellen, daß die Fehlerbehandlung nicht in einer Endlosschleife erfolgt!

TAMOS - Schnittstellen

TA.END

Dieses Programm ist aufzurufen, wenn das Anwenderprogramm korrekt beendet wird. Es schließt alle eröffneten Kanäle und vermerkt die Beendigung im Logbuch (TF.LOGFILE). Weiterhin wurden zu Rel. 5.0 in der TAMOS-Archivdatei die Copy-Merker bei jedem Update und zusätzlich gemäß den Selektoreinträgen des Programms gesetzt.
Ab Rel. 5.1 setzt das Betriebssystem einen Copy-Merker nur bei jedem Update.

Anmerkung:

Sind beim Aufruf von TA.END die Kanäle 0 und 1 nicht eröffnet, wird statt TA.END automatisch TA.ABO aktiviert.

TA.NCO

Anwenderprogramme, die beendet werden sollen, ohne Copy-Merker zu setzen, sind bis Rel. 5.0 mit dem Aufruf von TA.NCO zu beenden. Dies ist dann sinnvoll, wenn ein Anwenderprogramm abgebrochen wird, bevor es Datenbestände auf Magnetplatten verändert hat. Da ab Rel. 5.1 die Verwaltung des Copy-Merkers vom Betriebssystem übernommen wird, unterscheidet sich TA.NCO von TA.END nur noch durch einen unterschiedlichen Logbucheintrag.

TA.ABO

Muß ein Anwenderprogramm abgebrochen werden, das bereits Daten auf einer Magnetplatte verändert hat, ist das Programm TA.ABO aufzurufen.

TA.ABO schließt alle eröffneten Datenkanäle und vermerkt den Programmabbruch im Logbuch. Am Arbeitsplatz wird der Abbruch in der Nachrichtenzeile angezeigt, sobald der TAMOS-Selektor erscheint.

Wird ein Programm mit Selektoreintrag ABBRUCH: NO über TA.ABO-Aufruf abgebrochen, ist es nicht möglich, weitere Programme zu starten, da TAMOS nur noch Programme zur Datenrekonstruktion zuläßt (NACHRICHT: NUR FORMATIERUNG ODER REKONSTRUKTION MOEGLICH).

TAMOS - Schnittstellen

19.5 Logbuch und Nachrichtendatei

Logbuch

In der Logdatei werden automatisch Systemzustände wie z.B. Programmstart, -ende, -abbruch oder Systemfehler protokolliert. Zusätzlich kann der Bediener Bemerkungen in das Logbuch eintragen. Für automatische Eintragungen aus dem Anwenderprogramm ist der BASIC-CALL 84 vorgesehen. Aufgetretene BASIC-Fehler können auch mit der LGERR-Anweisung eingetragen werden. Der Eintrag hat das feste Format:

ERROR # ⟨Ganzzahl⟩ AT ⟨Zlnr⟩ IN LU/⟨Prog.Name⟩

Zur Abgrenzung gegen Systemsoftware sollte der Typ 40= USERS und als Spezifikation die Benutzernummer angegeben werden (s.CALL 84).

Nachrichtendatei

Nachrichtendateien sind relative Dateien mit einer Satzlänge von 512 Byte. Für eine Nachricht werden grundsätzlich 50 Byte Text plus 1 Grenzzeichen reserviert. In einen Satz passen also 10 Meldungen. Die Standardnachrichtendatei im System heißt MESSAGES, es können jedoch noch beliebig viele andere Nachrichtendateien angelegt werden.

Für die Bearbeitung einer Nachrichtendatei steht der CALL 84 zur Verfügung. Mit dessen Hilfe können Nachrichten aus der Nachrichtendatei gelesen, am Bildschirm angezeigt und in die Logdatei eingetragen werden.

TAMOS - Schnittstellen

- Beispiel:

```
500  N1 = 3     /* FUNKTION "LESEN, ANZEIGEN, LOGGEN"
510  N2 = 0     /* STATUS LOESCHEN
520  S2$= "MESSAGES"
530  S1$= " ",S1$
540  N5 = 1     /* TABELLENBASIS
550  N6 = 55    /* NACHR. NUMMER
560  CALL 84,N1,N2,S1$,S2$,N5,N6
```

Das Programmfragment liest aus der Datei MESSAGES die Meldung "55 DIREKTAUSFUEHRUNG DES STATEMENTS NICHT MOEGLICH", zeigt sie in Zeile 24 an und trägt sie in das Logbuch ein.

Zu beachten ist, daß die Tabellenbasis zusammen mit der Fehlernummer den Fehlertext bestimmt. Wird als Tabellenbasis der Wert 1 und als Nachrichtennummer der Wert 50 genommen, wird dieselbe Meldung angezeigt, wie bei Tabellenbasis 2 und Meldung 40.

TAMOS - Schnittstellen

19.6 Common-Area

Jedem Arbeitsplatz wird ein spezieller Speicherbereich in der Größe von 512 Byte zugewiesen. Dieser Speicherbereich (Common Area) kann von jedem Port aus unter Angabe der Portnummer adressiert werden. Die Common Area dient verschiedenen Zwecken:

- Übergabe von Standardparametern durch TAMOS zum Anwendungsprogramm
- Datenaustausch zwischen sich gegenseitig aufrufenden Programmen oder zwischen verschiedenen Ports
- Ausgabe von Fehlermeldungen eines Programmes in einer Hintergrundpartition mit Hilfe des MAIL-Prozessors

Die Bearbeitung der Common Area erfolgt in BASIC durch die Unterprogramme

CALL 2: Datentransport in die Common-Area
CALL 3: Datentransport aus der Common-Area

Für den Datenaustausch stehen alle 512 Byte der COMMON-Area zur Verfügung. Die Informationen können in maximal 12 Datenfelder mit beliebiger Darstellung strukturiert werden. Die Abstimmung der Darstellung wird vom Programmierer vorgenommen. Die Adressierung und Unterscheidung der COMMON-Areas erfolgt durch die Portnummer.

19.6.1 Übergabe von Standardparametern durch TAMOS

TAMOS stellt dem Anwenderprogramm in der Common-Area an folgenden Positionen folgende Parameter als String bereit:

1-4	Reserviert für Fehlerauswertung
5-25	Beschreibung der 3. Selektorerbene
26-46	Beschreibung der 2. Selektorebene
47-66	Beschreibung der 1. Selektorebene
67-74	Datum (JJ.MM.TT)
75-89	der Text: NACHRICHT
90-95	Selektor-Parameter

TAMOS - Schnittstellen

96-113 (LU/), Programmname, Programmnummer
Die LU-Nummer wird nur eingetragen, wenn sie bei der Selektoranlage vorgegeben wurde. LU-Nummer und Programmname werden linksbündig abgestellt, die Programmnummer rechtsbündig. Die letzten Stellen des Programmnamens können durch die Programmnummer überdeckt werden.
114-121 Benutzte logische Einheiten

Der daraus zu erstellende Bildschirmaufbau ist wie folgt:

Bildschirm-Kopfzeile

1-21	Beschreibung der 3. Selektorebene
22-42	Beschreibung der 2. Selektorebene
43-66	Beschreibung der 1. Selektorebebene
68-70	Arbeitsplatznummer (SPC 6)
72-79	Datum (JJ.MM.TT)

Nachrichtenzeile

1-15	Text: NACHRICHT:
16-65	Nachrichtentext

Die Selektorparameter und die Programmnummer können benutzt werden, um Programmabläufe zu steuern.

Unter Jobspooling werden noch weitere Parameter übergeben. Sie werden an der entsprechenden Stelle beschrieben.

TAMOS - Schnittstellen

19.7 Fehlerbehandlung

Behandlung schwerwiegender, zum Abbruch führender Fehler im Anwenderprogramm.

Dieses Unterprogramm enthält die folgenden Funktionen:

- Rücksprung -1 ins Anwenderprogramm, wenn ein BASIC-Fehler > 97 auftritt, d.h. Fortsetzung des Programms an derselben Stelle, obwohl "ESC" oder "CTL-C" gedrückt wurde (SPC 8 = 99), bzw. Wiederholung der Eingabe bei nichtnumerischer Eingabe für eine numerische Variable (SPC 8 = 98) oder beim Lesen eines gesperrten Datensatzes (SPC 8 = 123).
- Eintragen der Fehlermeldung in die Log-Datei (SPC 8 <= 97)
- Aufruf des TAMOS-Programms TA.ABO

Die Meldung, daß das Programm abgebrochen wurde, wird von TA.ABO am Bildschirm angezeigt, wenn das Programm aus einem Selektor heraus angewählt wurde.

- Aufruf:

```
550 IF ERR 0 GOSUB 1000
```

- Unterprogramm:

```
1000 IF SPC 8>97 RETURN -1
1010 IF ERR 0 GOTO 1030
1020 LGERR      /*LOG-DATEI EINTRAG
1030 CHAIN "TA.ABO"
```

Die Zeile 1000 zeigt, wie mit der IF ERR 0 - Anweisung eine "ESC"- oder "CTL-C"-Eingabe abgefangen werden kann. Die Zeile 1010 stellt den "IF ERR 0 - Schalter" von Zeilennummer 1000 auf 1030 um. Damit wird verhindert, daß bei Auftreten eines erneuten BASIC-Fehlers während des LOG-DATEI-Eintrags eine Endlosschleife durch Rücksprung auf Zeile 1000 erzeugt wird.

19.8 Job-Spooler

Der Job-Spooler arbeitet Programme ab, die weder Tastatureingaben erfordern noch Bildschirmanzeigen vornehmen. Er läuft in einem systeminternen zusätzlichen "Arbeitsplatz" ohne Tastatur und Bildschirm ab - dem sog. Phantom Port.

Eine ausführliche Behandlung der Bedienung des Job-Spoolers finden Sie im "Bedienerhandbuch".

⚠ Der Phantom Port des Job-Spoolers hat die Portnummer 1

19.8.1 Job in die Jobspooldatei eintragen

Um ein Programm für die Verarbeitung unter Jobspooling vorzusehen, muß sein Name zusammen mit weiteren Parametern durch ein Eintrag-Programm in die Jobspooldatei (TF.SPOOLQUEUE) eingetragen werden. Einige dieser Parameter können aus der Datei TF.PORT übernommen werden.

Nach dem Start des Spoolers unter TAMOS prüft dieser durch Lesen der Eintragungen in der Jobspooldatei, ob Programme zur Verarbeitung anstehen. Ist dies der Fall, werden sie nacheinander verarbeitet. Der Spooler liest außer dem Programmnamen auch die weiteren vom Eintrag-Programm in die Jobspooldatei geschriebenen Parameter. Diese stellt er teilweise in der Common-Area ab, von wo sich das zu verarbeitende Programm die Parameter mit Hilfe eines CALL 3 abholt.

Eintragen eines Textdatei-Jobs

Ein Textdatei-Job ist ein Auftrag an den Spooler, eine Textdatei über eine Standard-Druckroutine auszudrucken.

Folgende Informationen müssen in der Jobspooldatei abgestellt werden:

- Job-Typ (=0 für Textdatei-Job)
- Job-Beschreibung (beliebiger Job-Name, der am Masterplatz erscheint, wenn die Jobspooldatei angezeigt wird).
- LU/Textdateiname
- Die vierstellige Sammelinformation (XYOZ) mit:

- Löschen des Jobs und Ändern der Anzahl Durchläufe
  X=0 : erlaubt
  X=1 : nicht erlaubt
- Job-Skip-Erlaubnis
  Y=0 : Bei Abbruch weiter mit dem nächsten Job
  Y=1 : Bei Abbruch Job-Spooler anhalten

- Löschcode
  Z=0 : Textdatei nach Ausdruck nicht löschen
  Z=1 : Textdatei nach Ausdruck löschen

- Anzahl Durchläufe (1 - 7999)
- Papiercode: 0 und 1 = Standardpapier
  2 bis n = kein Standardpapier
- Papierbeschreibung (Text zur Bedienerinformation bei Papierwechsel).
- Papierformat (Erste und letzte Druckposition sowie Zeilenanzahl pro Seite).
- Seitennummer der ersten zu druckenden Seite.
- Parameter aus der Datei TF.PORT:
  Job-Erstellungszeit
  RUN-Mode
  Teilnehmernummer
  Programmselektor-Koordinaten
- Freie Benutzerinformation

Diese Parameter sind Werte des Eintragprogramms, das den Job erstellt. Soll ein Parameter eingegeben werden, der von denen in TF.PORT abweicht, muß dieser entsprechend geändert werden (z.B. RUN-Mode).

Für Textdateijobs, die gegen Löschen geschützt sind, gelten bei der Bearbeitung mit dem Programm "ANZ. JOB-SPOOL-DATEI" folgende Einschränkungen:
- Löschen (DEL) ist nicht möglich.
- Ändern der Anzahl Durchläufe (ISS) nicht möglich
- Ist der Job mit einem Fehler- oder HALT-Status versehen, kann er nur freigegeben (REL) werden.

Eintragen eines Programm-Jobs

Die folgenden Informationen müssen in die Jobspooldatei überstellt werden:
- Job-Typ (=1 für Programm-Job)
- Job-Beschreibung (beliebiger Job-Name, der am Masterplatz erscheint, wenn die Jobspooldatei angezeigt wird)
- LU/Programmname
- Die vierstellige Sammelinformation (WXYZ) mit:

TAMOS - Schnittstellen

- Aufsetzerlaubnis ("REL"):
  W=0 : Job darf nach Abbruch wieder aufgesetzt werden
  W=1 : Job darf nach Abbruch nicht wieder aufgesetzt werden, der Job kann nicht gelöscht werden, die Anzahl Durchläufe kann nicht geändert werden.

- Job-Skip-Erlaubnis:
  X=0 : Bei Abbruch weiter mit dem nächsten Job
  X=1 : Bei Abbruch Job-Spooler anhalten

- Abbruch-Kontroll-Code (vergl. Program-Abort-Flag)
  Y=0 : Programm darf abgebrochen werden
  Y=1 : Bei Abbruch wird Rekonstruktion erforderlich

- Run-Mode-Kontroll-Code:
  Z=0 : Run-Mode aus Eintragprogramm übernehmen
  Z=1 : Run-Mode 0

- Anzahl Durchläufe
- Papiercode: 0 = Drucker wird nicht benötigt
  1 = Standardpapier
  >1 = kein Standardpapier
- Papierbeschreibung (frei für Benutzerinformation) *
- Papierformat (frei für Benutzerinformation) *
- Seitennummer
- Parameter aus der Datei TF.PORT:
  Job-Erstellungszeit RUN-Mode
  Teilnehmernummer Selektor-Koordinaten
- Freie Benutzerinformation

Diese Parameter sind Werte des Eintragprogramms, das den Job erstellt. Soll ein Parameter von den Angaben in der Datei TF.PORT abweichen, muß er entsprechend geändert werden (z.B. RUN-Mode).

Die Common-Area des Phantom-Port wird vom Jobspooler benutzt, um dem Job vor dem Start Parameter aus der Jobspooldatei zu übergeben.

Für Programme, die nicht wieder aufgesetzt werden dürfen und gegen Löschen geschützt sind, gelten bei der Bearbeitung mit dem Programm "ANZ. JOB-SPOOLDATEI" folgende Einschränkungen:

- Löschen ist nur bei gesetztem Fehler- oder HALT-Status erlaubt.
- Ändern der Anzahl Durchläufe nicht möglich.

TAMOS - Schnittstellen

- Benutzte Variablen:

M7$(55)	= Nachricht (Hilfsfeld)
D7$(20)	= Job-Beschreibung
P7$(18)	= LU/Text- oder Programmname
S7$(25)	= Papierbeschreibung
F7$(25)	= Beliebige Benutzerinformation
Z7$(160)	= Benutzerinformation
T(0)	= Arbeitsplatznummer (2%)
T(1)	= Fehlernummer (2%)
T(2)	= Job-Erstellungszeit (2%)
T(3)	= Teilnehmernummer (2%)
S(0)	= Job-Typ (1%)
S(1)	= RUN-Mode (1%)
S(2)	= Selektor-Koordinate der 1. Ebene (1%)
S(3)	= Selektor-Koordinate der 2. Ebene (1%)
S(4)	= Selektor-Koordinate der 3. Ebene (1%)
S(5)	= Löschcode und Job-Skip-Erlaubnis bei Textdatei-Jobs bzw. Aufsetzerlaubnis, Job-Skip-Erlaubnis, Abbruchcode und RUN-Mode bei Programmjobs (1%)
S(6)	= Anzahl der Durchläufe (1%)
S(7)	= Papiercode (1%)
S(8)	= Erste Druckposition (1%)
S(9)	= Letzte Druckposition (1%)
S(10)	= Zeilen pro Seite (1%)
S(11)	= Hilfsfeld (1%)
S(12)	= Seitennummer der 1. zu druckenden Seite (1%)

- Aufruf:

```
550 GOSUB  710                  /* UNTERPROGRAMMAUFRUF
560 IF T(1)=164 GOTO ...        /* SPOOL-DATEI VOLL
```

Anmerkung:

Programme können sich selbst in die Jobspooldatei eintragen. Dazu sollte die Routine, die die Parameter in die Jobspooldatei einstellt, als Unterprogramm des zu spoolenden Programms organisiert sein. Alle oben aufgeführten Variablen sind vor dem Aufruf des Unterprogramms zu dimensionieren. Die Variablen S(1), T(3), S(2), S(3), S(4) werden aus der Datei TF.PORT übernommen, S(1) ggfs. noch verändert. Den Variablen P7$, T(0), T(1), T(2), S(0) müssen festgelegte Werte zugewiesen werden, den Variablen S(5), S(6), S(7), S(8), S(9), S(10) und S(12) wählbare Parameter. Der durch die Variable Z7$ belegte Bereich (max. 160 Byte) kann dazu benutzt werden, weitere Parameter an das zu spoolende Programm zu übergeben. Die Variablen D7$, S7$, F7$ sind mit Kommentaren zu belegen, M7$ und S(11) dienen hier nur als Hilfsfelder.

TAMOS - Schnittstellen

Die verwendeten Parameterleisten haben folgenden Aufbau:

TF.PORT

Ab Displacement 22 + 44 * Portnummer

```
T(2)        = Job-Erstellungszeit (2%)
S(1)        = RUN-Mode (1%)
T(3)        = Teilnehmernummer (2%)
S(2)        = Selektorkoordinate der 1. Ebene (1%)
S(3)        = Selektorkoordinate der 2. Ebene (1%)
S(4)        = Selektorkoordinate der 3. Ebene (1%)
```

TF.SPOOLQUEUE

```
D7$         = Job-Beschreibung (20 Byte)
S(0)        = Job-Typ (1%)
T(2)        = Job-Erstellungszeit (2%)
S(1)        = RUN-Mode (1%)
T(3)        = Teilnehmernummer (2%)
S(2)        = Selektorkoordinate der 1. Ebene (1%)
S(3)        = Selektorkoordinate der 2. Ebene (1%)
S(4)        = Selektorkoordinate der 3. Ebene (1%)
P7$         = LU/Text- oder Programmname
S(5)        = Löschcode und Job-Skip-Erlaubnis bei Textda-
              tei-Jobs bzw. Aufsetzerlaubnis, Job-Skip-
              Erlaubnis, Abbruchcode und RUN-MODE bei Pro-
              grammjobs (1%)
S(6)        = Anzahl der Durchläufe (1%)
S(7)        = Papiercode (1%)
S(8)        = Erste Druckposition (1%)
S(9)        = Letzte Druckposition (1%)
S(10)       = Zeilen pro Seite (1%)
S(11)       = Hilfsfeld (1%)
S(12)       = Seitennummer der ersten zu druckenden Seite
              (1%)
F7$         = Beliebige Benutzerinformation (25 Byte)
Z7$         = Benutzerinformation (160 Byte)
```

TAMOS - Schnittstellen

Dem gespoolten Programm werden von TAMOS in der Common-Area folgende Parameter zur Verfügung gestellt:

```
H7$     = Kopfzeile (122 Byte)
F7$     = Freie Benutzerinformation (25 Byte)
S7$       Papierbeschreibung (25 Byte)
S(8)    = Erste Druckposition (1%)
S(9)    = Letzte Druckposition (1%)
S(10)   = Zeilen pro Seite (1%)
S(7)    = Papiercode (1%)
S(6)    = Anzahl der Durchläufe (1%)
S(12)   = Seitennummer der ersten zu druckenden Seite (1%)
Z7$     = Benutzerinformation (160 Byte)
```

Unterprogramm:

```
705 REM ***JOB IN SPOOL-DATEI EINTRAGEN***
710 OPEN #2, "TF.PORT"
720 READ #2,0,22+SPC6*44;T(2),S(1),T(3),S(2),S(3),S(4);
725 CLOSE #2
730 OPEN #2,"TF.SPOOLQUEUE"
750 FOR S(11)=0 TO CHF 2-1
760   READ #2,S(11);M7$
770   IF M7$(1,1)=" " GOTO 800 /* LEERER EINTRAG GEFUNDEN
780 NEXT S(11)
785 LET T(1)=164
795 GOTO 845
800 LET T(1)
810 LET S(11)=0
820 LET T(2)=(SPC 2-INT(SPC2/24)*24)*3600+INT(SPC3/10)
840 WRITE #2,-2;D7$,S(0),T(2),S(1),T(3),S(2),S(3),S(4),
    P7$,S(5),S(6),S(7),S7$,S(8),S(9),S(10),S(11),S(12),
    F7$,Z7$;
845 CLOSE #2
850 RETURN
```

Ein unter Jobspooling laufendes Programm kann vom Spooler in der COMMON-Area folgende Informationen Übernehmen:

```
H7$,F7$,S7$,S(8),S(9),S(10),S(7),S(6),S(12),Z7$
```

Ein negativer Wert in S(7) besagt, daß ein Papiercode-Wechsel vorliegt, bei Programm-Jobs also entsprechende Einricht-Routinen abgewickelt werden müssen.

Ein Programm ist zu beenden, indem es eines der folgenden Programme aufruft:

- TA.END
- TA.NCO
- TA.ABO

Weitere Logische Einheiten, auf die ein BASIC-Programm-Job zugreifen muß, werden aus dem Programm-Selektoreintrag des Mutter-Programms übernommen.

19.9 Parameter für den Druckspooler

Die Parameter für den Druckspooler werden in zwei Hauptgruppen übergeben:

- Formularbeschreibung
- Spoolersteuerung.

Systemintern wird ein String in der Länge von 34 Byte übergeben. Dabei belegt die Formularbeschreibung 9 Byte. Zur Spoolersteuerung werden 25 Byte benötigt.

19.9.1 Formularsteuerung

Die Parameterübergabe für die Formularsteuerung erfolgt in den ersten 9 Byte eines mindestens 34 Byte langen Parameterstrings direkt beim OPEN. Ist der String kürzer, werden die fehlenden Parameter durch Standardwerte ersetzt.

Neben der Übergabe aus einem Programm können die Parameter auch aus der TAMOS-Gerätezuordnungstabelle oder aus einer systeminternen Liste von Standardwerten übernommen werden.

Dabei werden die Parameter nach folgenden Prioritäten eingesetzt:

Erste/letzte Druckposititon	Zeilen pro Seite
1. Gerätezuordnungstabelle	1. Programm
2. Programm	2. Gerätezuordnungstabelle
3. Standardwerte	3. Standardwerte

TAMOS - Schnittstellen

19.9.2 Spoolersteuerung

Die Parameter für die Spoolersteuerung werden ab Byte 10 des Parameterstrings beim OPEN übergeben.

Aufbau der Parameter:

1 Byte Priorität	H high *N normal L low S suspended	Eintrag in die Spooldatei, aber Druck erst nach manuellen Umsetzen auf H,N,L
	Werden die Prioritäten mit kleinen Buchstaben angegeben, so kann der Auftrag nicht aus der Spooldatei gelöscht werden.	
1 Byte Druckverzögerung	Y Druck nach Schließen der Datei *N Druck während des Programmlaufs	
1 Byte Data Hold	Y Druckdaten verbleiben im Spooler, bis der Ausdruck durch den Bediener bestätigt wurde. *N Druckdaten werden sofort nach dem Drucken gelöscht.	
1 Byte Papier einrichten (nach Druck)	Y Einrichtroutine nach Druck der ersten Seite aufrufen *N keine Einrichtroutine	
1 Byte Papier einrichten (vor Druck)	Y Einrichtroutine vor Druck der ersten Seite aufrufen *N keine Einrichtroutine	
2 Byte Anzahl Kopien	nn Anzahl Kopien ohne Orginal (00 $\leq$ nn $\leq$ 99)	
2 Byte Papiercode	nn 00 $\leq$ nn $\leq$ 63 (keine Angaben = 00)	
16 Byte Jobbeschreibung	Beliebiger Text	

*: Der Wert ist standardmäßig eingestellt

19.10 Start von Programmen auf einem Phantomport

Programme, die weder Tastatureingaben erfordern noch Bildschirmausgaben vornehmen, können automatisch unter der Kontrolle des Jobspoolers gestartet werden. Sie müssen dazu mit der beschriebenen Routine in den Jobspooler eingetragen werden und werden standardmäßig auf dem Port #1, dem ersten Phantomport, gestartet.

Programme können ohne Kontrolle des Jobspoolers mit Hilfe des BASIC-CALL's 98 gestartet werden. Diese Routine bricht die Aktivitäten in dem angesprochenen Phantomport unkontrolliert ab. Abläufe sind durch den Programmierer zu verwalten.

Zum Starten eines Programmes per CALL-Anweisung ist es erforderlich, die Anweisung CALL 98 zweimal auszuführen. Bei der ersten Ausführung muß an den adressierten Port der Code "Ö" gesendet werden. Damit wird der adressierte Platz definiert abgemeldet.
Im Anschluß daran kann durch Übersendung eines Systemkommandos ein beliebiges Programm gestartet werden.

Anmerkung:

Nach Absetzen einer CALL 98-Anweisung muß 2 Sekunden gewartet werden (SIGNAL 3,20), bevor der Status in ⟨NVar2⟩ abgefragt wird.

- Beispiel:

  Starten des Programms FAKTURA am Phantomport (Port-Nummer 1).

```
:
100 LET S=0              /* ⟨NVar2⟩ AUF 0
110 LET A$= "Ö"          /* CODE FÜR ABMELDEN
115 CALL 98,1,A$,S       /* PORT#1 ABMELDEN
120 SIGNAL 3,20          /* 2 SEKUNDEN WARTEN
125 IF S GOTO ...        /* STATUSABFRAGE
130 LET A$="FAKTURA"     /* PROGRAMMNAMEN LADEN
135 CALL 98,1,A$,S       /* PROGRAMM STARTEN
140 SIGNAL 3,20          /* 2 SEKUNDEN WARTEN
145 IF S GOTO ...        /* STATUSABFRAGE
:
```

TAMOS - Schnittstellen

19.11 Meldungen aus einem Phantomport

Meldungen aus einem Phantomport können automatisch über den MAIL-Prozessor abgewickelt werden (vergl.: Datenaustausch über die Common-Area).
Sind nur Meldungen ohne Rückantwort abzusetzen, kann dies über den BASIC-CALL 4 erfolgen.

Das folgende Unterprogramm beinhaltet folgende Funktionen:

- Ausgabe einer Meldung in der Nachrichtenzeile des Masterplatzes.
- Warten auf eine Eingabe vom Masterplatz, die im Common-Bereich dem Phantom-Port übergeben wird.

- Benutzte Variablen:
  M7$(55) = Nachricht und Antwort
  T(0) = Arbeitsplatznummer (2%)
  T(1) = Nummer der Nachricht in TF.PARAM

- Aufruf:

```
550 LET T(1)=xx      /* NACHR.-NUMMER LADEN
560 GOSUB 1220       /* UNTERPROGRAMMAUFRUF
oder
550 LET M7$="Nachricht"
560 GOSUB 1230       /* UNTERPROGRAMMAUFRUF
```

- Unterprogramm:

```
1220 READ #1,INT(T(1)/10)+15,FRA(T(1)/10)*510;M7$;
1230 LET T(0) = -SPC6
1240 CALL 2,T(0),T(0),M7$       /* MELDUNG ÜBERGEBEN
1250 READ #1,14,137;M7$(39);
1260 LET M7$="←207←←376←←211←←376←←221←B←230
     M7$(39), "←376←←212←"
1270 IF ERR 0 GOTO 1330         /* FEHLERAUSWERTUNG
1280 CALL 4,M7$                 /* HINWEIS AM BILDSCHIRM
1290 SIGNAL 3,250               /* 25 SEKUNDEN WARTEN
1300 CALL 3,T(0),T(0),M7$       /* ANTWORT ABHOLEN
1310 IF T(0)=-SPC 6 GOTO 1250   /* KEINE ANTWORT
1320 RETURN                     /* RÜCKSPRUNG
1330 IF SPC8 <> 38 GOTO 1010    /* FEHLERBEHANDLUNG
1340 GOTO 1280
```

Dieses Unterprogramm zeigt im Abstand von 25 Sekunden am Masterplatz die Meldung JOB-SPOOLER aus. Nach Aufruf des MAIL-Prozessors mit der "S-Taste" wird die eigentliche Meldung angezeigt. Da das Unterprogramm die IF ERR 0-Routine benutzt, ist die Bedingung im Hauptprogramm neu zu setzen.

20 Programmwartung

Im vorliegenden Kapitel werden die im TAMOS-System zur Verfügung stehenden Dienstprogramme zur Wartung von BASIC-Programmen beschrieben. Sie finden diese Programme, wenn Sie im EXPERT-Selektor "BASIC" und "PROGRAMMIER-SYSTEM" anwählen.

```
T A M O S        PROGRAMMIERSYSTEM    BASIC              # 0 85.03.04/15:29


                 P R O G R A M M S E L E K T O R

                 1 ... PROGRAMM-ERSTELLUNG
                 2 ... PROGRAMM-AUSFUEHRUNG
                 3 ... PROGRAMM-LISTE
                 4 ... KOMMENTARE AUSLAGERN
                 5 ... KOMMENTARE EINLAGERN
                 6 ... PROGRAMM-VERGLEICH
                 7 ... PRUEFSUMMENLISTE
                 8 ... PROGRAMM-OPTIMIERUNG
                 9 ... PROGRAMM-ANALYSE

               NR WAEHLEN ODER "CR"  ........


NACHRICHT:
```

Programmwartung

20.1 Programmliste

```
PROGRAMM-LISTE      PROGRAMMIERSYSTEM    BASIC          # 0 85.03.04/15:38

      DATUM                  = 040385                LU-NR  DATEINAME
      (<CR> = SYSTEMDATUM)
      PROGRAMMIERER          = FISCHER                1  0   LAWI
                                                      2  0   FIBU1
      ZEILEN PRO SEITE       = 72                     3  0   FIBU2
      KOMMENTARE AUF LU-NR   = 0                      4  0   NK.TEST
                                                      5  0   DEBITOREN
                                                      6  0   AUFTRA01
      AUSGABE AUF DRUCKER    = Y                      7  N
      LU-NR AUSGABEDATEI     =                        8
                                                      9
      LU-NR DER LIBR-LISTE   = N                      10
      NAME DER LIBR-LISTE    =                        11
      LU-NR DER PROGRAMME    =                        12

                          OKAY ? = Y


NACHRICHT:
```

Nach Eingabe von BASIC-Programmnamen wird von diesen Programmen je eine Liste

- der Anweisungen
- der verwendeten Variablen einschließlich der Querbeziehungen
- der Zeilen-Querverweise in Sprungbefehlen und Unterprogrammaufrufen
- der Kanalnummer-Referenzen

auf dem Drucker oder in Textdateien ausgegeben.

Eingaben:

DATUM :
Hier kann ein bis zu sechsstelliges Datum eingegeben werden, das in dieser Form in der Kopfzeile des Ausdrucks erscheint. Eingabe CR = Systemdatum.

PROGRAMMIERER :
Ein beliebiger Text, bis zu 16 Zeichen lang, der den Programmierer kennzeichnet und ebenfalls in der Kopfzeile des Ausdrucks erscheint.

Programmwartung

ZEILEN PRO SEITE :
Seitenhöhe des Druckpapiers. (Mindestens 24 Zeilen).
Standard: Eintrag in Gerätezuordnung für $LPT.

KOMMENTARE AUF LU-NR.:
Optional: Nummer der logischen Einheit, auf die die Kommentare der anschließend einzugebenden Programme ausgelagert wurden. Die Kommentare erscheinen in der Programmliste. Sie werden aus den Textdateien des gleichen Namens, die aber mit dem Anfangsbuchstaben Y. beginnen, gelesen.

AUSGABE AUF DRUCKER :
LU-NR. AUSGABEDATEI :
Auswahl zwischen dem zugeordneten Drucker und Textdateien, die unter dem gleichen Programmdatei-Namen, aber mit dem Anfangsbuchstaben X. beginnend gespeichert werden. Deshalb dürfen die Namen der BASIC-Programmdateien bei Anwendung der Wartungsprogramme maximal 12 Zeichen lang sein.

LU-NR. DER LIBR-LISTE N
NAME DER LIBR-LISTE
"CR" = Keine LIBR-LISTE
Bei Angabe einer LU-Nr. wird der Name der mit LIBR erstellten Textdatei verlangt, die alle zu listenden BASIC-Programme (Typ "B" und "D") enthält.

LU-NR. DATEINAME
Wurde keine LIBR-Liste angegeben, ist nun an dieser Stelle die Angabe von bis zu 12 BASIC-Programmnamen mit zugehörigen LU-Nummern möglich. Nicht angemeldete LU's und Nicht-BASIC-Programmnamen werden abgewiesen.
"CR" = Sprung zur O.K.-Abfrage

Das Listen der Programme erfolgt in der Reihenfolge der Eingabe.

Nach dem Ausdrucken der Programmliste wird zur Selektoranzeige verzweigt.

Erläuterungen zum Ausdruck sowie zur Direktanwahl der Funktion unter BASIC finden Sie im Kapitel "BASIC-Kommandos", Abschnitt "PLIST".

Programmwartung

20.2 Kommentare Auslagern

```
KOMMENTARE AUSLAGERN PROGRAMMIERSYSTEM   BASIC          # 0 85.03.04/15:42

                                                  LU-NR  DATEINAME

    NEUE KOMMENTARE                                1   0  DEBITOREN
    AUF LU-NR              = 2                     2   0  KREDITOREN
    ALTE KOMMENTARE                                3   0  LAUI
    LÖSCHEN                = Y                     4   0  FIBU2
                                                   5   0  AUFTRA01
    LU-NR DER LIBR-LISTE = N                       6 N
    NAME DER LIBR-LISTE  =                         7
                                                   8
    LU-NR DER PROGRAMME  =                         9
                                                  10
                                                  11
                                                  12
                          OKAY ? = Y

NACHRICHT:
```

Das Programm dient zum Reduzieren von kommentierten BASIC-Programmen auf den reinen Programmteil, um den durch Kommentare erhöhten Speicher- und Ausführungszeitbedarf möglichst gering zu halten. Mit "KOMMENTARE EINLAGERN" können die Programme jederzeit wieder in die ursprüngliche, kommentierte Form gebracht werden.

Wenn Sie das Programm "Kommentare Auslagern" aus dem Expert-Selektor heraus anwählen, müssen Sie folgende Eingaben machen:

NEUE KOMMENTARE AUF LU-NR:
Die Nummer einer logischen Platteneinheit ist anzugeben, auf die die Kommentare ausgelagert werden sollen. Die Kommentare werden in Textdateien abgestellt, deren Dateiname aus "Y." und dem BASIC-Programmnamen zusammengesetzt wird.

- Beispiel:

  BASIC-Programmname : TEST
  Kommentar-Textdateiname : Y.TEST

Programmwartung

Da die Länge von Dateinamen im System auf 14 Zeichen beschränkt ist, ist darauf zu achten, daß die BASIC-Programmnamen 12 Zeichen nicht überschreiten.

Beantworten Sie danach die Frage

ALTE KOMMENTARE LÖSCHEN.

Geben Sie anschließend die

LU-NR DER LIBR-LISTE und
NAME DER LIBR-Liste

ein, in der Ihre Programmnamen aufgelistet sind. Mit 'CR' übergehen Sie diese Fragen. In diesem Fall können Sie in der Spalte

LU-NR DATEINAME

die BASIC-Programmnamen der Programme eingeben, die um ihre Kommentare reduziert werden sollen.

LU-Nummer

gibt die logische Magnetplatteneinheit an, auf der das Programm gespeichert ist. Namen, die keine BASIC-Programme bezeichnen und Namen von BASIC-Programmen, deren Kommentare bereits ausgelagert sind, werden mit einer Fehlermeldung zurückgewiesen. Maximal 12 Programmnamen können eingegeben werden. Bei Eingabe von "CR" anstelle einer LU-Nummer wird die Liste beendet.

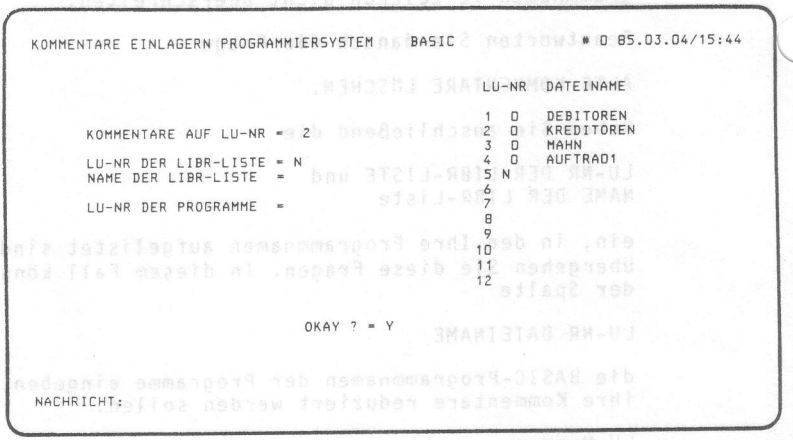
EINGABE OK

Y Starten des Auslagerns der Kommentare.
N Rücksprung zu "NEUE KOMMENTARE AUF LU-NR" und Korrekturmöglichkeit.
END Programmende ohne Auslagern der Kommentare.
PHA Der Job (das durch die Eingaben spezifizierte Programm) wird in die Job-Spooldatei eingetragen und vom Job-Spooler gestartet
PHD wie PHA - zusätzlich aber können noch Parameter für den Job-Spooler (Job-Beschreibung/Job-Skip-Erlaubnis und Papiercode) eingegeben werden.

Die Funktion "Kommentare Auslagern" kann für ein Programm auch mit dem BASIC-Kommando "UNCOMMENT" angewählt werden. Alle weiteren Erläuterungen zu dieser Funktion finden Sie im Kapitel "BASIC-Kommandos" unter "UNCOMMENT".

Programmwartung

20.3 Kommentare Einlagern



Durch "KOMMENTARE EINLAGERN" werden ausgelagerte Kommentare von BASIC-Programmen wieder in die jeweiligen Programme eingefügt, um die Übersichtlichkeit z.B. für Programmwartungszwecke zu erhöhen. Die Kommentare müssen vorher mit "KOMMENTARE AUSLAGERN" auf einer beliebigen logischen Einheit abgestellt worden sein.

Eingaben:

KOMMENTARE AUF LU-NR=
Die Nummer der logischen Einheit, auf der die einzulagernden Kommentare mit "KOMMENTARE AUSLAGERN" abgestellt worden sind. Die Kommentardateien werden nach erfolgreicher Ausführung gelöscht.

Programmwartung

Wenn die Namen der Programme, deren Kommentare wieder eingelagert werden sollen, in einer Bibliotheksliste aufgezeichnet sind, beantworten Sie anschließend die Frage

LU-NR DER LIBR-LISTE

mit Angabe der LU-Nummer, auf der Ihre Liste gespeichert ist. Ansonsten geben Sie "N" ein. Sie erhalten dann die Möglichkeit zur Eingabe der Namen der BASIC-Programme, deren Kommentierung wieder hinzugeladen werden soll, in der Form

LU-NR DATEINAME

Verfahren Sie hierbei analog zu "Kommentare auslagern". "CR" ohne Programmname beendet die Liste der Programmnamen.

EINGABE OK (Y/N/END) :
Y Das Laden der Kommentare beginnt.
N Rücksprung zu "KOMMENTARE AUF LU-NR" und Korrekturmöglichkeit.
END Programmende ohne Laden der Kommentare.
PHA/PHD s. "Kommentare Auslagern"

Die Funktion "Kommentare Einlagern" kann für ein Programm auch mit dem BASIC-Kommando "COMMENT" angewählt werden.

Alle weiteren Erläuterungen zu dieser Funktion finden Sie im Kapitel "BASIC-Kommandos" unter "COMMENT".

Programmwartung

20.4 Programmvergleich

```
PROGRAMM-VERGLEICH   PROGRAMMIERSYSTEM   BASIC          * 0 85.03.04/15:43


ENTER DATE OR END          : 03.05.85
LINES PER PAGE             : 72
OLD PROGRAM(S) ON LU.      : 2
NEW PROGRAM(S) ON LU.      : 0
PROGRAM NAME OR PART + '*'...: GL.

SELECT ON HASHTOTAL Y/N    : Y









NACHRICHT:
```

Das Programm bietet die Möglichkeit, verschiedene Entwicklungsstände von BASIC-Programmen (die sich auf unterschiedlichen logischen Platteneinheiten befinden müssen) durch das Auflisten der sich unterscheidenden Anweisungen auf dem Drucker darzustellen.

Eingaben:

ENTER DATE OR END
Ein sechs- bis achtstelliges Datum, das identisch zur Eingabeform in der Kopfzeile des Ausdrucks erscheint.

LINES PER PAGE
Zeilenhöhe des eingelegten Druckerpapiers (mindestens 20, höchstens 72).

OLD PROGRAM(S) ON LU
NEW PROGRAM(S) ON LU
Die Nummern der logischen Magnetplatteneinheiten, auf denen sich die alte bzw. neue Fassung der BASIC-Programme befinden.

Programmwartung

PROGRAM NAME OR PART+',' ,,,
Hier wird festgelegt, welche Programme verglichen werden sollen. Folgende Eingaben sind möglich: (Beispiel)

- einzelnes Programm : GL.MAHDRUS04
- Teilabschnitt eines Problemkreises : GL.MAHDRU,
- gesamter Problemkreis : GL.
- mehrere Problemkreise : GL.,FS.,LW.

Wird kein vollständiger Programmname eingegeben (mit "," abgeschlossen), werden alle Programme bearbeitet, deren Name wie eingegeben beginnt. Dasselbe trifft auch zu, wenn der eingegebene Programmname weniger als vier Zeichen lang ist (Problemkreis).

SELECT ON HASHTOTAL Y/N
Diese Frage ist nur nach der Eingabe von Problemkreisen zu beantworten. Im Dateikennsatz eines jeden BASIC-Programmes ist eine Prüfsumme (Hashtotal) gespeichert, die erkennen läßt, ob BASIC-Programme identisch sind (gleiche Prüfsummen) oder sich unterscheiden.

Y Sind die Prüfsummen alt und neu übereinstimmend, wird sofort mit dem nächsten Programm fortgefahren.

N Das Programm wird bei der Druckausgabe auch dann berücksichtigt, wenn die Prüfsummen alt und neu übereinstimmen.

Erläuterung des Ausdrucks:

REL x/xx
Genauer Freigabestand (RELEASE/LEVEL) (vgl. Anmerkung).

OLD HASHTOTAL
NEW HASHTOTAL
Prüfsumme der älteren bzw. neueren Programmversion.

REM 01
Eine Kommentarzeile eines jeden Basic-Standardprogramms muß einer bestimmten Form entsprechen und bei jeder neuen Version aktualisiert werden. (vgl. Anmerkung.) Wenn diese Zeile nicht in entsprechender Form vorhanden ist, erfolgt in der Liste eine Fehlermeldung.

Programmwartung

OLD-NEW
Auflistung der sich unterscheidenden, neu hinzugekommenen bzw. fortgefallenen Anweisungen des neuen Standes gegenüber der älteren Programmversion.

Nach Beendigung des Ausdruckens verzweigt das Programm zurück zu der Frage: "PROGRAM NAME OR PART+',' ,,,:".

Anmerkung:

BASIC-Programme können festgelegten Programmbereichen und darin unterschiedlichen Masterständen zugeordnet werden. Außerdem kann der Entwicklungsstand (Version) eines Programmes vermerkt werden. Diese Angaben werden von den Dienstprogrammen "PROGRAMMVERGLEICH" und "PRUEFSUMMEN-LISTE" respektiert, wenn sie in vorgeschriebenem Format in einer REM-Anweisung aufgeführt werden:

REM 01 "Programmname" PPVRRB

Dabei bedeuten:

PP : Programmbereich
Hier ist eine zweistellige Nummer zur Identifizierung des Programmbereichs anzugeben. Für Anwenderprogramme sind die Nummern von 51 bis 70 reserviert.

V : Von Master
Die Nummer des ersten Masterstands innerhalb des Programmbereichs, zu dem das Programm gehört.

RR : Release
Eine zweistellige Nummer zur Kennzeichnung der Programmversion.

B : Bis Master
Die Nummer des letzten Masterstandes innerhalb des Programmbereichs, zu dem das Programm gehört.

Programmwartung

20.5 Prüfsummenliste

```
PRUEFSUMMENLISTE     PROGRAMMIERSYSTEM    BASIC              # D 85.03.04/16:20

ENTER DATE OR END                  :  04.03.85
PROGRAM(S)  ON  LU.                :  0
ON PRINTER Y/N                     :  Y
LINES PER PAGE                     :  72
PROGRAM NAME OR PART + '*' * * *   :  ..................................

PROGRAM NAME          AREA     RELEASE      LEVEL          HASHTOTAL

00/KREDITOREN          07         0          00              44773
00/AUFTRAG1            07         0          00              63821
                                                             108594
NACHRICHT:
```

Das Programm gibt den Freigabestand und die Prüfsumme (Hashtotal) von BASIC-Programmen auf dem Bildschirm oder dem Drucker aus.

Eingaben:

ENTER DATE OR END
Ein sechs- bis achtstelliges Datum, das identisch zur Eingabeform in der Kopfzeile des Ausdrucks (bei Ausgabe auf dem Drucker) erscheint. Vorgegeben wird das Systemdatum in der Form TT.MM.JJ.

PROGRAM(S) ON LU
Nummer der logischen Platteneinheit, auf der sich die zu überprüfenden Programme befinden.

OUTPUT ON PRINTER Y/N
Y Ausgabe auf dem Drucker
N Ausgabe auf dem Bildschirm

Programmwartung

PROGRAM NAME OR PART+',' ,,,
Hier wird festgelegt, von welchen Programmen Freigabestand und Prüfsumme aufgelistet werden sollen. Folgende Eingaben sind möglich: (Beispiel)

- einzelnes Programm : GL.MAHDRUSO4
- Teilabschnitt eines Problemkreises : GL.MAHDRU,
- gesamter Problemkreis : GL.
- mehrere Problemkreise : GL.,FS.,LW.

Wird kein vollständiger Programmname eingegeben (mit "," abgeschlossen), werden alle Programme bearbeitet, deren Name wie eingegeben beginnt. Dasselbe gilt auch für den Fall, daß der eingegebene Name weniger als vier Zeichen lang ist.

Nach dieser Eingabe wird folgende Liste errechnet und auf dem Bildschirm bzw. dem Drucker ausgegeben:

PROGRAM NAME	AREA	RELEASE	LEVEL	HASHTOTAL
xx/GL.MOS1	07	4	01	xxxxx
xx/GL.MOS2	07	4	01	xxxxx
:	:	:	:	:
:	:	:	:	:
				xxxxxx

Erläuterung der Ausgabe:

PROGRAM NAME
Name des überprüften Programms in der Form

LU-Nummer/Programmname.

AREA
Programmbereich (vgl. "PROGRAMMVERGLEICH").

RELEASE/LEVEL
Freigabestand (von Master)/Version (Release, vgl. Anmerkung zu "PROGRAMMVERGLEICH").

HASHTOTAL
Prüfsumme des betreffenden Programms. Die Gesamtsumme aller Prüfsummen wird am Schluß der Liste angegeben.

Nach Ausgabe dieser Liste springt das Programm erneut zur Anzeige: "PROGRAM NAME OR PART+',' ,,,".

Programmwartung

20.6 Sichern unter anderem Filetyp (Programmoptimierung)

```
PROGRAMM-OPTIMIERUNG PROGRAMMIERSYSTEM    BASIC          # 0 85.03.04/15:46

          ZU ERZEUGENDER PROGRAMM CODE = 1
               1 ... OPTIMIERT
               2 ... STANDARD

          AUF LOGISCHER EINHEIT        = 0

          LU-NR DER LIBR-LISTE         = 0
           (N = GESAMTE LU)
          DATEINAME DER LIBR-LISTE     = NK.DUMMY

          LU-NR FEHLERPROTOKOLL        = 0
          DATEINAME FEHLERPROTOKOLL    = NK.FEHLER

                                OKAY ? = Y



NACHRICHT:
```

Mit Hilfe der über TAMOS anwählbaren "PROGRAMM-OPTIMIERUNG" kann eine Folge von BASIC-Programmen des Filetyps B als Files vom Typ D (laufzeitgünstiger) gesichert werden.

Eingaben:

ZU ERZEUGENDER PROGRAMM CODE = 1

Mit 'CR' übernehmen Sie die "1", d.h. Sie wünschen die Optimierung von Programmen

AUF LOGISCHER EINHEIT =

Sie geben nun die Nummer der Logischen Einheit an, auf der sich die zu optimierenden Programme befinden.

Programmwartung

LU-NR DER LIBR-LISTE =
(N = GESAMTE LU)
DATEINAME DER LIBR-LISTE =

Falls die Programme, die Sie optimieren möchten, in einer LIBR-Liste aufgeführt sind, müssen Sie nun die LU-Nummer dieser Liste angeben. Mit der Eingabe 'N' veranlassen Sie, daß alle Programme der oben eingegebenen Logischen Einheit optimiert werden. Nach der LU-Nummer der LIBR-Liste geben Sie bitte noch den Namen der Liste ein (DATEINAME).

LU-NR FEHLERPROTOKOLL =
DATEINAME FEHLERPROTOKOLL =

Hier ist die LU-Nummer und der Name einer Textdatei anzugeben, in die Meldungen über die Programmkonvertierungen abgestellt werden. Die Struktur der Meldungen ist:

⟨Programmname⟩		⟨Programmname⟩
	oder	
ERROR xx AT xxxxx		DIF.: +-xxxx WORDS

OKAY Y/N/PHA/PHD/END

Die Eingaben, die an dieser Stelle möglich sind, sind bereits bei "Kommentare Auslagern" am Anfang dieses Kapitels beschrieben.

Mit Hilfe der PROGRAMM-OPTIMIERUNG ist auch eine Konvertierung von D-Files in B-Files möglich, wenn Sie am Anfang des Programms "2" (STANDARD) statt der "1" eingeben.

Wenn Sie die Programmoptimierung für Ihre Programme einzeln durchführen wollen, können Sie das mit Hilfe des BASIC-Kommandos SAVE in der Form SAVE,D erreichen. Mit dem Kommando SAVE,B erhalten Sie wieder den Standardfile für Ihr Programm.

Programmwartung

20.7 Programm-Analyse

```
PROGRAMM-ANALYSE     PROGRAMMIERSYSTEM   BASIC              * 0 85.03.04/15:46

      DATUM                  = 040385              LU-NR  DATEINAME
      (<CR> = SYSTEMDATUM)
      PROGRAMMIERER          = SEIBT                1  0   LAWI
                                                    2  0   FIBU2
      ZEILEN PRO SEITE       = 72                   3  0   DEBITOREN
                                                    4  0   KREDITOREN
      AUSGABE AUF DRUCKER    = Y                    5  0   MAHN
      LU-NR AUSGABEDATEI     =                      6  N
                                                    7
                                                    8
      LU-NR DER LIBR-LISTE   = N                    9
      NAME DER LIBR-LISTE    =                     10
                                                   11
      LU-NR DER PROGRAMME    =                     12

                                OKAY ? = Y


NACHRICHT:
```

Die Programm-Analyse bietet eine Unterstützung bei der Umstellung von Programmen an.
Dieses Dienstprogramm analysiert Programme und markiert in einer Programmliste die Befehle und Variablen, die ungünstig für die Generierung von Variablenadressen sind.

Die Bedienung des Programms entspricht im wesentlichen der der Programmliste unter TAMOS. Die Abfrage KOMMENTARE AUF LU-NR entfällt.

Diese Funktion kann für ein Programm auch mit dem BASIC-Kommando ANALYSE angewählt werden (siehe Kap. "BASIC-Kommandos").

Nach dem Ausdrucken der Programmliste wird zur Selektoranzeige verzweigt.

Erläuterungen des Ausdrucks finden Sie im Kapitel "BASIC-Kommandos" unter "ANALYSE".

21 Programmierbeispiele

Dieses Kapitel enthält eine Sammlung von Programmierbeispielen zu folgenden Themenkreisen:

- Indexdateibearbeitung
- Eingabe über Tastatur mit CALL 1-Unterstützung
- Relative Datei auf Band kopieren
- Textdatei drucken
- Druckerdatei in Textdatei zwischenspeichern
- Größe einer Textdatei begrenzen
- Lochkartendatei einlesen
- Dateien-Übersicht erstellen
- Einbinden eines Programms in das TAMOS-System

21.1 Programmierbeispiele Indexdateibearbeitung

Es wird eine Indexdatei mit mehreren Schlüsselverzeichnissen von je 3 Worten Schlüssellänge und einer Datensatzlänge von 41 Worten bearbeitet. In allen Beispielen werden folgende Variablennamen verwendet:

```
C  = Kanalnummer
V  =  Verzeichnisnummer
V$ = Ordnungsbegriffe
N  = Relative Satznummer
S  = Parameter- und Statusvariable
```

Dimensionierung und Dateieröffung für alle Beispiele:

```
10 REM01 Programmname
20 DIM R$(80),B$(17)          /* DATENSATZLAENGE, HILFSFELD
30 DIM V$(6)                  /* SCHLUESSEL
40 DIM 1%,C,V,S,2%,N
50 IF  ERR 0 GOSUB ...        /* FEHLER BEI DATEIERÖFFNUNG
60 INPUT 'CS',TAB(10,5),"NAME DER INDEXDATEI:",
   TAB(35,5),B$
70 INPUT TAB(10,6),"ERÖFFNUNG AUF KANAL:",TAB(35,6),C
80 OPEN #C,B$
90 IF ERR 0 GOTO ...
```

Programmierbeispiele

21.1.1 Direktes Lesen eines Datensatzes (in Verzeichnis 1)

```
100 LET V=1
110 PRINT  TAB (10,9);"ORDNUNGSBEGRIFF:";
120 INPUT  TAB (35,9),V$
130 SEARCH #C,2,V;V$,N,S      /* BEGRIFF SUCHEN
140 IF S  GOTO 1500           /* STATUS ≠ 0
150 READ #C,N;R$
160                           /* DATENSATZ VERARBEITEN
 :
600 INPUT TAB(10,22),"NOCH EINMAL (Y/N)?",TAB(35,22),B$
610 IF B$="Y" GOTO 100
620 CHAIN ""
1500 IF S=1 PRINT TAB(10,20);"BEGRIFF NICHT VORHANDEN"
1510                          /* STATUSAUSWERTUNG
 :
1590 GOTO 600
```

21.1.2 Einfügen eines Ordnungsbegriffs mit zugehörigem Datensatz

```
100 INPUT 'CS',TAB(10,8),"VERZEICHNIS-NR.:",TAB(35,8),V
110 INPUT TAB(10,9),"SCHLÜSSELWORT:",TAB(35,9),V$
120 SEARCH #C,2,V;V$,N,S
130 IF S=0 GOTO 1560
135 IF S<>1 GOTO 1500
140 LET S=2
150 SEARCH #C,1,0;V$,N,S:
160 IF S GOTO 1500
170                           /* DATENSATZ AUFBEREITEN
 :
210 WRITE #C,N;R$
220 LET Z=N
230 SEARCH #C,4,V;V$,N,S      /* BEGRIFF EINTRAGEN
240 IF S  GOTO 1500           /* STATUS ≠ 0
250 INPUT TAB(10,22),"NOCH EINMAL (Y/N)?",TAB(35,22),B$
260 IF B$="Y" GOTO 100
270 CHAIN ""
1500 IF S=3 GOTO 1530
1510 IF S=1 GOTO 1560
1520 STOP          /*DATEI ZERSTÖRT ODER KEINE INDEXDATEI
1530 PRINT 'BEL'; TAB (10,20);"KEINE FREIEN SÄTZE!";
1540 SIGNAL 3,20
1550 GOTO 250
1560 PRINT 'BEL';TAB(10,20);"BEGRIFF BEREITS VORHANDEN!";
1570 GOTO 250
```

Programmierbeispiele

21.1.3 Ordnungsbegriff löschen und zugehörigen Datensatz freigeben

```
100 INPUT 'CS',TAB(10,8),"VERZEICHNIS-NR.:",TAB(35,8),V
110 INPUT TAB(10,9),"SCHLÜSSELWORT:",TAB(35,9),V$
120 SEARCH #C,5,V;V$,N,S:          /* BEGRIFF LÖSCHEN
130 IF S  GOTO 1500                /* STATUS ≠ 0
140 LET S=3
150 SEARCH #C,1,0;V$,N,S           /* SATZ FREIGEBEN
160 IF S STOP                      /* DATEI ZERSTÖRT
170 INPUT TAB(10,22),"NOCH EINMAL (Y/N)?",TAB(35,22),B$
180 IF B$="Y" GOTO 100
190 CHAIN ""
1500 IF S=1 GOTO 1520
1510 STOP
1520 PRINT 'BEL';TAB(10,20);"BEGRIFF NICHT VORHANDEN";
1530 GOTO 170
```

21.1.4 Sequentielle Bearbeitung ab dem nächstgrößeren Begriff

```
100 INPUT  TAB (10,8),"VERZEICHNIS-NR.:", TAB (35,8),V
110 INPUT  TAB (10,9),"SCHLÜSSELWORT:", TAB (35,9),V$
120 SEARCH #C,3,V;V$,N,S       /* GRÖSSEREN BEGRIFF SUCHEN
130 IF S GOTO 1500             /* STATUS ≠ 0
140 READ #C,N;R$               /* DATENSATZ LESEN
150                            /* DATENSATZ BEARBEITEN
 :
250 GOTO 120
1500 IF S<>2 STOP /*DATEI ZERSTÖRT ODER KEINE INDEXDATEI
1510 PRINT 'BEL'; TAB (10,20);"ENDE DES VERZEICHNISSES
     ERREICHT!";
1520 SIGNAL 3,20
1530 CHAIN ""
```

Programmierbeispiele

21.2 Beispiel einer Eingabe-Routine

Viele Anwendungsprogramme besitzen eingabeorientierte Teile, in denen durch entsprechende INPUT-Anweisungen die Eingaben alphanumerischer oder numerischer Daten vom BA gefordert wird, die dann von anderen Teilen des Programms verarbeitet werden. Mit Hilfe des CALL 1 läßt sich für solche Routinen ein optimaler Bedienungskomfort erreichen, der vor allem aus folgenden Möglichkeiten besteht:

- Aufbereiten eines Eingabefeldes
  - Festlegung der Position des Feldes
  - Begrenzung durch ein Hintergrund-Blank
  - Vorgabe von Werten
  - Eingabemaske in Form von Punkten
- Prüfung der Eingabe
  - auf vorgesehenen Typ (alphanumerisch, numerisch gepackt oder ungepackt)
  - auf zulässige Anzahl von Vor- und Nachkommastellen
  - auf Vorzeichen
- Vorgabe von Standardeingaben mit speziellen Aufgaben
  - zum Anzeigen der möglichen Eingaben (durch ?)
  - zur Übernahme vorhandener Werte (durch @)
  - zum Zurückspringen zu vorherigen Eingaben (durch ↑)
  - für weitere Spezialaufgaben (durch festzulegende 3-Zeichen-Kommandos)

Eine typische Eingabe besteht dann aus folgenden Schritten:

- Ausgabe eines Führungstextes für das Eingabefeld
- Festlegung der Parameter für die Aufbereitung des Eingabefeldes und eines eventuellen Vorgabewertes
- Festlegung der benutzbaren Sondereingaben
- Unterprogrammsprung:
  - CALL 1, Funktion 1: Eingabevorbereitung
  - Eingabe
  - CALL 1, Funktion 2: Eingabeprüfung:
    - falls Sondereingabe oder entsprechende Sonderbehandlungsroutine
    - falls korrekte Eingabe: Verarbeitung

Programmierbeispiele

Im folgenden Beispiel wird mit der Eingaberoutine eine Entfernungstabelle angelegt, die in eine formatierte Datei geschrieben wird, d.h. es wird je ein Städtename und eine zugehörige Entfernungsangabe in einen Datensatz geschrieben.

Die eigentliche Eingabe-Routine ist hier als ein Unterprogramm realisiert, das in dieser Form auch in andere Programme eingebunden werden kann. Das heißt insbesondere, daß einige Funktionen des Unterprogrammms durch das vorliegende Beispiel weder genutzt noch geprüft werden.

Nach der Eingabe werden verschiedene Prüfungen vorgenommen, die zu den Fehlermeldungen 170 bis 176 in der Datei TF.PARAM führen können.

Andere Programmteile (Fehlerbehandlung, TAMOS-Schnittstellenroutine u.a.) werden hier nicht dargestellt.

```
 10 DIM S1$(13),S2$(21),S3$(6),S4$(4),S5$(18),T2$(21)
 20 DIM S$(21),1%,N1,N2,I,E
 30 OPEN #1,"TF.PARAM"
 31 REM ---- TF.PARAM enthält Fehlermeldungen für CALL 1
 40 OPEN #2,"ENTF"
 41 REM ---- in ENTF wird die Entfernungstabelle
 42 REM ---- gespeichert.
 50 REM 50-99: Positionierung der Dateizeiger
100 LET S3$="CANEND"
101 REM ---- S3$ gibt mögliche Sondereingaben an.
102 REM ---- hier: CAN und END
110 PRINT 'CS';'SB'; TAB (10,5);"STADT:"; TAB (10,7);
    "ENTFERNUNG:";
111 REM ---- Führungstexte für Eingaben
120 REM ********** STADTNAMEN EINGEBEN ******************
130 LET S1$="2125052102000"
131 REM ---- S1$: Parameterstring für CALL 1
132 REM ---- 2-1-25-05-21-02-0-0-0 bedeutet:
133 REM ---- 2 : Eingabemaske aus Punkten mit 'CF'
134 REM ---- 1 : alphanumerische Eingabe
135 REM ---- 25-05 : Anfangsposition des Eingabefeldes
136 REM ---- 21-02 : maximale und minimale Anzahl Zeichen
137 REM ---- 0 : keine Nachkommastellen
138 REM ---- 0 : keine Vorzeichen
139 REM ---- 0 : 3-Zeichen-Eingabe möglich
140 LET S4$="1100"
141 REM ---- S4$ : Auswahlstring für CALL 1
142 REM ---- 1-1-0-0 bedeutet:
143 REM ---- 1 : Eingabe @ nicht erlaubt
144 REM ---- 1 : Eingabe ↑ nicht erlaubt
```

Programmierbeispiele

```
145 REM ---- 0 : Eingabe des 1. 3-Zeichen-Kommandos
146 REM ----     (hier: CAN) erlaubt
147 REM ---- 0 : Eingabe des 2. 3-Zeichen-Kommandos
148 REM ----     (hier: END) erlaubt
150 GOSUB 9000  /*            --> Eingabe mit Prüfung
160 ON N1+1 GOTO  180, 130, 130, 400, 500
161 REM ---- Reaktion auf Ergebnis der Eingabeprüfung:
162 REM ---- N1+1 = ... bedeutet:
163 REM ---- ...1 : korrekte Eingabe (keine Sondereingab)
164 REM ---- ...2 : @ eingegeben (hier ausgeschlossen)
165 REM ---- ...3 : ↑ eingegeben (hier ausgeschlossen)
166 REM ---- ...4 : CAN eingegeben: Eintrag löschen
167 REM ---- ...5 : END eingegeben: Programmbeendigung
170 LET S$=S2$
171 REM ---- S$ : speichert Stadtnamen zum späteren
172 REM ----       schreiben in die Datei ENTF
180 REM ********** ENTFERNUNG EINGEBEN ******************
190 LET S1$="1225070401000"
191 REM ---- Punkt-Eingabemaske (1), numerische Eingabe
192 REM ---- (2), Anfangsposition (25,07), Anzahl Vorkom-
193 REM ---- mastellen (min 01, max 04), Anzahl Nachkom-
194 REM ---- mastellen (), kein Vorzeichen (0),
195 REM ---- 3-Zeichen-Eingabe möglich (0)
200 LET S4$="1001"
201 REM ---- ↑ und CAN erlaubt, @ und END nicht
210 GOSUB 9000  /*            --> Eingabe mit Prüfung
220 ON 11+1 GOTO  230, 190, 130, 450, 190
230 LET E=N2
240 REM ********* DATENSATZ SCHREIBEN ******************
250 WRITE #2;E,S$;
260 GOTO  130  /*             --> neue Eingabe
350 REM ********** SONDEREINGABEN **********************
400 REM 400-449: Reaktion auf Eingabe CAN bei Stadtname
450 REM 450-499: Reaktion auf Eingabe CAN bei Entfernung
500 REM 500-549: Reaktion auf Eingabe END bei Stadtname
9000 REM ********** EINGABE - ROUTINE ******************
9010 LET T2$=S2$        /* Retten eines vorhandenen Wertes
9020 LET N1=0
9030 CALL 1,N1,S1$,S2$,N2,S5$      /* Eingabevorbereitung
9040 INPUT 'BP', S2$             /* Eingabe auf Feldanfang
9050 IF S2$="?" GOSUB 9200       /* mögliche Eingaben ?
9055 IF S2$="?" GOTO 9040        /* erneute Eingabe
9060 IF S1$(2,2)="1" LET N2= LEN S2$
9070 CALL 1,N1,S1$,S2$,N2,S3$,S4$,S5$ /* Eingabeprüfung
9080 IF N1=0 RETURN                   /* korrekte Eingabe
9090 IF N1>1 RETURN   /* erlaubte Sondereingabe (außer @)
9100 IF N1=1 GOTO 9150       /* @ erlaubt und eingegeben
9110 GOTO 9020               /* N1=-1 : unkorrekte Eingabe
9150 LET N1=0             /* Verarbeitung der Eingabe @ :
9160 LET S1$(1,1)="3"
```

```
9170 LET S2$=T2$          /* geretteten Wert zurückholen
9180 CALL 1,N1,S1$,S2$,N2,S5$          /* und prüfen
9190 RETURN
9200 PRINT  TAB (17,24);'CFF'; TAB (17,24);
9210 IF S4$(1,1)="0" PRINT " @ /";  /* Display der
9220 IF S4$(2,2)="0" PRINT " ▌ /";  /* möglichen Eingaben
9230 FOR I=3 TO  LEN S4$            /* in Zeile 24
9240   IF S4$(I,I)="0" PRINT S3$(I*3-8,I*3-6);"/";
9250 NEXT I
9270 PRINT 'BS';'BP';
9280 RETURN
```

21.3 Relative Datei auf Magnetband kopieren

Folgende Variablen werden benutzt:

- U = Gerätenummer Magnetband
- F = Funktion
- S = Status
- C = Block-/Satzlänge
- A = Aktuelle (geschriebene) Länge
- R$ = Datensatz
- D$ = Name Plattendatei

Ausgabe erfolgt ohne Codeumwandlung.

Programmierbeispiele

```
10 IF ERR 0 GOSUB 330
20 DIM D$(20)
30 PRINT 'CS';TAB(0,24);"RELATIVE DATEI AUF MB KOPIEREN";
40 INPUT 'CR',"DATEINAME:", TAB (30),D$ /* RELATIVE DATEI
50 INPUT 'CR',"SATZLÄNGE:", TAB (30),C
60 IF C>2048 GOTO 50                    /* MAX. SATZLÄNGE=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
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 BEIM SCHREIBEN
180 LET R$=" ",R$                       /* STRING LÖSCHEN
190 GOTO 150                            /* NÄCHSTEN PLATTTENSATZ
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 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 FALSCHEINGABE
340 IF SPC 10<110 GOTO 360              /* NOCH KEIN BANDOPEN
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
```

Programmierbeispiele

21.4 Textdatei drucken

Wird als Ausgabegerät ein Zeilendrucker benutzt, werden die Eingaben "ERSTE DRUCKPOSITION" und "LETZTE DRUCKPOSITION" ignoriert.

```
 10 REM ************************************************
 11 REM ************** MASKE ERSTELLEN ******************
 12 REM ************************************************
 20 PRINT 'CS'; TAB (10);"PROGRAMM : Textdatei drucken"
 30 PRINT TAB (0,2); "Übernahme von Druckparametern
    durch CR";
 40 PRINT TAB (10,5); "DATEINAME"; TAB (40); ":";
    'CR'; TAB (10); "STRINGLÄNGE"; TAB (40); ":";
    'CR'; TAB (10); "DRUCKERNAME"; TAB (40); ":"
 50 PRINT TAB (10); "ERSTE DRUCKPOSITION"; TAB (40); ":";
    'CR'; TAB (10); "LETZTE DRUCKPOSITION";TAB (40); ":";
    'CR'; TAB (10); "BLATTHÖHE"; TAB(40); ":"
 60 REM ************************************************
 61 REM ************** EINGABE MIT PRÜFUNG **************
 62 REM ************************************************
 70 IF  ERR 0 GOSUB  590
 80 DIM D$(20), P$(3), P1$(9)
100 LET P1$="0",P1$
110 INPUT  TAB (42,5),D$  /*     EINGABE: Textdateiname
120 OPEN #2,D$
130 INPUT  TAB (42,6),N  /*      EINGABE: Stringlänge
140 IF FRA N GOTO  110
150 IF N<0 GOTO 110
160 IF N>999 GOTO 110
170 DIM A$(N)
180 INPUT  TAB (42,7),D$  /*     EINGABE: Druckername
190 INPUT  TAB (42,8),P$  /*     EINGABE: 1. Druckposition
200 CALL 62,P$
210 IF P$="" LET P$="000"
220 LET P1$(4- LEN P$,3)=P$
230 INPUT  TAB (42,9),P$  /*     EINGABE: letzte Druckpos.
240 CALL 62,P$
250 IF P$="" LET P$="131"
260 LET P1$(7- LEN PS,6)=P$
270 IF P1$(1,3) =P1$(4,6) GOTO  190
280 INPUT  TAB (42,10),P$  /*    EINGABE: Blatthöhe
290 CALL 62,P$
300 IF  LEN P$=1 GOTO 280
310 IF P$="" LET PS="072"
320 LET P1$(10- LEN P$,9)=P$
330 REM ***********************************************
331 REM ************* VERARBEITUNG **********************
332 REM ***********************************************
340 OPEN #3;P1$,D$
```

Programmierbeispiele

```
350 DIM B$(1)
351 REM ---- B$ : Hilfsvariable für Test auf Dateiende
360 LET X=0
370 LET Y=0
380 READ #2,X,Y;A$
381 REM ---- Position des Dateizeigers vor READ # :
382 REM ---- Block X, Byte Y
390 PRINT #3;A$;
400 LET X1= CHF 102
410 LET Y1= CHF 202
411 REM ---- Position des Dateizeigers nach READ # :
412 REM ---- Block X1, Byte Y1
413 REM ---- X = X1 : Dateizeiger nach Lesevorgang noch
414 REM ----           in altem Block
415 REM ---- X < X1 : Sprung über Blockgrenze
420 IF X=X1 LET D=Y1-Y
430 IF X<X1 LET D=((X1-X)*512)+(Y1-Y)
431 REM ---- D : Länge des Sprungs des Dateizeigers
432 REM ---- D = LEN A$ oder D = LEN A$ + 1
440 IF D= LEN A$ GOTO 510
450 REM ---- D = LEN A$ + 1 : Dateizeiger muß um
451 REM ---- 1 Zeichen zurückpositioniert werden
452 LET Y=Y1-1
460 LET X=X1
461 REM ---- X und Y für direktes Lesen setzen
462 REM ---- Y >= 0 : Rückpositionierung korrekt ausge-
463 REM ----           führt, Dateizeiger blieb im Block
470 IF Y>=0 GOTO  380
471 REM ---- Y < 0  : Rückpositionierung nicht korrekt
472 REM ----           ausgeführt, Dateizeiger auf letztes
473 REM ----           Byte des vorhergehenden Blocks
474 REM ----           positionieren
480 LET Y=512
490 LET X=X1-1
500 GOTO  380
510 REM ---- D = LEN A$ : Beendigung der Zeichenüber-
511 REM ---- tragung für READ # - Anweisung durch
512 REM ---- Erreichen eines CR-zeichens (dann weiter-
513 REM ---- lesen), oder Dateiende erreicht (dann
514 REM ---- Programmende)
515 READ #2;B$
516 REM ---- B$ = "" : Dateiende ereicht
517 REM ---- B$ ≠ "" : weiterlesen ! X und Y für direktes
518 REM ----           Lesen setzen
520 IF B$="" GOTO 560
530 LET X=X1
540 LET Y=Y1
550 GOTO  380
560 CLOSE #2,#3
570 END
```

Programmierbeispiele

```
580 REM ************************************************
581 REM ************* FEHLERBEHANDLUNG *****************
582 REM ************************************************
590 IF  SPC 8=38 RETURN -2
591 REM ---- SPC 8 = 38 : CALL 62 hat fehlerhafte
592 REM ---- Eingabe erkannt: Neueingabe ermöglichen
600 IF  ERR 0
610 PRINT  TAB (0,20);"BASIC-FEHLER NR."; SPC 8;
    "IN ZEILE NR."; SPC 10
620 IF  SPC 10>100 CLOSE #2
630 IF  SPC 10>340 CLOSE #3
640 PRINT "PROGRAMMENDE"
```

21.5 Druckerdatei in Textdatei zwischenspeichern

Wird in dem Programm des vorherigen Beispiels bei der Druckeröffnung ein Fehler festgestellt, wird eine Textdatei erstellt und die zu druckenden Sätze werden druckfertig in der Textdatei abgestellt.

```
        MASKE ERSTELLEN und
        EINGABE MIT PRÜFUNG (Zeilen 10-320)
        wie im vorherigen Beispiel

330 REM ************************************************
331 REM ************* VERARBEITUNG *********************
332 REM ************************************************
335 IF ERR 0 GOTO          /* Fehler bei Druckereröffnung
340 OPEN #3;P1$,D$         /* Druckereröffnung
345 IF ERR 0 GOSUB 590     /* alte Fehlerbehandlung

        weitere VERARBEITUNG
        und FEHLERBEHANDLUNG (Zeilen 350-650)
        wie im vorherigen Beispiel

660 REM ************************************************
661 REM **** FEHLERBEHANDLUNG BEI DRUCKERERÖFFNUNG ******
662 REM ************************************************
670 IF ERR 0 GOSUB 590
680 INPUT  CS , TAB(10,5), "NAME DER TEXTDATEI: ",D$
690 LET D$(LEN D$+1)="!"/* vorhandene Datei überschreiben
700 IF ERR 0 GOTO 740    /* Fehler bei Textdateieröffnung
710 BUILD #3,D$          /* Textdateieröffnung
720 IF ERR 0 GOSUB 590
730 GOTO 350
740 PRINT TAB(15,24);"NAME VON ANDEREM KONTO BELEGT";
750 GOTO 670
```

Programmierbeispiele

21.6 Beipiel zur CHF-Funktion

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.

```
10 REM01 Programmname
20 DIM M$(50)
30                                  /* AUFBEREITUNG VON M$
 :
100 GOSUB 9000                      /* AUFRUF LOG-DATEI-EINTR.
 :
9000 PRINT #3,CHF103,CHF203;M$ /* DIREKT SCHREIBEN
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
```

21.7 Lochkartendatei in relative Datei einlesen

Übernahme einer Lochkartendatei in eine relative Datei auf einer Platte. Als Endkriterium für die Lochkartendatei ist eine /*-Karte vorgelegt.

```
10 DIM D$(17),R$(80)
20 INPUT 'CR',"RELATIVE DATEI : ",D$
30 OPEN #3,"$CRD1"              /* LOCHKARTENDATEI
40 OPEN #4,D$                   /* RELATIVE DATEI
50 READ #3,R$                   /* OCHKARTE LESEN
60 IF R$="/*" GOTO 100          /* DATEIENDE
70 WRITE #4;R$                  /* DATENSATZ SCHREIBEN
80 GOTO 50                      /* NÄCHSTE KARTE
 :
100 CLOSE #3                    /* LOCHKARTENDATEI
110 CLOSE #4                    /* RELATIVE DATEI
120 PRINT "PROGRAMMENDE";
130 CHAIN ""                    /* CHAIN SCOPE
```

Programmierbeispiele

21.8 CALL 97-Anwendung

Das Programm erstellt eine Liste sämtlicher Dateien einschließlich Dateieigenschaften der Sytemplatte.

Erläuterung der vom Programm ausgedruckten Werte:

```
FILENAME: Dateiname
SIZE    : Dateigröße in Blöcken (Sektoren)
USER    : Benutzernummer
AGE     : Vergangene Zeit in Stunden seit der Dateier-
          stellung
HSLA    : Vergangene Zeit in Stunden seit der letzten
          Dateieröffnung
TYPE    : Dateiart
HBA     : Plattenadresse des Dateikennsatzes (oktal)
```

```
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$(108),L$(125),H1$(108)
110 REM
120 REM  ********** ÜBERSCHRIFT AUFBEREITEN **********
130 REM
150 LET H$=" ",H$
160 LET H$(1,8)="FILENAME"
190 LET H$(38,41)="SIZE"
200 LET H$(49,55)="USER"
210 LET H$(63,65)="AGE"
220 LET H$(73,76)="HLSA"
230 LET H$(85,88)="TYPE"
250 LET H$(106,108)="HBA"
260 LET H1$="-",H1$
270 OPEN #2,"$LPT"
280 REM  ********** ÜBERSCHRIFT DRUCKEN **********
290 LET P9=P9+1
300 PRINT #2;"CALL 97 BEISPIEL";TAB(90);"SEITE:";P9;
    "←215←"
310 PRINT #2;"←215←←215←"
320 PRINT #2;H$
330 PRINT #2;H1$
340 LET Z1=0
350 CALL 97,U,R,F$,A,T,S,Q,C,I,D,L,H
```

Programmierbeispiele

```
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   **********
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
580 REM  ********** HOLEN KONTOBENUTZER-NR.
610 LET U1=A
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$(1,LEN F$)=F$
740 LET L$(38,41)=S USING "####"
770 LET L$(53,54)=U1 USING "##"
780 LET L$(61,65)=D USING "#####"
790 LET L$(72,76)=L USING "#####"
800 LET L$(85,86)=A$
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 ""
```

Programmierbeispiele

21.9 Standard-Basisteil für Anwenderprogramme im TAMOS-System

21.9.1 Einführung und Handhabung

Dieser Standard-Basisteil kann jedem individuellen Anwenderprogramm vorangestellt werden. Er umfaßt folgende Routinen:

- TAMOS-Kopfzeile übernehmen und am Bildschirm ausgeben
- TAMOS-Nachrichtenausgabe
- TAMOS-Loggingroutine
- Standard-Fehler-Behandlung
- Einfüge-Routine für Indexdateien
- INPUT-Routine für Eingabeaufbereitung und Prüfung
- TAMOS-Job in Jobdatei eintragen
- TAMOS-Kommunikation Phantom zu Master
- Druckerfehlerroutine
- Papierwechsel unter Spooler
- Übernahme des TAMOS-Bereiches beim Spooler mit Druckereröffnung

Damit dieser Basisteil universell verwendbar wird, sollte er nach der ersten Eingabe und Überprüfung mit dem BASIC-Kommando DUMP in einer Textdatei auf der Systemplatte (logische Einheit 0) abgestellt werden.

Anwenderprogramme, denen der Basisteil vorangestellt werden soll, müssen mit Zeilennummer 1660 oder größer beginnen. Mit BASIC-Kommando LOAD ist der Basisteil einzubinden.

Die Numerierung beginnt mit 10 in 10er Sprüngen. Das erlaubt jederzeit eine Neunumerierung der mit dieser Basis erstellten Programme, ohne daß sich die Ansprung-Adressen in dem Basisteil verändern.

Es wird davon ausgegangen, daß in dem Basisteil außer den Zeilen 10 und 400 (eventuell auch 90) keine Veränderungen vorgenommen werden. In der Zeile 10 ist mit "REM01" der Programmname, Fachbereich, Release usw. einzutragen.

Programmierbeispiele

Muß aus Kapazitätsgründen trotzdem die Basis verkleinert werden, so sollte folgendermaßen vorgegangen werden:

- Von hinten nach vorn sollten Teile gelöscht werden, falls diese nicht benötigt werden und zusammenhängend sind:

```
1310 DELETE 1630    wenn das Programm kein Druckprogramm
                    ist.

1170 DELETE 1630    wenn das Programm nicht unter dem
                    Spooler ablaufen kann und kein
                    Druckprogramm ist.

 990 DELETE 1630    wenn mit diesem Programm kein Job in
                    die Spool-Datei eingetragen wird und
                    die Bedingungen wie bei 1160 erfüllt
                    sind.

 590 DELETE 1630    wenn der "CALL 1" nicht benötigt
                    wird und die Bedingungen wie bei 980
                    erfüllt sind.

 360 DELETE 1630    wenn keine Einfüge-Routine in eine
                    Indexdatei benötigt wird und die Be-
                    dingungen wie bei 590 erfüllt sind.
```

- Müssen ausnahmsweise Teile zwischen zwei Routinen gelöscht werden, so sollten diese Zeilen nicht gelöscht, sondern mit REM versehen werden.

Beim Einsatz der LINK-Technik sollten die DIM-Anweisungen (20-60) des zu linkenden Programmes ebenfalls mit REM versehen werden. Durch diese Vorgehensweise wird erreicht, daß die Sprungadressen in dem Basisteil immer auf die gleichen Zeilennummern verweisen.

Wird mit Drucker gearbeitet, so ist dieser auf Kanal 12 zu benutzen.
Beim Eintragen eines Jobs in die Spooldatei wird vom Basisteil vorübergehend Kanal 13 benutzt.

Programmierbeispiele

21.9.2 Beschreibung der einzelnen Routinen

Vor der ersten Routine werden die Variablen dimensioniert, die der Basisteil zusätzlich benötigt.
Wird der Bereich 990 bis 1630 gelöscht, können die Zeilen 40 bis 60 mit REM versehen werden.

TAMOS-Kopf-und-Nachrichtenzeile aufbauen

Diese Routine wird automatisch in Zeile 1650 aufgerufen. Sollen Release und Level verändert werden, ist dies in Zeile 90 vorzunehmen.

TAMOS-Nachrichtenausgabe

Aufruf zur Ausgabe einer Nachricht aus TF.PARAM

```
... GOSUB 150
Vorgabe: T(1)  = Nachrichtennummer (2%)
```

Aufruf zur Ausgabe einer Nachricht mit eigenem Text

```
... GOSUB 160
Vorgabe: M7$   = Nachricht
```

TAMOS-Logging

Aufruf:

```
... GOSUB 200
Vorgabe: M7$   = Eintrag-Text
```

TAMOS-Standard-Fehlerbehandlung

Aufruf:

```
... IF ERR 0 GOSUB 280
Keine Vorgaben.
```

Einfügen-Routine in Indexdatei

Diese Routine ermöglicht das Einfügen eines Datensatzes und des zugehörigen Ordnungsbegriffs in Verzeichnis 1.

Aufruf:

```
... GOSUB 370
Vorgaben: I2   = Kanalnummer
          M7$  = Ordnungsbegriff
```

Programmierbeispiele

Die Routine "Datensatz schreiben" muß individuell programmiert werden. Hierzu ist in Anweisung 400 die Kommentarkennung REM zu löschen und die erste Zeilennummer der Routine "Datensatz schreiben" einzutragen. Die relative Satznummer wird in T(0) übergeben.

Rücksprung aus der Einfüge-Routine:

- Satz richtig eingefügt: RETURN 1
- Datei voll : Log-Datei-Eintrag, Nachrichtenausgabe und RETURN
- Satz bereits vorhanden: Nachrichtenausgabe und RETURN
- Dateifehler : Nachrichtenausgabe, Log-Datei-Eintrag und Programmabbruch

INPUT-Routine zur Eingabeaufbereitung und -Prüfung

Diese Routine ermöglicht die Eingabe numerischer und alphanumerischer Werte. Die Bildschirmaufbereitung wird je nach Vorgabe des Kommandostrings vorgenommen.
Bei Eingabe eines ? werden alle Eingabemöglichkeiten an Steuerzeichen und -Symbolen am Bildschirm angezeigt.

Wird "Packen" im Kommandostring gefordert, wird es von dieser Routine mitübernommen.
Wird " " (Duplizieren) eingegeben, wird der übernommene Wert am Bildschirm angezeigt.

Aufruf:

```
... GOSUB 600
Vorgaben: M7$ = zu duplizierender alphanumerischer Wert
          IO  = zu duplizierender numerischer Wert
          I$  = Kommandosting
          I2$ = Zulässigkeitskennzeichen für Steuerinformationscodes
          I1$= Steuerinformationscodes (je 3 Zeichen)
```

Ergebnis:

```
IO$ = eingegebener Wert
IO  = eingegebener Wert numerisch
I1  = Steuerfunktionsnummer
```

Bei Eingabe einer 3-Byte-Steuerinformation enthält IO$ allerdings diese Steuerinformation.

I1$, I2$ und I$ werden nicht verändert. Sie brauchen deshalb bei mehreren Eingaben nur gezielt geändert zu werden.

Programmierbeispiele

TAMOS-Job in Spool-Datei eintragen

Aufruf:

```
   ... GOSUB 1000
   Vorgaben: D7$  = Job-Beschreibung
             P7$  = LU/Text- oder Programmname
             F7$  = Beliebige Benutzerinformation
             S7$  = Papierbeschreibung
             S(0) = Job-Typ
             S(5) = Löschcode bzw. RUN-mode und Job-Skip-
                    Erlaubnis
             S(6) = Anzahl Durchläufe
             S(7) = Papiercode
             S(8) = Erste Druckposition
             S(9) = Letzte Druckposition
             S(10)= Zeilen pro Seite
             S(12)= Erste Seitennummer
```

Ergebnis:

```
   Job in die Spooldatei eingetragen : T(1)=0
   Spooldatei voll                   : T(1)=164
```

TAMOS-Kommunikation Phantom zu Master

Wenn das Anwenderprogramm am Phantom-Port ausgeführt wird, kann durch den Sprung in diese Routine eine Nachricht am Masterplatz und eine Anwort empfangen werden.

Aufruf mit Nachricht aus TF.PARAM:

```
   ... GOSUB 1180
   Vorgabe: T(1) = Nachrichtennummer in TF.PARAM
```

Aufruf mit eigenem Text:

```
   ... GOSUB 1190
   Vorgabe: M7$  = Nachricht
```

Ergebnis: Antwort in M7$.

Programmierbeispiele

Druckerfehlerroutine

Diese Routine prüft die Fehler 26 und 50 ab. Bei Fehler 50 wird am Bildschirm ausgegeben "Drucker belegt oder nicht bereit", bei Fehler 26 "Drucker abgeschaltet oder Papierende".
Mit der Eingabe CAN kann das Programm abgebrochen werden. In diesem Fall wird "Unterbrechung durch Bediener" als Nachricht ausgegeben und in die Logdatei eingetragen. Jede andere Eingabe als CAN führt zur Befehlswiederholung.

Diese Routine kann sowohl an einem Normal-Port als auch unter dem Spooler laufen, wobei dann die Kommunikationsroutine mitbenutzt wird.

Aufruf:

```
... IF ERR 0 GOSUB 1320  vor Druckereröffnung
... IF ERR 0 GOSUB 1330  vor Druckbefehlen
```

Tritt ein anderer Fehler als 26 oder 50 auf, so wird in die normale Fehlerroutine verzweigt.

Papierwechsel unter Spooler

Es wird der Inhalt von S7$ (Papierbeschreibung) und zusätzlich "EINGELEGT Y/CAN" angezeigt.

Aufruf:

```
... GOSUB 1490
```

Ergebnis:

Master gibt Y ein	: RETURN
Master gibt CAN ein	: Nachrichtenausgabe, Log-Datei-Eintrag und Programmabbruch
sonst	: Wiederholung der Anzeige und Eingabe

Übernahme des TAMOS-Bereiches beim Spooler und Druckereröffnung

Programmierbeispiele

Die vom Spooler bereitgestellten Daten werden übernommen und es wird auf Papierwechsel abgefragt. Bei Papierwechsel wird in die oben beschriebene Routine verzweigt. Anschließend erfolgt die Druckereröffnung mit den Parametern "Seitenhöhe" und "von/bis-Druckerposition". Tritt ein Fehler beim Eröffnen auf, so wird in die Druckerfehlerroutine verzweigt.

Aufruf:

```
... GOSUB 1550
```

Ergebnis: Bei normalem Verlauf: RETURN
Bei CAN bei Papierwechsel oder OPEN-Fehler:
Programmabbruch

Programmierbeispiele

21.9.3 Der Basisteil

```
10 REM01 Programmname
20 DIM H7$(122),V7$(4),L7$(126),M7$(50),I$(13),IO$(50),
   I1$(18),I2$(8)
30 DIM 4%,IO,1%I1,I2,2%,T(3),N1,N2,N5,N6
40 DIM D7$(20),P7$(18),S7$(25),F7$(25),S2$(8)
50 DIM 1%,S(12)
60 REM ÜBERSPRINGEN STANDARDROUTINE**********************
70 GOTO 1640
80 REM BILDSCHIRM AUSGEBEN*******************************
90 LET V7$="400/"
100 LET T(0)= SPC 6     /* BA-NR.
110 CALL 3,T(0),H7$     /* LESEN AUS COMMON-BEREICH DES BA
120 PRINT 'CS';'SB';V7$;H7$(5,66);TAB(71,0);H7$(67,74);
    TAB(0,24);H7$(75,89);
130 RETURN
140 REM NACHRICHT AUSGEBEN*******************************
150 N1=3                /* MELDUNG LESEN, ANZEIGEN, LOGGEN
152 N5=15               /* TABELLE BASIS
154 N6=T(1)             /* MELDUNGS-NR.
156 S2$="TF.PARAM"

160 CALL 84,N1,N2,M7$,S2$,N5,N6
165 IF NOT N2 GOTO 170
166 PRINT TAB(0,24);"STATUS:";N2; /* STATUSMELD. AUSWERTEN
167 SIGNAL 3,100
168 CHAIN "TA.END"    /* ABBRUCH
170 SIGNAL 3,20
180 RETURN
270 REM FEHLERBEHANDLUNG*********************************
280 IF SPC 8>97 RETURN -1
290 IF ERR 0 GOTO 350
340 LGERR                           /* LOG-DATEI-EINTRAG
350 CHAIN "TA.ABO"
360 REM SATZ EINFÜGEN************************************
370 LET I1=2
380 SEARCH #I2,1,0;M7$,T(0),I1    /* 1. FREIER SATZ
390 IF I1 GOTO 550                /* NICHT DA
400 REM ON I2 GOSUB ... UNTERPROGRAMM DATENSATZ SCHREIBEN
410 LET IO=T(0)
420 SEARCH #I2,4,1;M7$,T(0),I1    /* BEGRIFF EINFÜGEN
430 IF I1=0 RETURN 1              /* OK
440 IF I1=1 GOTO 490
450 LET M7$="KANAL,FEHLER "
460 LET M7$(14)=I2+I1/10
470 GOSUB 160                   /* NACHRICHTENAUSGABE
480 GOTO 340                    /* LOG-DATEI UND ABBRUCH
490 LET T(1)=263                /* SATZ BEREITS VORHANDEN
500 GOSUB 150                   /* NACHRICHTENAUSGABE
```

Programmierbeispiele

```
510 LET I1=3
520 SEARCH #I2,1,0;M7$,I0,I1    /* SATZ FREIGEBEN
530 IF I1 GOTO 450              /* DATEIFEHLER
540 RETURN
550 IF I1<>3 GOTO 450
560 LET T(1)=459
570 GOSUB 150
580 GOTO 200
590 REM CALL 1*****************************************
600 LET V7$=I$
610 LET I1=0
620 LET I0$=M7$
630 IF I$(1,2)="32" IF CHR I0<0 GOSUB 890
640 CALL 1,I1,I$,I0$,I0,L7$
650 LET I$(1,2)=V7$
660 LET M7$=I0$
670 INPUT 'BP',I0$
680 IF I0$<>"?" GOTO 770
690 PRINT  TAB (17,24);'CFF'; TAB (17,24);
700 IF I2$(1,1)="0" PRINT "@  /";
710 IF I2$(2,2)="0" PRINT "▲  /";
720 FOR I2=3 TO LEN I2$
730   IF I2$(I2,I2)="0" PRINT I1$(I2*3-8,I2*3-6);"/";
740 NEXT I2
750 PRINT 'BS';'BP';
760 GOTO 670
770 IF P$(2,2)="1" LET I0=LEN I0$
780 CALL 1,I1,I$,I0$,I0,I1$,I2$,L7$
790 IF I1=-1 GOTO 610
800 IF I$(2,2)="0" IF I1=0 CALL 60,I0$,I0$
810 ON I1 GOTO 840
820 LET I$(1,2)=V7$
830 RETURN
840 LET I$(1,1)="3"
850 IF I$(2,2)="2" IF CHR I0<0 GOSUB 890
860 LET I2=0
870 CALL 1,I2,I$,M7$,I0,L7$
880 GOTO 820
890 LET L7$="-",L7$(1,12),"#.#####"
900 LET I2=I$(7,8)
910 IF I$(12,12)="0" LET I2=I2-1
920 LET T(0)=I$(11,11)
930 LET M7$=I0 USING L7$(14-I2,15+T(0))
940 LET I2=I$(3,4)
950 LET T(0)=I$(5,6)
960 PRINT  TAB (I2,T(0));M7$;
970 LET I$(1,1)="0"
980 RETURN
990 REM JOB IN SPOOL-DATEI EINTRAGEN*********************
1000 OPEN #13,"TF.PORT"
```

Programmierbeispiele

```
1010 READ #13,0,22+SPC6*44;T(2),S(1),T(3),S(2),S(3),S(4);
1020 CLOSE #13
1030 OPEN #13,"TF.SPOOLQUEUE"
1040 FOR S(11)=0 TO  CHF 13-1
1050   READ #13,S(11);M7$
1060   IF M7$(1,1)=" " GOTO  1100
1070 NEXT S(11)
1080 LET T(1)=164
1090 GOTO 1150
1100 LET T(1)=0
1110 LET S(11)=0
1120 LET T(2)=(SPC 2-INT(SPC 2/24)*24)*3600+INT(SPC 3/10)
1130 IF  LEN D7$<3 LET D7$=H7$(5)
1140 WRITE #13,-2;D7$,S(0),T(2),S(1),T(3),S(2),S(3),S(4),
  P7$,S(5),S(6),S(7),S7$,S(8),S(9),S(10),S(11),S(12),F7$;
1150 CLOSE #13
1160 RETURN
1170 REM KOMMUNIKATION PHANTOM ZU MASTER*****************
1180 READ #1, INT (T(1)/10)+15, FRA (T(1)/10)*510;M7$;
1190 LET T(0)=- SPC 6                 /* TASK-NR NEGATIV
1200 CALL 2,T(0),T(0),M7$
1210 READ #1,14,137;M7$(38);
1220 LET M7$="←377←←207←←376←←211←←376←←221←
B←230←",M7$(38),"←376←←212←"
1230 IF ERR 0 GOTO 1290
1240 CALL 4,M7$                       /* SPOOLSTATUS AUSGEBEN
1250 SIGNAL 3,250
1260 CALL 3,T(0),T(0),M7$             /* ANTWORT ABHOLEN
1270 IF T(0)=- SPC 6 GOTO 1210 /* KEINE ANTWORT
1280 RETURN
1290 IF  SPC 8<>38 GOTO 290
1300 GOTO 1250
1310 REM DRUCKERFEHLER***********************************
1320 IF  SPC 8=50 GOTO 1460     /* DRUCKER BELEGT
1330 IF  SPC 8<>26 GOTO 280     /* STANDARDFEHLER
1340 LET T(1)=183
1350 IF  SPC 6=1 GOTO 1430      /* BEI SPOOLING
1360 GOSUB 150                  /* NACHRICHT OHNE SPOOLING
1370 INPUT 'SF', TAB (70,24),M7$
1380 IF M7$(1,3)<>"CAN" RETURN -1
1390 LET T(1)=123
1400 GOSUB 150
1410 GOSUB 200                  /* LOGGING
1420 CHAIN "TA.ABO"
1430 GOSUB 1180                 /* NACHR. UEBER SPOOLER
1440 IF ERR 0 GOSUB 1320
1450 GOTO 1380
1460 LET T(1)=182               /* DRUCKER BELEGT
1470 GOTO 1350
1480 REM PAPIERWECHSEL UNTER SPOOLER*********************
```

Programmierbeispiele

```
1490 LET M7$=S7$," EINGELEGT? Y/CAN"
1500 GOSUB 1190
1510 LET S(7)= ABS (S(7))
1520 IF M7$(1,1)="Y" RETURN
1530 IF M7$(1,3)="CAN" GOTO 1390
1540 GOTO 1490
1550 LET T(0)=SPC 6 /*TAMOS-BER. ÜBERNEHMEN+DRUCKER-OPEN
1560 CALL 3,T(0),H7$,F7$,S7$,S(8),S(9),S(10),S(7),S(6),
      S(12)
1570 IF S(7)<0 GOSUB 1490
1580 LET M7$=(S(8)+1000)*1000000+S(9)*1000+S(10)
1590 LET L7$=M7$(3)
1600 IF  ERR 0 GOSUB 1320
1610 OPEN #12;L7$,"$LPT"
1620 IF  ERR 0 GOSUB 280
1630 RETURN
1640 REM PROGRAMMANFANG**********************************
1650 GOSUB 90
```

22 Fehlerbehandlung

In diesem Kapitel werden die Bedeutungen der BASIC-Fehlernummern erläutert sowie mögliche Reaktionen zur Fehlerbehebung beschrieben.
Das gleiche erfolgt für Arbeitsplatzfehler, soweit diese von einem BASIC-Programm verursacht werden können.

22.1 BASIC-Fehler

Die mit * gekennzeichneten Fehler können nicht mit einer IF ERR 0-Anweisung abgefangen werden. Wurde eine IF ERR 0-Anweisung abgesetzt, führen diese Fehler zum Programmabbruch.

Fehler	Bedeutung	Mögliche Reaktion oder Erläuterung
1	Syntaktischer Fehler	Syntax der Anweisung überprüfen
2	Unzulässige String-Operation	String dimensionieren/Anweisung ändern
3*	Speicherüberlauf (Programm zu groß)	Programm in Segment zerlegen oder mit SAVE Partitiongröße erhöhen.
4	Formatfehler	String-Literale überprüfen
5	Unzulässiges Zeichen	Anweisung überprüfen
6*	Zeilennummer nicht existent	In einem Kommando, einer GOTO- oder GOSUB-Anweisung wird eine nichtvorhandene Zeilennummer referiert
7	Renumber-Abbruch durch "ESC", Programmverlust	Nur Neueingeben oder Neuladen von Platte möglich
8	Mehr als 340 Variablennamen dimensioniert	Numerische Variablen in Vektoren zusammenfassen
9	Unzulässiges Wort	Verursachende Anweisung überprüfen
10	RUN-Kommando mit Zeilennummer nicht erlaubt	RUN ohne vorangestellte Zeilennummer absetzen

Fehlerbehandlung

Fehler	Bedeutung	Mögliche Reaktion oder Erläuterung
11	Fehlerhafter Klammerausdruck	Anzahl der Klammern überprüfen
12	Programm ist gegen Listen/Kopieren geschützt	Manager benachrichtigen
13	Numerischer Wert > 9.9999999999999E+62	Berechnung nicht möglich
14	Keine weiteren Konstanten definiert	DATA-Anweisungen überprüfen
15	Arithmetischer Überlauf	Division durch 0 oder Wert nicht mehr darstellbar
16*	Zu tiefe Unterprogramm-Schachtelung	Unterprogrammschachtelung auf 32 Stufen begrenzen
17	RETURN auf Hauptprogramm-Ebene	GOSUB- und RETURN-Anweisungen überprüfen
18*	Zu tiefe "FOR/NEXT"-Schachtelung	FOR/NEXT-Schachtelung auf 8 Stufen begrenzen
19*	"FOR" ohne zugehöriges "NEXT"	FOR- und NEXT-Anweisungen überprüfen
20*	"NEXT" ohne zugehöriges "FOR"	FOR- und NEXT-Anweisungen überprüfen
21	Ausdruck zu komplex	Ausdurck durch Zerlegen in 2 Anweisungen vereinfachen
22	Nicht genug Plattenblöcke für "swap-out" vorhanden	CLEANUP auf der Systemplatte veranlassen
23	Matrixgröße überschreitet Erstdefinition	Matrix mit DEALLO vorher freigeben
24	Stringdimensionierungen dürfen nur einen Index haben	DEALLO einsetzen
25	String oder Matrix nicht dimensioniert	DIM-Anweisung einfügen

Fehlerbehandlung

Fehler	Bedeutung	Mögliche Reaktion oder Erläuterung
26	Logische Einheit ist nicht bereit	Manager benachrichtigen
27*	Syntaktischer Fehler in Anwenderfunktion	DEF-Anweisung überprüfen
28	Unzulässiger Wert für Index, Kanalnummer, SIGNAL-Parameter	Verursachende Anweisung überprüfen
29	Unzulässiger Funktions-Aufruf	Funktion in gesonderter Anweisung aufrufen
30	Anwenderfunktion nicht definiert	DEF FN-Anweisung fehlt oder tritt nach Anwender-funktionsaufruf auf
31*	Anwenderfunktion zu tief geschachtelt	Schachtelungstiefe auf 5 begrenzen
32	Matrizen haben ungleiche Dimensionen	Dimensionierungen überprüfen
33	Operand ist keine Matrix	Matrix dimensionieren
34	Dimensionen sind nicht verträglich	Dimensionierungen überprüfen
35	Matrix ist nicht quadratisch	Dimensionierung überprüfen
36	Aufgerufenes Unterprogramm ist nicht vorhanden	Falsche CALL- oder SOFT-SUB-Nummer, sonst Manager benachrichtigen
37	Ausdruck als Operand in einer "CALL"-Anweisung	Wert des Ausdrucks durch LET-Anweisung in einer Variablen abstellen
38	Fehler von aufgerufenem "CALL"-Unterprogramm erkannt	siehe "CALL-Anweisungen"
39	Formatierte Ausgabe überschreitet Puffergröße	Ausgabe auf zwei Anweisungen verteilen
40	Kanal bereits belegt	Andere Kanalnummer wählen

Fehlerbehandlung

Fehler	Bedeutung	Mögliche Reaktion oder Erläuterung
41	Unzulässiger Dateiname	Namens-Syntax überprüfen
42	Datei nicht gefunden	Manager benachrichtigen (Datei auf anderer logischer Einheit?)
44	Unzulässige Dateiart	Dateityp überprüfen
45	Datei ist gegen Lesen geschützt	Manager benachrichtigen
46	Datei ist gegen Schreiben geschützt	Manager benachrichtigen
47	Nicht genügend freie Blöcke auf der logischen Einheit	Andere logische Einheit wählen
48	Nicht genügend freie Blöcke vorhanden	Manager benachrichtigen
49	Kanal nicht eröffnet	Falsche Kanalnummer/Fehlende OPEN #-Anweisung
50	Datei ist bereits eröffnet	Datei gesperrt, später erneut versuchen
51	Unzulässige Satznummer	Dateiende?/Satznr. ändern
52	Satz nicht vorhanden	Dateiende?/Satznr. ändern
53	Unzulässige Feldnummer	Feldnummer ändern
54	Unverträglicher Feldtyp	Ausgabeliste ändern
55	Direktausführung der Anweisung nicht möglich	-
56	Kein Programm zur Ausgabe vorhanden	Programm laden
57	Stringvariable bereits definiert	Dimensionierung löschen
58	Fehler in Aufbereitungs-Maske	Maske überprüfen

Fehlerbehandlung

Fehler	Bedeutung	Mögliche Reaktion oder Erläuterung
59*	RUNMAT oder SOFTSUBC nicht im System	Manager benachrichtigen
60	Zu viele Werte eingegeben	Programm erneut ausführen
61	Matrizen haben unterschiedliche Element-Formate	Element-Formate in DIM-Anweisung angleichen
62	Signalpuffer voll/Teilnehmernummer nicht vorhanden	SIGNAL-Anweisung überprüfen/später absetzen
63	Kommando ist im Modus "LOAD" unzulässig	Kommando aus Textdatei löschen
64	Fehlende Zeilennummer im Modus "LOAD"	Zeilennummer in Textdatei einfügen
65	Dateiname bereits von einer Datei anderen Typs belegt	Anderen Dateinamen wählen
66	Dateiname belegt (Datei wird angelegt/ersetzt)	Anderen Dateinamen wählen
67	Dateiname belegt und kein "!" angegeben	"!" anhängen oder anderen Namen wählen
68	Dateiname belegt	Anderen Dateinamen wählen
69	Datei ist ein Processor oder Driver	Dateinamen überprüfen
70	Lesefehler	Lesevorgang wiederholen
71	Datei ist kein Peripheriegerät	Dateinamen überprüfen
72	Kommando-String in "CHAIN" zu lang	Teil des Kommandostrings in eine CHAIN-Anweisung im Folgesegment einbauen

Fehlerbehandlung

Fehler	Bedeutung	Mögliche Reaktion oder Erläuterung
73	Unzulässiger Parameter/Angesprochener Driver nicht konfiguriert	siehe "ENV # - Anweisung"/Systemkonfiguration mit SYSMOD anpassen/Common Bufferpool nicht konfiguriert/fehlende Statusabfrage.
74	Fehler in Datenübertragung	siehe "Der programmierbare Leitungscontroller" (DFÜ-Handbuch).
75	Kein aufrufbares BASIC-Programm	Datei kein BASIC-Programm oder lesegeschützt
76*	Partition zu klein für LINK	Partitionangabe des Startsegments vergrößern
77*	Variablenname nicht definiert im Folgesegment	Variable im Folgesegment dimensionieren
78	BASIC-Anweisung länger als 254 Zeichen	Anweisung umformulieren
79	Kein freier Kanal vorhanden	-
80	Variablenname existiert nicht	-
81*	Programm ist größer als das Active-file	Active-file-Größe einstellen
82	INDEX-DATEI: Versuch, über Satzende zu schreiben	Dateieintrag kürzen oder Datei umorganisieren
83	Fehler bei einer GLOBAL-Anweisung: 1. DEALLO auf GLOBAL-Variable 2. DIM-Statement zwischen GLOBAL-Statements 3. GLOBAL-Anweisung mit implizitem Index	Variable lokal (DIM) deklarieren, oder DEALLO durch CLEAR ersetzen/ DIM hinter GLOBAL-Statements einfügen Nur Ganzzahl als Index verwenden

Fehlerbehandlung

Fehler	Bedeutung	Mögliche Reaktion oder Erläuterung
84	Fehler bei SNA-DFÜ-Auftrag	siehe DFÜ-Handbuch
85	Unterschiedliche Reihenfolge der GLOBAL-Statements	GLOBAL-Variablen in einheitlicher Reihenfolge deklarieren
86*	BASIC-Programm ist zerstört	Manager benachrichtigen
87	Datenträger ist schreibgeschützt	-
88	Ungülige Laufwerksnummer	-
89	frei	-
90	frei	-
91	frei	-
92	frei	-
93	frei	-
94	frei	-
95	frei	-
96	frei	-
97	"$DEC" nicht im System	Manager benachrichtigen
98	Eingabe ist nicht numerisch	Programmausführung wiederholen
99	Taste "ESC" oder "CTL-C" gedrückt	-
123	Datensatz gesperrt	Datensatz später erneut ansprechen

Fehlerbehandlung

22.2 Fehlermeldungen des Arbeitsplatzes

Auf der rechten oberen Seite der Tastatur befindet sich eine Rasttaste mit der Aufschrift "S" und einer roten LED-Anzeige, die immer dann aufleuchtet, wenn der Arbeitsplatz einen Fehler erkennt.
Der Fehler kann durch den Arbeitsplatz selbst sowie durch den Arbeitsplatzdrucker verursacht worden sein.

Wird daraufhin die Taste "S" gedrückt, wird die Nachrichten-Zeile (unterste Zeile) des Bildschirms gelöscht und es erscheint:

```
ERROR : xx yy zz
```

Anhand der Nummern xx, yy und zz kann der Fehler entschlüsselt und gegebenenfalls behoben werden.

Die Meldung wird gelöscht, sobald die Taste "S" zum zweiten Mal gedrückt wird. Sollte die Bildschirmzeile vor dem ersten Betätigen der "S"-Taste beschrieben sein, bleibt die Information erhalten und wird nun wieder sichtbar.

Die Fehlerzeile ist wie folgt aufgebaut:

```
ERROR : xx yy zz
              |  |  |
              |  |  +-- Fehlermeldungen des 1. Platzdruckers
              |  +----- Fehlermeldungen des 2. Platzdruckers
              +-------- Fehlermeldungen des Arbeitsplatzes
```

Fehler, die durch das Anwenderprogramm verursacht werden können, sind z.B.:

```
ERROR 05 00 00  LEAD-IN-Code ohne Folgecode

ERROR 06 00 00  Folgecode nach LEAD-IN > 240
```

Reaktion:

INPUT- und PRINT-Anweisungen auf korrekte Codierung überprüfen.

Alle weiteren ERROR-Codes sind im Handbuchmodul Fehlermeldungen beschrieben.

Stichwortverzeichnis

23 Stichwortverzeichnis

Stichwortverzeichnis

B

C

Stichwortverzeichnis

Stichwortverzeichnis

D

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis