Allen-Bradley

BASIC-Programmiersprache

(Bestellnummer 1746-BAS)

Referenzhandbuch

Wichtige Anwendungshinweise

Elektronische Geräte unterscheiden sich in ihrer Arbeitsweise von elektromechanischen Geräten. Die Publikation “Safety Guidelines for the Application, Installation, and Maintenance of Solid State Controls” (Publication SGI-1.1) beschreibt einige wichtige Unterschiede zwischen elektronischen und festverdrahteten, elektromechanischen Geräten. Aufgrund dieser Unterschiede und der vielfältigen Einsatzmöglichkeiten elektronischer Geräte müssen Sie als Verantwortlicher für die Anwendung dieses Gerätes sicherstellen, daß jede Anwendung dieses Gerätes die jeweiligen Anforderungen erfüllt.

Für Schäden, die direkt oder indirekt durch die Verwendung oder den Einsatz dieser Geräte entstehen, kann Allen-Bradley Company keinesfalls verantwortlich oder haftbar gemacht werden.

Die Beispiele und Abbildungen in diesem Handbuch sind ausschließlich zur besseren Texterläuterung aufgeführt. Aufgrund der vielfachen Möglichkeiten und Anforderungen jedes einzelnen Verwendungszwecks kann die Allen-Bradley Company keine Verantwortung oder Haftung für tatsächliche Einsätze, die auf diesen Beispielen und Abbildungen beruhen, übernehmen.

Allen-Bradley Company übernimmt keine Patenthaftpflicht für die Anwendung von Informationen, Schaltungen, Geräten bzw. Softwareprogrammen, die in diesem Handbuch beschrieben werden.

Jede Wiedergabe des Inhalts dieses Handbuchs, ganz oder auszugsweise, ist ohne die schriftliche Genehmigung von Allen-Bradley nicht gestattet.

Besondere Hinweise in diesem Handbuch sollen den Anwender auf Sicherheitsmaßnahmen aufmerksam machen.

ACHTUNG: Weist auf Informationen und Verfahrensweisen oder Umstände hin, die zu Körperverletzungen oder sogar Lebensgefahr, zu Sachschaden oder wirtschaftlichem Verlust führen können.

Achtungshinweise helfen Ihnen:

- eine Gefahr festzustellen
- die Gefahr zu vermeiden
- die Konsequenzen zu erkennen

Wichtig: Weist auf Informationen hin, die äußerst wichtig für die erfolgreiche Anwendung und für das gründliche Verstehen des Produktes sind.

Zusammenfassung der Änderungen

Die nachstehenden Informationen fassen die Änderungen zusammen, die seit der letzten Drucklegung vorgenommen wurden.

Neue Informationen

Die folgende Tabelle führt Abschnitte auf, die neue Merkmale und zusätzliche Informationen über bestehende Funktionen dokumentieren, und zeigt, an welcher Stelle im Handbuch diese neuen Hinweise zu finden sind.

Neue Informationen über:	siehe Kapitel
Dokumentationssatz des BASIC-Moduls	Vorwort

BASIC-Befehle

Kapitel 4

Aufrufe der Befehlszeile

Kapitel 5

Zuordnungsfunktionen

Kapitel 6

Steuerfunktionen

Kapitel 7

Statusfunktionen

Kapitel 11

Ausgangsfunktionen

Kapitel 12

Eingangsfunktionen

Kapitel 13

Konfigurationsfunktionen

Kapitel 14

Zeichenkettenfunktionen

Anhang A

Vorwort

Lesen Sie dieses Vorwort, um sich mit dem restlichen Handbuch vertraut zu machen. Es werden die folgenden Themen behandelt:

- Leser dieses Handbuchs
- Zweck dieses Handbuchs
- Verwendung dieses Handbuchs
- Begriffe und Abkürzungen
- Konventionen in diesem Handbuch
- Technische Unterstützung durch Allen-Bradley

Leser dieses Handbuchs

Dieses Handbuch soll den Bediener und Programmierer bei der Entwicklung, Installation, Programmierung und Fehlersuche im Zusammenhang mit Steuersystemen, die speicherprogrammierbare Steuerungen von Allen-Bradley verwenden, unterstützen.

Sie sollten grundlegende Kenntnisse über SLC-500™-Geräte besitzen und mit programmierbaren Steuerungen vertraut sein. Ferner sollten Sie in der Lage sein, die zur Steuerung einer Anwendung erforderlichen Kontaktplanbefehle zu interpretieren. Verfügen Sie nicht über die erforderlichen Kenntnisse, setzen Sie sich bitte vor der Verwendung dieses Gerätes mit Ihrer Allen-Bradley-Geschäftsstelle in Verbindung, die Sie gerne über Ausbildungsseminare informiert.

Zweck dieses Handbuchs

Das Referenzhandbuch zur BASIC-Programmiersprache wird bei der Programmierung des BASIC-Moduls verwendet und dient ausschließlich zur Bezugnahme.

Tabelle V.1
Inhalt dieses Handbuchs

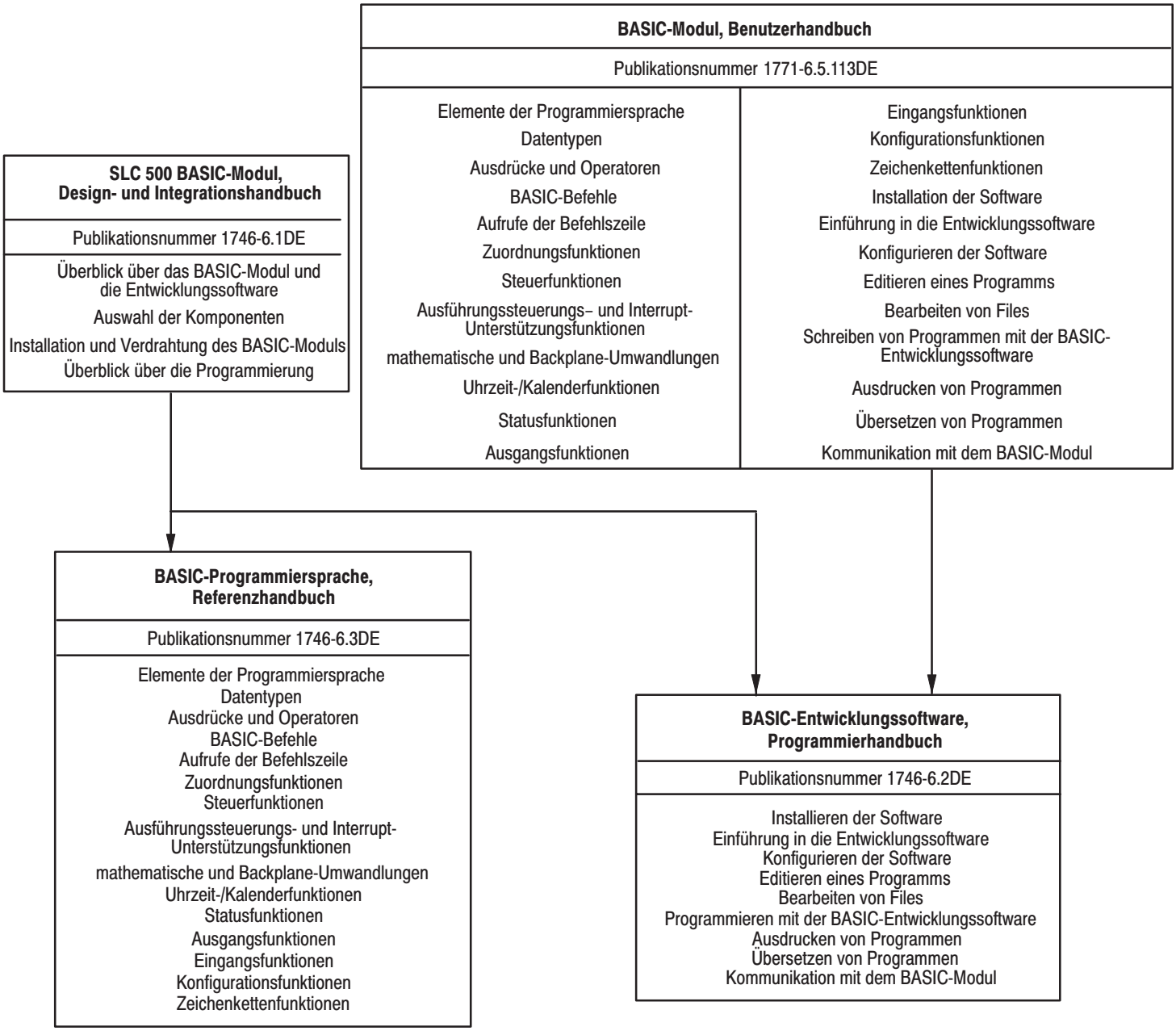
Kapitel	Titel	Inhalt
	Vorwort	Hinweise zum Zweck, Hintergrund, Umfang und zur beabsichtigten Leserschaft dieses Handbuchs
1	Elemente der Programmiersprache	Beschreibung der Zeilen, Zeilennummern, Anweisungen, Befehle, Operatoren und Zeilenlängen eines BASIC-Programms
2	Datentypen	Beschreibung und Darstellung von Datentypen sowie der Bezeichnungen und Typen von Variablen
3	Ausdrücke und Operatoren	Beschreibung und Darstellung arithmetischer, logischer, relationaler, trigonometrischer, funktioneller, logarithmischer, Zeichenketten- und Sonderfunktionsoperatoren
4	BASIC-Befehle	Beschreibung und Darstellung der Befehle BRKPNT, CONT, [CTRL–C] (einschließlich Deaktivierung und Aktivierung dieses Befehls), [CTRL–S], [CTRL–Q], EDIT, ERASE, IDLE, LIST, LIST@, LIST#, MODE, NEW, NULL, PROG, PROG1, PROG2, RAM, REM, REN, ROM, RROM, RUN, SNGLSTP, VER und XFER sowie der Aufrufe 18 und 19
5	Aufrufe der Befehlszeile	Beschreibung und Darstellung der Aufrufe 73, 74, 77, 81, 82, 101, 103, 104, 109, 110 und 111
6	Zuordnungsfunktionen	Beschreibung und Darstellung der Funktionen CLEAR, CLEARI, CLEARS, DATA, DIM, LET und RESTORE
7	Steuerfunktionen	Beschreibung und Darstellung der Funktionen CLOCK1, CLOCK0, DO–WHILE, DO–UNTIL, END, FOR–TO–(STEP)–NEXT, GOTO, IF–THEN–ELSE, NEXT und ON–GOTO
8	Ausführungssteuerungs- und Interrupt-funktionen	Beschreibung und Darstellung der Aufrufe 16, 17, 20, 21, 26, 38, 70, 71, 72 sowie der Funktionen GOSUB, ONERR, ON–GOSUB, ONTIME, PUSH, POP, RETI, RETURN und STOP
9	Mathematische und Backplane-Umwandlungsfunktionen	Beschreibung und Darstellung der Aufrufe 14, 15, 24 und 25
10	Uhrzeit-/ Kalenderfunktionen	Beschreibung und Darstellung der Aufrufe 40, 41, 42, 43, 44, 45, 46, 47, 48 und 52
11	Statusfunktionen	Beschreibung und Darstellung der Aufrufe 36, 51, 55, 58, 59, 75, 80, 86, 87, 95, 97, 98, 108, 113, 120 und 121
12	Ausgangsfunktionen	Beschreibung und Darstellung der Aufrufe 23, 28, 29, 31, 37, 54, 57, 85, 91, 93, 94, 96, 112, 114, 115, 123 sowie der Funktionen PRINT, PH0., PH1. und ST@

Kapitel	Titel	Inhalt
13	Eingangsfunktionen	Beschreibung und Darstellung der Aufrufe 22, 27, 29, 35, 53, 56, 84, 90, 92, 117, 118, 122 sowie der Funktionen GET, INPL, INPS, INPUT, LD@ und READ
14	Konfigurationsfunktionen	Beschreibung und Darstellung der Aufrufe 30, 78, 99, 105, 119 sowie der MODE-Funktion
15	Zeichenkettenfunktionen	Beschreibung und Darstellung der Aufrufe 60, 61, 62, 63, 64, 65, 66, 67, 68 und der STRING-Funktion
Anhang A	Umwandlungstabelle dezimal/hexadezimal/ oktal/ASCII	Auflistung der Umwandlungen von Dezimal-/Hexadezimal-/Oktal-/ASCII-Werten
Anhang B	Schnellinformation zu BASIC-Befehlen, Anweisungen und Aufrufen	Auflistung der verschiedenen Befehle, Anweisungen und Aufrufe, die für die BASIC-Programmierung erforderlich sind

Dokumentationssatz des BASIC-Moduls

Der Dokumentationssatz des BASIC-Moduls ist entsprechend den Aufgabenbereichen in verschiedene Handbücher untergliedert. Zu diesen Aufgabenbereichen gehören die Entwicklung, Integrierung und Programmierung des BASIC-Moduls. In Abbildung V.1 sind die einzelnen BASIC-Dokumentationen und die in den einzelnen Handbüchern enthaltenen Informationen aufgeführt.

Abbildung V.1
Dokumentationssatz des BASIC-Moduls



Literaturhinweis

Die folgenden Dokumentationen enthalten zusätzliche Informationen über die SLC™- und PLC®-Geräte von Allen-Bradley und sind bei Ihrer Allen-Bradley-Geschäftsstelle bzw. bei Ihrem Allen-Bradley-Händler erhältlich.

Tabelle V.2
Themenverwandte Dokumentationen

Thema	Dokumentation	Dokumentationsnummer
Überblick über die Geräte der Reihe SLC 500	SLC 500, Systemübersicht	1747–2.30DE
Beschreibung der Installation und Anwendung der modularen speicherprogrammierbaren Steuerung SLC 500	Installations- und Benutzerhandbuch für die modulare Hardware-Konfiguration speicherprogrammierbarer Steuerungen	1747–6.2DE
Beschreibung der Installation und Anwendung der fest konfigurierten speicherprogrammierbaren Steuerungen SLC 500	Installation & Operation Manual for Fixed Hardware Style Programmable Controllers	1747–NI001
Ein verfahrensbezogenes Handbuch für technisches Personal zur Entwicklung von Steueranwendungen mit der APS-Software	Benutzerhandbuch für die erweiterte Programmiersoftware (APS) von Allen-Bradley	1747–6.4DE
Referenzhandbuch mit Statusfiledaten, Befehlssatz- und Fehlersuchinformationen zur APS-Software	Bedienerhandbuch für die erweiterte Programmiersoftware (APS) von Allen-Bradley	1747–6.11DE
Detaillierte Informationen über die Erdung und Verdrahtung von speicherprogrammierbaren Steuerungen von Allen-Bradley	Richtlinien zur Erdung und Verdrahtung von speicherprogrammierbaren Steuerungen von Allen-Bradley	1770–4.1DE
Beschreibung der wichtigsten Unterschiede zwischen elektronischen speicherprogrammierbaren Steuerungen und festverdrahteten elektromechanischen Geräten	Application Considerations for Solid–State Controls	SGI–1.1
Artikel über Leitergrößen und -arten zur Erdung von elektrischen Geräten	National Electrical Code	Veröffentlichung durch National Fire Protection Association of Boston, MA.
Eine komplette Auflistung aktueller Dokumentationen von Allen-Bradley, einschließlich Bestellinformationen. Gibt ferner an, ob die Dokumentationen auf CD-ROM bzw. in anderen Sprachen erhältlich sind.	Allen-Bradley Publikationsindex	SD499
Glossar mit Begriffen und Abkürzungen im Bereich der industriellen Automatisierungstechnik	Allen–Bradley Industrial Automation Glossary	AG–7.1DE
Beschreibung der Installation und Anwendung eines Moduls, das als Übergangseinheit zwischen DH-485-Netzwerken und Geräten, die das DF1-Protokoll einsetzen, fungiert.	DH–485/RS–232C Interface Module User's Manual	1747–NU001
Beschreibung der Integration des BASIC-Moduls 1746-BAS von Allen-Bradley in ein SLC-Steuerungssystem	SLC 500 BASIC-Modul, Design- und Integrationshandbuch	1746-6.1DE
Beschreibung der Programmierung eines BASIC-Moduls 1746-BAS von Allen-Bradley mit der BASIC-Entwicklungssoftware	Programmierhandbuch der BASIC-Entwicklungssoftware	1746-6.2DE
Beschreibung der Installation und Anwendung eines Handprogrammiergerätes	Handprogrammiergerät, Benutzerhandbuch	1747-NP002

Begriffe und Abkürzungen

Die folgenden Begriffe und Abkürzungen beziehen sich speziell auf dieses Gerät. Eine vollständige Auflistung der englischen Allen-Bradley-Fachterminologie befindet sich in Publikation ICCG-7.1, Allen-Bradley Industrial Automation Glossary.

- **BASIC** — die Programmiersprache BASIC-52
- **BASIC-Modul** — SLC 500 BASIC-Modul (Bestellnummer 1746-BAS)
- **BASIC-Entwicklungssoftware** — BASIC-Entwicklungssoftware (Bestellnummer 1747-PBASE)
- **Terminal** — das an den Programmierport des BASIC-Moduls angeschlossene Gerät. Dieses bildet die Schnittstelle zwischen Anwender und BASIC-Programm
- **DH-485** — Netzwerk-Kommunikationsprotokoll
- **EPROM** — lösch- und programmierbarer Nur-Lese-Speicher
- **EEPROM** — elektrisch löschbarer, programmierbarer Nur-Lese-Speicher
- **Speichermodul** — EEPROM oder UVPROM des BASIC-Moduls
- **MTOP** — Systemsteuerwert, der die zuletzt gültige Speicheradresse enthält
- **Programmierport** — Port zur Programmierung des BASIC-Moduls. Der PRT1- oder DH485-Port kann als Programmierport konfiguriert werden.
- **RS-232/423** — serielle Kommunikationsschnittstelle
- **RS-422** — Differential-Kommunikationsschnittstelle
- **RS-485** — Netzwerk-Kommunikationsschnittstelle
- **RAM** — Direktzugriffspeicher
- **ROM** — Nur-Lese-Speicher; bezieht sich auf den Speicherbereich des optionalen Speichermoduls (EEPROM oder UVPROM)
- **SCADA** — Fernwirk- und Datenerfassungssystem
- **skalare Variable** — eine Variable mit nur einem Wert
- **SLC 500** — SLC-500-Steuerungen mit fester und modularer Hardware-Konfiguration
- **UVPROM** — programmierbarer UV-Festwertspeicher

Konventionen in diesem Handbuch

In diesem Handbuch werden die folgenden Konventionen verwendet:

- Mit Aufzählungspunkten wie hier gekennzeichnete Auflistungen enthalten Informationen und keine Verfahrensweisen.
- Numerierte Auflistungen enthalten sequentielle Schritte oder hierarchisch angeordnete Informationen.
- *Kursivschrift* wird zur Hervorhebung von Informationen verwendet.
- Text in **`diesem Font`** kennzeichnet Worte bzw. Sätze, die Sie eingeben müssen.
- Tastenbezeichnungen entsprechen den dargestellten Bezeichnungen und erscheinen als fettgedruckte, von eckigen Klammern umgebene Großbuchstaben (Beispiel: **`[ENTER]`**).
- **[expr]** - kennzeichnet einen Ausdruck, der mit einem Systembefehl, einem Operator oder einer Systemanweisung eingesetzt wird.
- **[ln num]** - kennzeichnet eine Zeilennummer, die mit einem Systembefehl, einem Operator oder einer Systemanweisung verwendet wird.
- **[var]** - kennzeichnet eine Variable, die mit einem Systembefehl, einem Operator oder einer Systemanweisung verwendet wird.

Technische Unterstützung durch Allen-Bradley

Mit 75 Geschäftsstellen in den Bereichen Verkauf/Technische Unterstützung, 512 autorisierten Distributoren und 260 autorisierten Systemintegratoren allein in den USA bietet Allen-Bradley weltweite technische Unterstützung. Außerdem ist Allen-Bradley in jedem größeren Land der Welt vertreten.

Produktspezifische Unterstützung

Ihre Allen-Bradley-Vertretung berät Sie gerne über:

- Verkauf und Beratung
- technische produktspezifische Schulung
- Garantieansprüche
- Unterstützungs- und Serviceverträge

Technische Unterstützung

Bevor Sie technische Unterstützung von Allen-Bradley anfordern, lesen Sie bitte zuerst die Informationen im entsprechenden Kapitel durch. Setzen Sie sich anschließend mit Ihrer Allen-Bradley-Vertretung in Verbindung.

Fragen und Kommentare zu diesem Handbuch

Wenn Sie zu diesem Handbuch Fragen haben, füllen Sie bitte das beiliegende Formular “Publication Problem Report” aus und senden es an uns.

Wenn Sie Verbesserungsvorschläge zum Aufbau dieses Handbuchs haben, können Sie uns unter der folgenden Adresse erreichen:

Allen-Bradley Company, Inc.
Automation Group
Technical Communication, Dept. J602V, T121
P.O. Box 2086
Milwaukee, WI 53201–2086, USA

Kapitel 1

Elemente der Programmiersprache

Dieses Kapitel enthält eine Einführung in die Elemente der BASIC-Programmiersprache. Zu diesen gehören:

- Zeilennummern
- Anweisungen, Befehle und Operatoren
- Zeilenlänge

Zeichensatz

BASIC-Programme setzen sich aus einer Reihe von BASIC-Programmzeilen zusammen. Jede dieser Zeilen besteht wiederum aus einer Reihe von ASCII-Zeichen. Eine vollständige Auflistung der ASCII-Zeichencodes ist in Anhang A enthalten.

BASIC-Programmzeile

Die BASIC-Programmzeilen bestehen aus einer BASIC-Zeilennummer und BASIC-Anweisungen und -Operatoren. Ihre Länge ist auf die BASIC-Zeilenlänge beschränkt.

BASIC-Zeilennummern

BASIC-Zeilennummern werden wie folgt dargestellt:

[ln num]

BASIC-Zeilennummern geben die Reihenfolge an, in der die Programmzeilen im Speicher abgelegt sind. Darüber hinaus werden sie bei Programmverzweigungen und beim Editieren als Referenzen verwendet. Die Zeilennummer kann eine beliebige Ganzzahl zwischen 1 und 65535 sein. Generell beginnt die Numerierung eines BASIC-Programms mit der Zeilennummer 10, wobei die folgenden Zeilen jeweils um 10 erhöht werden. Auf diese Weise besteht die Möglichkeit, dem Programm zu einem späteren Zeitpunkt weitere Zeilen hinzuzufügen.

Da der Computer die einzelnen Anweisungen in numerischer Reihenfolge abarbeitet, brauchen die zusätzlichen Zeilen nicht in fortlaufender Reihenfolge auf dem Bildschirm zu erscheinen. Auch wenn Sie beispielsweise Zeile 35 hinter Zeile 40 eingeben, wird sie vom Computer nach Zeile 30 und vor Zeile 40 abgearbeitet. Diese Vorgehensweise erspart Ihnen die erneute Eingabe des gesamten Programms, falls Sie eine Zeile vergessen haben sollten.

Wichtig: Die erste Zeile des Programms muß ein Kommentar sein.

Im allgemeinen sehen die Zeilennummern eines Programms zu Beginn etwa so aus, wie es nachfolgend in der ersten Spalte dargestellt ist, und können nach der Durchführung von Änderungen etwa so, wie in der zweiten Spalte dargestellt, aussehen.

```
#1          #2
10          5
20          7
30          10
40          15
50          20
60          30
70          35
80          40
.           .
.           .
.           .
```

Wichtig: Wird eine bestimmte Zeilennummer erneut verwendet, gehen alle der ursprünglichen Zeilennummer zugewiesenen Informationen verloren. Gehen Sie deshalb bei der Eingabe von Zeilennummern im Befehlsmodus äußerst sorgfältig vor, da sonst Programmzeilen möglicherweise gelöscht werden. Eine bestehende Zeile kann gelöscht werden, wenn Sie die Zeilennummern ohne nachfolgende Informationen eingeben und anschließend die [**RETURN**]-Taste drücken.

BASIC-Anweisungen, -Befehle und -Operatoren

BASIC-Programmzeilen enthalten eine BASIC-Zeilennummer sowie BASIC-Anweisungen und -Operatoren. Je nach der Programmlogik kann eine Zeile mehr als eine Anweisung enthalten. Diese müssen dann durch einen Doppelpunkt (:) getrennt werden.

BASIC-Zeilenlänge

Eine BASIC-Programmzeile beginnt immer mit einer Zeilennummer und muß mindestens ein Zeichen enthalten. Die maximale Zeilenlänge beträgt 68 Zeichen. Eine Programmzeile wird durch Drücken der [**RETURN**]-Taste abgeschlossen.

Kapitel 2

Datentypen

Dieses Kapitel enthält eine Erläuterung der Definition bzw. Anzeige von Daten innerhalb der BASIC-Programmiersprache unter Verwendung von:

- Datentypen
- Variablen

Datentypen

Datentypen werden in drei Bereiche unterteilt: Argumentstapel-, Zeichenketten- und numerische Elementardatentypen sowie Backplane–Umwandlungsdaten.

Argument-Stapel

Der Argumentstapel (A-Stapel) speichert alle vom BASIC-Modul gegenwärtig verwendeten Konstanten. Operationen wie z.B. Addition, Subtraktion, Multiplikation und Division werden immer auf der Grundlage der ersten beiden Zahlen des Argumentstapels ausgeführt. Das Ergebnis wird anschließend an den Stapel zurückgesendet. Der Argumentstapel ist 203 Bytes lang. Jeder in den Stapel eingespeicherte Fließkommawert erfordert sechs Speicherbytes. Der Argumentstapel kann bis zu 33 Fließkommazahlen speichern, bevor es zu einem Überlauf kommt.

Darüber hinaus können Sie Daten mit dem PUSH-Befehl auf dem Argumentstapel speichern und diese mit dem POP-Befehl aus dem Stapel abrufen. Generell werden die PUSH- und POP-Befehle zusammen mit Aufrufen verwendet und dienen zur Übertragung von Daten an und aus Aufrufroutinen.

Mit dem PUSH-Befehl wird die einzulesende Variable kopiert und die Kopie im oberen Bereich des Argumentstapels gespeichert. Mit dem POP-Befehl wird der Wert aus dem oberen Bereich des Argumentstapels abgelesen und der zu lesenden Variablen zugewiesen.

Zeichenketten-Datentypen

Bei einer Zeichenkette handelt es sich um ein oder mehrere gespeicherte Zeichen. Die in einer Zeichenkette gespeicherten Zeichen bilden im allgemeinen ein Wort oder einen Satz. Zeichenketten ermöglichen die Verwendung von Zeichen anstelle von Ziffern und werden wie folgt dargestellt:

$([expr])

Das BASIC-Modul verwendet eindimensionale Zeichenkettenvariablen, $([expr]). Die Dimension einer Zeichenkettenvariablen (der [ausdr]-Wert) liegt im Bereich zwischen 0 und 254, d.h. Sie können im BASIC-Modul 255 verschiedene Zeichenketten definieren und manipulieren. Den Zeichenketten wird zunächst kein Speicher zugewiesen. Die Speicherzuweisung erfolgt mit der STRING-Anweisung. Zeichenketten werden mit dem $-Operator angegeben und manipuliert.

Bei der Speicherzuordnung einer Zeichenkette müssen Sie die zusätzlichen Bytes berücksichtigen, die vom BASIC-Programm zur Manipulation von Zeichenketten verwendet werden. In BASIC wird für jede angegebene Zeichenkette ein zusätzliches Byte sowie ein weiteres Byte verwendet.

Beispiel:

```
String 106,20
```

Hier wird Speicherplatz für fünf Zeichenketten mit jeweils 20 Bytes (100 Bytes) zugewiesen. Ferner enthalten sind fünf zusätzliche Bytes (1 je Zeichenkette) und ein weiteres zusätzliches Byte.

Im BASIC-Modul können Sie Zeichenketten mit Hilfe der LET- und INPUT-Anweisung sowie mit dem ASC-Operator definieren.

Beispiel:

```
>10 STRING 106,20
>20 $(1)="THIS IS A STRING, "
>30 INPUT "WHAT'S YOUR NAME? - ",$(2)
>40 PRINT $(1),$(2)
>50 END

READY
>RUN

WHAT'S YOUR NAME? - FRED
THIS IS A STRING, FRED

READY
>
```

Darüber hinaus können Zeichenketten mit der LET-Anweisung einander zugeordnet werden.

Beispiel:

```
LET $(2)=$(1)
```

Ergebnis: Der Zeichenkette $(2) wird der in $(1) enthaltene Wert zugewiesen.

Numerische Datentypen

Es gibt zwei verschiedene Arten numerischer Daten:

- Ganzzahlwerte
- Fließkommawerte

Zahlen können in vier Formaten eingegeben und angezeigt werden: Ganzzahlen, Dezimal-, Hexadezimal und Exponentialwerte.

Beispiel:

```
129, 34.98, 0A6EH, 1.23456E+3
```

Das BASIC-Modul interpretiert alle Zahlen als Fließkommawerte, sofern keine Logikoperation ausgeführt wird. Bei der Ausführung logischer Operationen wandelt das BASIC-Modul die Fließkommawerte in Ganzzahlen um, führt die Operation aus und wandelt das Ergebnis wieder in einen Fließkommawert um.

Ganzzahlen

Das BASIC-Modul führt Operationen mit 16-Bit-Ganzzahlen ohne Vorzeichen im Bereich von 0 bis 0FFFFH aus. Sie können alle Ganzzahlwerte im Dezimal- oder Hexadezimalformat eingeben. Bei einer Hexadezimalzahl müssen Sie den Buchstaben H hinter der betreffenden Zahl eingeben (Beispiel: 170H). Beginnt eine Hexadezimalzahl mit den Buchstaben A–F, muß ihr die Ziffer 0 vorangestellt werden (Beispiel: die Zahl A567H muß als 0A567H eingegeben werden). Erfordert ein Operator, z.B. .AND., eine Ganzzahl, schneidet das BASIC-Modul den Teil der Zahl nach dem Komma ab, um das Ganzzahlformat einzuhalten. Ganzzahlen werden wie folgt dargestellt:

[Ganzzahl]

Wichtig: Der SLC-500-Prozessor führt Operationen mit 16-Bit-Ganzzahlen im Bereich von -32768 bis 32767 aus. Überträgt das BASIC-Modul einen Wert an den Prozessor, der größer als 32767 ist, wird dieser Wert vom Prozessor als negative Zahl interpretiert.

Fließkommawerte

Im BASIC-Modul werden alle Zahlen als Fließkommawerte gespeichert. Fließkommawerte sind Zahlen, in denen das Dezimalkomma in Abhängigkeit von den signifikanten Ziffern einer bestimmten Zahl “fließt”. Der Prozessor berücksichtigt die Position des Dezimalkommas und speichert nur die signifikanten Ziffern eines Wertes, wodurch Speicherplatz eingespart wird.

Der folgende Zahlenbereich kann im BASIC-Modul verarbeitet werden.

+1E -127 to +0,99999999 +127

Es werden acht signifikante Ziffern verwendet. Die Zahlen werden intern gerundet, damit diese Genauigkeit eingehalten werden kann.

Backplane-Umwandlungsdaten

Das BASIC-Modul kommuniziert mit dem Prozessor über die E/A-Backplane SLC 500. Alle Daten werden im SLC-500-Format an die bzw. von der SLC-500-Steuerung übertragen. Folgende SLC-500-Formate werden verwendet:

- 16-Bit-Ganzzahl mit Vorzeichen (-32768 bis 32767)
- 16-Bit-Binärwert (0000000000000000 bis 1111111111111111)

Wichtig: Überträgt das BASIC-Modul einen Wert an den Prozessor, der größer als 32767 ist, wird dieser Wert vom Prozessor als negative Zahl interpretiert.

Variablen

Variablen, die einen eindimensionalen Ausdruck [ausdr] enthalten, werden als dimensionierte oder als Datenfeldvariablen bezeichnet. Variablen, die einen Buchstaben oder einen Buchstaben und eine Ziffer enthalten, sind Skalarvariablen. Alle in Kleinbuchstaben eingegebenen Variablen werden in Großbuchstaben umgewandelt. Sie werden wie folgt dargestellt:

[var]

Die Zuweisung von Variablen durch das BASIC-Modul erfolgt nach einem genau festgelegten Verfahren: Wird eine Variable zum ersten Mal verwendet, wird ihr vom BASIC-Modul ein bestimmter Speicherplatz (8 Bytes) zugeordnet. Diese Zuordnung kann nicht rückgängig gemacht werden, um den Speicherbereich anschließend einer anderen Variablen zuzuordnen. Bei der Ausführung einer Anweisung (Beispiel: >10 Q = 3) können Sie dem BASIC-Modul nicht mitteilen, daß die Variable Q nicht mehr exisitiert, um die dieser Variablen zugeordneten acht Speicherbytes wieder freizugeben. Der einer Variablen zugeordnete Speicherbereich kann jedoch mit einer CLEAR-Anweisung gelöscht werden. Mit dieser Anweisung wird sämtlicher, den Variablen zugeordneter Speicher wieder freigegeben. Variablen können für den späteren Gebrauch reserviert werden, um Speicherplatz zu sparen.

Wichtig: Das BASIC-Modul benötigt zum Auffinden einer Skalarvariablen weniger Zeit, da kein Ausdruck bewertet werden muß. Zur schnellstmöglichen Ausführung eines Programms sollten Sie eindimensionale Variablen nur im Bedarfsfall verwenden. Verwenden Sie Skalarvariablen als Zwischenvariablen, und ordnen Sie das Endergebnis einer dimensionierten Variablen zu. Häufig verwendete Variablen sollten zuerst definiert werden, da diese schnell auffindbar sind.

Variablennamen

Variablen repräsentieren numerische Werte oder Zeichenketten. Variablennamen können maximal acht Zeichen lang sein. Das BASIC-Modul vergleicht das erste und letzte Zeichen sowie die Zeichenanzahl eines Variablennamens mit dem ersten und letzten Zeichen und mit der Zeichenanzahl anderer Variablennamen, um zu überprüfen, ob der Variablenname nur einmal vergeben wurde. Die in einem Variablennamen zulässigen Zeichen sind Buchstaben, Ziffern und der Dezimalpunkt. Ferner sind bestimmte, zur Kennzeichnung des Variablentyps verwendete Zeichen, zulässig.

Eine Variable kann ein Buchstabe sein (z.B. **`A, X, oder I`**), gefolgt von:

- einem dimensionalen Ausdruck (z.B.: **`J(4), G(A+6), I(10*SIN(X))`**)
- einer Zahl und einem eindimensionalen Ausdruck (z.B. **`A1(8), P7(10*SIN(X)), W8(A + C`**)
- einer Ziffer (0 bis 9) oder einem Buchstaben (z.B.: **`AA, AC, XX, A1, X3, G8`**). Die folgenden Kombinationen sind jedoch nicht zulässig: **`CR, DO, IE, IF, IP, ON, PI, SP, TO, UI, UO`**

Wichtig: Reservierte Worte (d.h. Worte, die breits in BASIC-Funktionen oder -Anweisungen verwendet werden) dürfen nicht als Variablennamen verwendet werden.

Variablentypen

Typenbeschreibungszeichen geben an, um welchen Variablentyp es sich handelt. Das folgende Typenbeschreibungszeichen ist gültig:

Zeichen	Variablentyp
$	Zeichenkettenvariable

Der einzige weitere zulässige Variablentyp ist die Fließkommavariable, die keine Typenkennzeichnung erfordert.

Kapitel 3

Ausdrücke und Operatoren

In diesem Kapitel wird die Manipulation und/oder Bewertung von Ausdrücken und Anweisungen innerhalb eines BASIC-Programms bzw. in der Befehlszeile beschrieben und veranschaulicht. Die entsprechende Mnemonik ist in Tabelle 3.A aufgelistet.

Tabelle 3.A
Kapitelinhalt

Funktion	Mnemonik	Seite
Abruf und/oder Zuordnung des Wertes der freilaufenden Uhr	TIME	3-20
Abruf oder Zuordnung von Daten aus dem bzw. an den externen Datenspeicher des BASIC-Moduls	XBY( )	3-19
Addition	+	3-5
Division	/	3-5
Potenzierung	**	3-5
Multiplikation	*	3-5
Subtraktion	–	3-5
Logisches UND	.AND.	3-8
Logisches ODER	.OR.	3-8
Logisches Exklusiv-ODER	.XOR.	3-8
Direkte Kommunikation über PRT1	@	3-17
Direkte Kommunikation über PRT2	#	3-17

Ausdrücke und Operatoren

Ein Ausdruck ist ein logischer mathematischer Ausdruck, der Operatoren, Konstanten und Variablen enthält. Die folgenden acht Arten von Operatoren können in einem Ausdruck angewendet werden:

- arithmetische
- logische
- relationale
- trigonometrische
- funktionale
- logarithmische
- Zeichenketten
- Sonderfunktionen

Ausdrücke

Ausdrücke können einfach oder komplex sein.

Beispiel:

Einfacher Ausdruck: **`12*EXP(A)/100,H(1) + 55,`**

oder

komplexer Ausdruck: **`(SIN(A)*SIN(A)+COS(A)* COS(A)/2)`**

Eigenständige Variablen [var] und Konstanten [const] gehören auch zu den Ausdrücken. Ausdrücke werden wie folgt dargestellt:

[ausdr]

Operatoren

Operatoren führen eine bestimmte Operation mit Variablen oder Konstanten aus. Sie erfordern einen oder zwei Operanden. Zu den typischen Operatoren mit zwei Operanden gehören die ADDITION(+), SUBTRAKTION (–), MULTIPLIKATION (*) und DIVISION (/). Operatoren, die nur einen Operanden erfordern, heißen Einzeloperand-Operatoren und umfassen generell Funktionen wie SIN, COS und ABS.

Hierarchie der Operatoren

Die Hierarchie der Operatoren ist die Reihenfolge, in der die Operationen eines Ausdrucks ausgeführt werden. Sie können komplexe Ausdrücke erstellen, die nur wenige Klammern erfordern. Diese Hierarchie der Operatoren wird anhand der folgenden Gleichung erläutert:

4+3*2 = ?

In dieser Gleichung hat die Multiplikation Vorrang gegenüber der Addition. Somit wird (3*2) multipliziert und dann 4 addiert.

4+3*2 = 10

Bei der Abarbeitung eines Ausdrucks von links nach rechts wird eine Operation erst ausgeführt, wenn ein Operator mit niedrigerer oder gleicher Priorität gefunden wird. Im vorausgehenden Beispiel kann die Addition erst nach der Multiplikation ausgeführt werden, da die Multiplikation eine höhere Priorität besitzt. Verwenden Sie Klammern, wenn Sie über die Reihenfolge der Prioritäten im Zweifel sind, oder wenn die Lesbarkeit des Programms optimiert werden soll. Im folgenden ist die Priorität der Operatoren für das BASIC-Modul aufgeführt, wobei 1 die höchste und 9 die niedrigste Priorität darstellt:

1. Operatoren, die Klammern verwenden ( )

2. Potenzierung (**)

3. Negation (-)

4. Multiplikation (*) und Division (/)

5. Addition (+) und Subtraktion (-)

6. Relationale Ausdrücke (=, <>, >, >=, <, <=).

7. Logisches UND (.AND.)

8. Logisches ODER (.OR.)

9. Logisches XOR (.XOR.)

Arithmetische Operatoren

Das BASIC-Modul enthält einen vollständigen Satz arithmetischer Operatoren, die in zwei Gruppen unterteilt sind: Doppeloperand- und Einzeloperand-Operatoren.

Das allgemeine Format für alle Doppeloperand-Befehle lautet:

(ausdr) OP (ausdr), wobei OP einer der folgenden arithmetischen Operatoren ist:

Addition (+)

Mit dem Additionsoperator wird der erste Ausdruck zum zweiten Ausdruck addiert.

Beispiel: `>PRINT 3+2`

Ergebnis: `5`

Division (/)

Mit dem Divisionsoperator wird der erste Ausdruck durch den zweiten Ausdruck dividiert.

Beispiel: `>PRINT 100/5`

Ergebnis: `20`

Potenzierung (**)

Mit dem Potenzierungsoperator wird der erste Ausdruck um den zweiten Ausdruck potenziert. Der maximal zulässige Exponent ist 255.

Beispiel: `>PRINT 2**3`

Ergebnis: `8`

Multiplikation (*)

Mit dem Multiplikationsoperator wird der erste Ausdruck mit dem zweiten Ausdruck multipliziert.

Beispiel: `>PRINT 3*3`

Ergebnis: `9`

Subtraktion (–)

Mit dem Subtraktionsoperator wird der zweite Ausdruck vom ersten Ausdruck subtrahiert.

Beispiel: `>PRINT 9-6`

Ergebnis: `3`

Negation (–)

Mit dem Negationsoperator wird ein positiver Ausdruck in einen negativen Ausdruck umgewandelt.

Beispiel: `>PRINT -(9+4)`

Ergebnis: `-13`

Überlauf und Division durch Null

Wenn während der Bewertung eines Ausdrucks ein Überlauf, Unterlauf oder eine Division durch Null eintritt, erzeugt das BASIC-Modul Fehlermeldungen und schaltet in den Befehlsmodus zurück. Weitere Hinweise zur Eingrenzung dieser Fehler sind in Kapitel 8 unter “ONERR” enthalten.

Das höchste zulässige Ergebnis einer Rechenoperation is 0,99999999 E+127. Bei Überschreitung dieses Wertes erzeugt das BASIC-Modul die Fehlermeldung **`ERROR: ARITH. OVERFLOW`** und schaltet in den Befehlsmodus zurück.

Das kleinste zulässige Ergebnis einer Rechenoperation ist 0,99999999 E–128. Bei Unterschreitung dieses Wertes erzeugt das BASIC-Modul die Fehlermeldung **`ERROR: ARITH. UNDERFLOW`** und schaltet in den Befehlsmodus zurück.

Wenn Sie versuchen, eine Zahl durch Null zu dividieren, erzeugt das BASIC-Modul die Fehlermeldung **`ERROR: DIVIDE BY ZERO`** und schaltet in den Befehlsmodus zurück.

```
>10 PRINT 9/0
>20 PRINT “PROGRAM SHOULD NOT GET HERE.”

READY
>RUN

ERROR: DIVIDE BY ZERO – IN LINE  10

10     PRINT 9/0
----------------X
READY
>

>10 PRINT 9.9E126*(2)
>
```

```
READY
>RUN

ERROR: ARITH. OVERFLOW – IN LINE  10

10     PRINT 9.9E126*(2)
------------------------X
READY
>
```

Logische Operatoren

Das BASIC-Modul verfügt über einen vollständigen Satz logischer Operatoren, die in zwei Gruppen unterteilt sind: Doppeloperand- und Einzeloperand-Operatoren.

Das allgemeine Format für alle Doppeloperand-Befehle lautet:

(Ausdr) OP (Ausdr), wobei OP einer der folgenden arithmetischen Operatoren ist:

Diese Operatoren führen BITWEISE logische Operationen an Zahlen von 0 (0000H) bis einschließlich 65535 (0FFFFH) aus. Liegt das Argument außerhalb dieses Bereichs, erstellt das BASIC-Modul die Fehlermeldung **ERROR: BAD ARGUMENT** und schaltet in den Befehlsmodus zurück. Alle Dezimalstellen werden abgeschnitten und nicht gerundet. Verwenden Sie die folgende Tabelle für die Bitmanipulation von 16-Bit-Werten.

Tabelle 3.B
Bitmanipulation von 16-Bit-Werten

X	Y	X .AND.Y	X .OR.Y	X .XOR.Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

.AND.

Mit dem logischen Operator .AND wird eine logische AND-Verknüpfung zweier Ausdrücke durchgeführt.

Beispiel: `>PRINT 3.AND.2`

Ergebnis: `2`

.OR.

Mit dem logischen Operator .OR wird eine logische OR-Verknüpfung zweier Ausdrücke durchgeführt.

Beispiel: `>PRINT 1.OR.4`

Ergebnis: `5`

.XOR.

Mit dem logischen Operator .XOR. wird eine logische XOR-Verknüpfung zweier Ausdrücke durchgeführt.

Beispiel: `>PRINT 7.XOR.6`

Ergebnis: `1`

Relationale Operatoren

Relationale Ausdrücke enthalten die Operatoren =, < >, >, >=, < und <=. Im BASIC-Modul werden diese Operatoren gewöhnlich zur Prüfung einer Bedingung verwendet. Ist der relationale Ausdruck wahr, geben die relationalen Operatoren des BASIC-Moduls das Ergebnis 65535 (0FFFFH) aus. Ist der relationale Ausdruck falsch, wird als Ergebnis der Wert 0 ausgegeben. Das Ergebnis wird im Argumentstapel gespeichert. Hierdurch ist es möglich, das Ergebnis eines relationalen Ausdrucks anzuzeigen. Relationale Ausdrücke werden wie folgt dargestellt:

[rel Ausdr]

Beispiele:

```
>PRINT 1=0
0
>PRINT 1>0
65535
>PRINT A<>A
0
>PRINT A=A
65535
```

Mit den logischen Operatoren .AND., .OR. und .XOR. können relationale Ausdrücke verkettet werden, um eine komplexe Bedingung mit nur EINER Anweisung zu prüfen.

Beispiel:

```
>10 IF (A>E).AND.(A>C).OR.(A>D)THEN...
```

Zusätzlich können Sie den Operator NOT([Ausdr]) verwenden.

Beispiel:

```
>10 IF NOT(A>E).AND.(A>C)THEN...
```

Durch die Verkettung relationaler Ausdrücke mit Hilfe logischer Operatoren können bestimmte Bedingungen mit nur einer Anweisung geprüft werden.

Wichtig: Bei der Verwendung logischer Operatoren zur Verknüpfung relationaler Ausdrücke müssen Sie sicherstellen, daß die Operationen in der korrekten Reihenfolge ausgeführt werden. Verwenden Sie im Zweifelsfall Klammern.

Trigonometrische Operatoren

Das BASIC-Modul verfügt über einen vollständigen Satz trigonometrischer Operatoren. Hierbei handelt es sich um Einzeloperand-Operatoren.

SIN([Ausdr])

Mit dem Operator SIN wird der Sinus des Arguments ausgegeben. Das Argument wird in Radianten ausgedrückt. Die Berechnung erfolgt bis auf sieben signifikante Stellen. Das Argument muß zwischen ±200000 liegen.

Beispiele:

```
>PRINT SIN(PI/4)     >PRINT SIN(0)
.7071067             0
```

COS([Ausdr])

Mit dem Operator COS wird der Kosinus des Arguments errechnet. Das Argument wird in Radianten ausgedrückt. Die Berechnung erfolgt bis auf sieben signifikante Stellen. Das Argument muß zwischen ±200000 liegen.

Beispiele:

```
>PRINT COS(PI/4)     >PRINT COS(0)
.7071067             1
```

TAN([Ausdr])

Mit dem Operator TAN wird der Tangens des Arguments errechnet. Das Argument wird in Radianten ausgedrückt. Die Berechnung erfolgt bis auf sieben signifikante Stellen. Das Argument muß zwischen ±200000 liegen.

Beispiele:

```
>PRINT TAN(PI/4)     >PRINT TAN(0)
1                    0
```

ATN([Ausdr])

Mit dem Operator ATN wird der Arkustangens des Arguments errechnet. Das Ergebnis wird in Radianten angezeigt. Die Berechnung wird bis auf sieben signifikante Stellen ausgeführt. Das Ergebnis liegt zwischen -PI/2 (3,1415926/2) und PI/2.

Beispiele:

```
>PRINT ATN(PI)       >PRINT ATN(1)
1.2626272            .78539804
```

Hinweise zu den trigonometrischen Funktionen

Die Operatoren SIN, COS und TAN verwenden zur Berechnung der Funktion eine Taylor-Reihe. Diese Operatoren reduzieren das Argument auf einen Wert zwischen 0 und PI/2. Diese Reduzierung wird durch die folgende Gleichung erreicht:

```
reduziertes Argument=(anw arg/PI - INT(anw arg/PI) *PI
```

Das durch diese Gleichung reduzierte Argument liegt zwischen 0 und PI. Das reduzierte Argument wird anschließend geprüft, um festzustellen, ob es größer ist als **PI/2**. Ist es größer, wird es von **PI** subtrahiert, um den Endwert zu ermitteln. Anderenfalls entspricht das reduzierte Argument dem Endwert.

Obwohl dieses Verfahren der Winkelreduzierung die einfache und rationelle Erstellung des entsprechenden Arguments für eine Taylor-Reihe ermöglicht, weist es jedoch ein Genauigkeitsproblem auf. Dieses macht sich besonders bei großen Anwenderargumenten bemerkbar (z.B.: **`größer als 1000`**) und ist darauf zurückzuführen, daß die signifikanten Ziffern des Dezimalanteils (Bruchwert) des reduzierten Arguments im Ausdruck **`(anw arg/PI - INT (anw arg/PI))`** verlorengehen. Generell sollten für trigonometrische Funktionen möglichst kleine Argumente verwendet werden.

Funktionale Operatoren

Das BASIC-Modul verfügt über einen vollständigen Satz funktionaler Einzeloperand-Operatoren.

ABS([Ausdr])

Mit dem ABS-Operator wird der absolute Wert des Ausdrucks errechnet.

Beispiele:

```
>PRINT ABS(5)        >PRINT ABS(-5)
5                    5
```

NOT([Ausdr])

Mit dem NOT-Operator wird ein 16-Bit-Einerkomplement des Ausdrucks ausgegeben. Der Ausdruck muß eine gültige Ganzzahl sein (z.B.: **`zwischen 0 und einschließlich 65535 (0FFFFH)`**). Werte, die keine Ganzzahlen sind, werden abgeschnitten und nicht gerundet.

Beispiele:

```
>PRINT NOT(65000)    >PRINT NOT(0)
535                  65535
```

INT([Ausdr])

Mit dem INT-Operator wird der Ganzzahlteil des Ausdrucks ausgegeben.

Beispiele:

```
>PRINT INT(3.7)      >PRINT INT(100.876)
3                    100
```

PI

PI ist eine im BASIC-Modul gespeicherte Konstante, deren Wert 3,1415926 beträgt.

SGN([Ausdr])

Mit dem SNG-Operator wird der Wert +1 ausgegeben, wenn das Argument größer als Null ist. Wenn das Argument gleich Null ist, wird der Wert 0 ausgegeben, und wenn das Argument kleiner als Null ist, wird der Wert –1 ausgegeben.

Beispiele:

```
>PRINT SGN(52)   >PRINT SGN(0)   >PRINT SGN(-8)
1                0               -1
```

SQR([Ausdr])

Mit dem SQR-Operator wird die Quadratwurzel des Arguments ausgegeben. Das Argument darf nicht kleiner als Null sein.

Beispiele:

```
>PRINT SQR(9)   >PRINT SQR(45)   >PRINT SQR(100)
3               6.7082035        10
```

RND

Mit dem RND-Operator wird eine Pseudozufallszahl zwischen 0 und 1 ausgegeben. Auf der Grundlage eines 16-Bit-Keims erzeugt dieser Operator 65536 Pseudozufallszahlen, bevor er den Vorgang wiederholt. Die generierten Zahlen liegen im Bereich von 0/65535 bis einschließlich 65535/65535.

Wichtig: Im Gegensatz zu den meisten BASIC-Programmiersprachen erfordert der RND-Operator des BASIC-Moduls kein Argument bzw. Formalargument. Wird nach dem RND-Operator ein Argument gesetzt, erscheint die Fehlermeldung einer ungültigen Syntax.

Beispiele:

```
>PRINT RND
.26771954
```

Logarithmische Operatoren

Das BASIC-Modul verfügt über einen vollständigen Satz logarithmischer Einzeloperand-Operatoren.

LOG([Ausdr])

Mit dem LOG-Operator wird der natürliche Logarithmus des Arguments ausgegeben. Das Argument muß größer als 0 sein. Diese Berechnung wird bis auf sieben signifikante Stellen ausgeführt.

Beispiele:

```
>PRINT LOG(12)          >PRINT LOG(EXP(1))
2.484906                1
```

Der 10er Logarithmus kann mit dem folgenden Ausdruck errechnet werden:

$$\log_{10}(x)=\log(x)/\log(10)$$

Es handelt sich hier um einen natürlichen Logarithmus.

EXP([Ausdr])

Mit dem EXP-Operator kann der Wert “e” (2,7182818) mit dem Argument potenziert werden.

Beispiele:

```
>PRINT EXP(1)           >PRINT EXP(LOG(2))
2.7182818               2
```

Zeichenkettenoperatoren

Zur Manipulierung von Zeichenketten stehen im BASIC-Modul zwei Operatoren zur Verfügung: ASC( ) und CHR( ).

ASC([Ausdr])

Mit dem ASC-Operator wird der Ganzzahlwert des in Klammern gesetzten ASCII-Zeichens ausgegeben.

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PRINT ASC(a)
>20 PRINT ASC(A)


READY
>RUN

 65
 65

READY
>
```

In der Dezimalschreibweise wird das ASCII-Zeichen “A” durch die Zahl 65 und das ASCII-Zeichen “a” durch die Zahl 97 dargestellt. Alle nicht zwischen Anführungszeichen eingegebenen ASCII-Zeichen werden jedoch vom BASIC-Modul in Großbuchstaben umgewandelt. ASCII-Sonderzeichen, deren Dezimalwert größer als 127 ist, sollten nicht verwendet werden.

Darüber hinaus ermöglicht der ASCI-Operator die Bewertung einzelner Zeichen einer definierten ASCII-Zeichenkette.

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) =“DIES IST EINE ZEICHENKETTE”
>20 PRINT $(1)
>30 PRINT ASC($(1),1)
>40 END


READY
>RUN

DIES IST EINE ZEICHENKETTE
68

READY
>
```

Wenn Sie den ASC-Operator in der oben beschriebenen Weise verwenden, kennzeichnet $([ausdr]) die Zeichenkette, auf die Sie gerade zugreifen. Mit dem Ausdruck nach dem Komma wird ein einzelnes Zeichen der Zeichenkette gewählt. Im o.g. Beispiel ist dies das erste Zeichen der Zeichenkette. Das ASCII-Zeichen “D” wird in der Dezimalschreibweise durch die Zahl 68 dargestellt. Die Zeichenkettenposition 0 ist unzulässig.

Beispiel:

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1)=“ABCDEFGHIKJL”
>20 FOR X = 1 TO 12
>30 PRINT ASC($(1),X),
>40 NEXT X
>50 END


READY
>RUN

65  66  67  68  69  70  71  72  73  75  74  76
READY
>
```

Die im vorangehenden Beispiel enthaltenen Zahlen stellen die ASCII-Zeichen A bis L dar.

Mit dem ASC-Operator können außerdem einzelne Zeichen einer spezifizierten Zeichenkette geändert werden.

Generell können Sie mit dem ASC-Operator einzelne Zeichen einer Zeichenkette manipulieren. Mit Hilfe eines einfachen Programms kann überprüft werden, ob zwei Zeichenketten identisch sind.

Beispiel:

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) = “ABCDEFGHIJKL”
>20 PRINT $(1)
>30 ASC($(1),1) = 75 : REM DECIMAL EQUIVALENT OF K
>40 PRINT $(1)
>50 ASC($(1),2) = ASC($(1),3)
>60 PRINT $(1)

READY
>RUN

ABCDEFGHIJKL
KBCDEFGHIJKL
KCCDEFGHIJKL

READY
>
```

CHR([Ausdr])

Mit dem CHR-Operator kann ein numerischer Ausdruck in ein ASCII-Zeichen umgewandelt werden.

Beispiel:

```
>PRINT CHR(65)
A
```

Ähnlich wie der ASC-Operator wählt der CHR-Operator einzelne Zeichen einer spezifizierten ASCII-Zeichenkette.

Beispiel:

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) = “The BASIC Module”
>20 FOR I = 1 TO 16 : PRINT CHR($(1),I),: NEXT I


READY
>RUN

Das BASIC-Modul
READY
>
```

Im oben aufgeführten Beispiel haben die in Klammern gesetzten Ausdrücke nach dem CHR-Operator dieselbe Bedeutung wie die Ausdrücke des ASC-Operators.

Im Gegensatz zum ASC-Operator kann dem CHR-Operator *kein* Wert zugeordnet werden. Eine Anweisung wie z.B. CHR ($(1),1)= H ist UNGÜLTIG und erzeugt die Fehlermeldung **`ERROR: BAD SYNTAX`**, wodurch veranlaßt wird, daß das BASIC-Modul in den Befehlsmodus zurückschaltet. Verwenden Sie den ASC-Operator, um einen Wert in einer Zeichenkette zu ändern, oder rufen Sie die CALL-Routine, die Zeichenketten unterstützt, um eine Zeichenkette innerhalb einer anderen Zeichenkette zu ersetzen.

Wichtig: Die CHR-Funktion darf nur in einer PRINT-Anweisung verwendet werden. Der CHR-Operator kann nicht in einer DATA-Anweisung eingesetzt werden.

Sonderfunktionsoperatoren

Das BASIC-Modul verfügt über einen vollständigen Satz von Sonderfunktionsoperatoren, die die E/A-Hardware und Speicheradressen des BASIC-Moduls manipulieren.

und @

Mit den Operatoren # und @ wird die Kommunikation festgelegt. Bei Programmierung des Operators @ erfolgt die Kommunikation über den PRT1-Port, und bei Programmierung des Operators # erfolgt die Kommunikation über den PRT2-Port. Ist keiner dieser beiden Operatoren vorhanden, sollte die Kommunikation über einen Programmierport (PRT1 oder DH485) erfolgen.

Beispiel für den Operator #:

```
>10 A = GET#
```

Ergebnis: Das nächste Zeichen im PRT2-Eingangspuffer wird der Variablen A zugeordnet.

Beispiel für den Operator @:

```
>10 A = GET@
```

Ergebnis: Das nächste Zeichen im PRT1-Eingangspuffer wird der Variablen A zugeordnet.

EOF

Mit dem EOF-Operator wird das System vor der Ausführung einer Eingangsanweisung bzw. funktion auf einen leeren Eingangspuffer überprüft. Dadurch wird verhindert, daß Eingangsanweisungen eine unbestimmte Zeit lang leere Eingangszwischenspeicher ablesen. Mit der Anweisung EOF# können Sie überprüfen, ob für den PRT2-Port ein leerer Eingangspuffer vorhanden ist. Die Anweisung EOF@ ermöglicht die Durchführung dieser Prüfung für den PRT1-Port.

Beispiel:

```
>10 REM EXAMPLE PROGRAM
>20 IF (NOT(EOF)) THEN A=GET
>30 REM IF BUFFER NOT EMPTY, READ SINGLE CHARACTER
```

FREE

Der Systemsteuerwert FREE gibt an, wieviele Bytes dem Anwender im RAM-Speicher zur Verfügung stehen. Befindet sich das gewählte Programm im RAM-Speicher, gilt die folgende Relation:

```
FREE = MTOP - LEN - 511
```

LEN

Der Systemsteuerwert LEN gibt an, wieviele Speicherbytes das aktuelle Programm belegt. Es handelt sich hierbei um die Länge des Programms, wobei die Größe des Zeichenkettenspeichers bzw. die Speicherbelegung durch Variablen und Datenfelder nicht enthalten sind. LEN kann nur gelesen werden, d.h. Sie können LEN keinen Wert zuordnen. Bei einem NULL-Programm (z.B.: `kein Programm`) wird für LEN der Wert 1 ausgegeben. Der Wert 1 kennzeichnet das Ende des Programmfilezeichens.

Wichtig: Das BASIC-Modul erfordert für die Systemsteuerwerte kein Formalargument.

MTOP

Mit dem MTOP-Operator wird die zuletzt gültige RAM-Speicheradresse, die dem BASIC-Modul zur Verfügung steht, abgerufen. Nach der Rücksetzung ermittelt das BASIC-Modul die Größe des externen Speichers und ordnet dem Systemsteuerwert MTOP die letzte gültige Speicheradresse zu. Das Modul belegt keinen externen RAM-Speicher, der diesen zugeordneten Wert überschreitet. Sofern dieser Wert nicht durch Aufruf 77 geändert wurde, ist die letzte gültige BASIC-Adresse 5FFFH (24575).

Beispiele:

```
>PRINT MTOP          or        PH0.MTOP
24575                          5FFFH
```

CBY([Ausdr])

Mit dem CBY-Operator werden Daten aus dem Programm oder der Code-Speicheradresse des BASIC-Moduls abgerufen. Sie können CBY keinen Wert zuweisen, sondern nur den ausgegebenen Wert lesen. Das Argument für den CBY-Operator muß eine gültige Ganzzahl zwischen 0 und 65535 (0FFFFH) sein. Anderenfalls erscheint die Fehlermeldung eines ungültigen Arguments.

Wichtig: Die unsachgemäße Verwendung dieses Operators kann Funktionsstörungen des BASIC-Moduls verursachen.

Beispiel:

```
A = CBY(1000)
```

Ergebnis: Der Wert der Programm- oder Code-Speicheradresse 1000 wird der Variablen A zugewiesen.

DBY([Ausdr])

Mit dem DBY-Operator werden Daten aus dem internen Datenspeicher des BASIC-Moduls abgelesen bzw. diesem zugewiesen. Sowohl der Wert als auch das Argument des DBY-Operators müssen zwischen 0 und einschließlich 255 liegen, weil im BASIC-Modul nur 256 interne Speicheradressen verfügbar sind und ein Byte immer nur eine Größe von 0 bis einschließlich 255 darstellen kann.

Wichtig: Die unsachgemäße Verwendung dieses Operators kann Funktionsstörungen im BASIC-Modul verursachen.

Beispiel:

```
A = DBY(B)
```

Ergebnis: Der Wert in der internen Speicheradresse B wird der Variablen A zugeordnet. B muß ein Wert zwischen 0 und 255 sein.

```
DBY(250) = CBY(1000)
```

Ergebnis: Der Wert in der Programm- bzw. Code-Speicheradresse 100 wird der internen Speicheradresse 250 zugeordnet.

XBY([Ausdr])

Mit dem XBY-Operator werden Daten aus dem externen Datenspeicher des BASIC-Moduls abgelesen bzw. diesem zugewiesen. Das Argument für den XBY-Operator muß eine gültige Ganzzahl zwischen 0 und 65535 (0FFFFH) sein. Der dem XBY-Operator zugewiesene Wert muß zwischen 0 und einschließlich 255 liegen. Anderenfalls wird die Fehlermeldung eines ungültigen Arguments angezeigt.

Wichtig: Die unsachgemäße Verwendung dieses Operators kann Funktionsstörungen im BASIC-Modul verursachen.

Beispiel:

```
A = XBY(0F000H)
```

Ergebnis: Der Wert in der externen Speicheradresse 0F00H wird der Variablen A zugeordnet.

```
XBY(4000H) = DBY(100)
```

Ergebnis: Der Wert in der internen Speicheradresse 100 wird der externen Speicheradresse 4000H zugewiesen.

TIME

Mit dem TIME-Operator wird die Zuweisung eines Wertes an die freilaufende Uhr im BASIC-Modul bzw. der Abruf eines Wertes aus dieser ermöglicht. Nach der Rücksetzung ist die Zeit 0. Mit der Anweisung CLOCK1 wird die freilaufende Uhr aktiviert. Danach wird der Sonderfunktionsoperator TIME alle fünf Millisekunden inkrementiert. Die Zeiteinheit ist die Sekunde.

Wenn dem TIME-Operator mit der LET-Anweisung ein Wert zugeordnet wird (z.B.: **`TIME=100`**), wird nur der Ganzzahlanteil von TIME geändert.

Beispiel:

```
>CLOCK1             (freilaufende Uhr aktivieren)

>CLOCK0             (freilaufende Uhr deaktivieren)

>PRINT TIME         (Anzeige des TIME-Operators)
3.315

>TIME = 0           (TIME-Operator = 0)

>PRINT TIME         (Anzeige des TIME-Operators)
.315                (nur der Ganzzahlanteil wird geändert)
```

Der Bruchteil des TIME-Operators kann durch Manipulieren des Inhalts der internen Speicheradresse 71 (47H) geändert werden. Hierzu kann eine DBY-Anweisung (71) verwendet werden. Beachten Sie bitte, daß jeder Zählvorgang in der internen Speicheradresse 71 (47H) durch den TIME-Operator fünf Millisekunden beträgt.

Fortsetzung des Beispiels:

```
>DBY(71) = 0        (Bruchteil des TIME-Operators = 0)

>PRINT TIME
0

>DBY(71) = 3        (Bruchteil des TIME-Operators = 3*5 ms
= 15 ms)

>PRINT TIME
1.5 E-2
```

Bei der Zuordnung eines Wertes wird nur der Ganzzahlanteil des TIME-Operators geändert. Auf diese Weise können Sie genaue Zeitintervalle definieren. Beispiel: Es soll eine exakte 12-Stunden-Uhr erstellt werden. 12 Stunden entsprechen 43200 Sekunden, deshalb müssen Sie die Anweisung ONTIME 43200, [zeil nr] verwenden. Im Falle eines TIME-Interrupts wird die Anweisung TIME0 ausgeführt, dem Millisekundenzähler wird jedoch kein neuer Wert zugeordnet. Überschreitet die Interrupt-Verzögerung fünf Millisekunden, bleibt die Taktgenauigkeit erhalten.

Kapitel 4

BASIC-Befehle

In diesem Kapitel sind die Worte und Ausdrücke beschrieben, die vom BASIC-Programm bzw. von der Befehlszeile aus verschiedene Funktionen veranlassen. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 4.A enthalten.

Tabelle 4.A
Kapitelinhalt

BRKPNT

Funktion:

Mit dem BRKPNT-Befehl kann ein Programmstopp in der jeweils spezifizierten Zeile gesetzt werden. Die Programmausführung wird unmittelbar vor der im BRKPNT-Befehl spezifizierten Zeilennummer gestoppt. Bei Eingabe der Zeilennummer 0 wird der Fixpunkt deaktiviert. Wenn der Fixpunkt erreicht wird, können Sie Variablen mit Hilfe von Zuordnungsanweisungen überprüfen. Die Programmausführung wird mit dem CONT-Befehl fortgesetzt. Nachdem der Fixpunkt erreicht wurde, wird er deaktiviert. Um das Programm an derselben Stelle zweimal zu stoppen, muß der Fixpunkt zweimal gesetzt werden. Der BRKPNT-Befehl ist nur bei Programmen, die vom RAM-Speicher aus abgearbeitet werden, anwendbar. Ein vom ROM-Speicher aus ausgeführtes Programm wird mit diesem Befehl nicht gestoppt.

Syntax:

BRKPNT[Zeil-Nr]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  D=0 : SU=0 : AV=0
>20  REM GET 100 DATUM POINTS
>30  FOR I=1 TO 100
>40  REM GET ANOTHER DATUM
>50  GOSUB 140
>60  REM SUM THE DATA
>70  GOSUB 170
>80  NEXT I
>90  REM AVERAGE THE DATA
>100 GOSUB 200
>110 REM PRINT RESULT
>120 PRINT “THE AVERAGE VALUE IS ”,AV
>130 END
>140 REM THIS SUBROUTINE GENERATES RANDOM DATA
>150 D=RND
>160 RETURN
>170 REM THIS SUBROUTINE SUMS THE DATA
>180 SU=SU+D
>190 RETURN
>200 REM THIS SUBROUTINE AVERAGES THE DATA
>210 AV=SU/I
>220 RETURN

READY
>BRKPNT 160

Breakpoint enabled.

READY
>RUN

STOP – IN LINE  160
READY
>PRINT D,SU,AV
 .86042573  0  0

>D = .5

>PRINT D,SU,AV
 .5  0  0

>CONT

THE AVERAGE VALUE IS  .48060383

READY
>
```

CONT

Funktion:

Mit dem CONT-Befehl wird die Ausführung eines durch **[CTRL–C]**, den BRKPNT-Befehl oder eine STOP-Anweisung angehaltenen Programms fortgesetzt. Wird ein Programm durch die Eingabe von **[CTRL–C]** auf dem Terminal oder durch Ausführung einer STOP-Anweisung angehalten, können Sie die Ausführung des Programms durch Eingabe von CONT fortsetzen. Wenn Sie **[CTRL–C]** drücken, während diese Tastenkombination aktiviert ist, wird die Programmausführung gestoppt. Die Fortsetzung erfolgt mit dem CONT-Befehl. Zwischen der Programmunterbrechung und der Wiederaufnahme können Sie Variablenwerte anzeigen lassen und diese Werte ändern. Wird das Programm jedoch während einer Unterbrechung oder nach einem Fehler modifiziert, ist die Fortsetzung mit CONT nicht möglich.

Wichtig: Mit **[CTRL–C]** werden alle Eingangs- und Ausgangspuffer gelöscht.

Syntax:

CONT

Beispiel:

```
>NEW

>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 10000
>20 PRINT I
>30 NEXT I
>40 END

READY
>RUN
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10 [[Ctrl-C] hier eingeben]

STOP – IN LINE 15
READY
>CONT

20
21
22
```

Control C

Funktion:

Mit dem Befehl **[CTRL-C]** wird die Abarbeitung des aktuellen Programms unterbrochen und der Befehlsmodus aufgerufen. In einigen Fällen können Sie die Programmausführung mit dem CONT-Befehl fortsetzen. Weitere Hinweise hierzu finden Sie in der Beschreibung des CONT-Befehls.

Wichtig: Mit **[CTRL-C]** werden alle Eingangs- und Ausgangspuffer gelöscht.

Syntax:

[**CTRL-C**]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 10000
>20 PRINT I
>30 NEXT I
>40 END
>RUN
  1
  2
  3
  4
  5 - (CONTROL C hier eingeben)

STOP - IN LINE 20

READY
>PRINT I
  27

>I =10

>CONT

  10
  11
  12
```

Beachten Sie bitte, daß nach der Eingabe von **[CTRL-C]** und der Ausgabe von I dessen Wert 27 ist. Dieser Wert wird mehrmals inkrementiert, bevor **[CTRL-C]** zur Programmunterbrechung führt.

CALL 18 - Aktivierung der Unterbrechungsfunktion Control-C

Funktion:

Durch die Ausführung von CALL 18 in einem BASIC-Programm oder von der Befehlszeile aus wird die Unterbrechungsfunktion **[CTRL-C]** reaktiviert.

Wichtig: Wenn **[CTRL-C]** deaktiviert ist, kann die Programmausführung nicht mit einem BASIC-Befehl gestoppt werden. Durch das Aus- und erneute Einschalten der Versorgungsspannung wird die Funktion **[CTRL-C]** erneut aktiviert, bis sie im Programm wieder deaktiviert wird. Um die Programmausführung zu stoppen, müssen Sie die Versorgungsspannung aus- und wieder einschalten und vor der Abarbeitung der Zeile, mit der **[CTRL-C]** deaktiviert wird, **[CTRL-C]** eingeben.

Syntax:

CALL 18

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 19
.
>90 CALL 18
```

CALL 19 - Deaktivierung der Unterbrechungsfunktion Control-C

Funktion:

Durch die Ausführung von CALL 19 in einem BASIC-Programm oder von der Befehlszeile aus wird die Unterbrechungsfunktion **[CTRL-C]** deaktiviert.

Bei der Ausführung von CALL 19 wird die Unterbrechungsfunktion **[CTRL-C]** sowohl für den LIST- als auch für den RUN-Modus deaktiviert. Wenn die Funktion **[CTRL-C]** von der Befehlszeile aus deaktiviert wurde, wird sie durch Aus- und erneutes Einschalten der Versorgungsspannung wieder aktiviert.

Wichtig: **[CTRL-C]** ist vorgabemäßig aktiviert.

Syntax:

CALL 19

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 19
```

Control S

Funktion:

Mit dem Befehl **[CTRL–s]** wird das Durchblättern eines BASIC-Programms während der Ausführung eines LIST-Befehls unterbrochen. Bei der Ausführung eines Programms wird mit **[CTRL–s]** die Ausgabe des übertragenden Ports gestoppt. In diesem Fall gilt für XOFF (**[CTRL–s]**) folgendes:

Wichtig: **[CTRL–s]** ist nur dann funktionsfähig, wenn die Software-Quittierung über den Programmierport aktiviert wurde.

- XOFF gilt nur für PRINT-Anweisungen.
- Wird der Befehl während einer PRINT-Anweisung empfangen, wird die Datenausgabe sofort unterbrochen, die Programmabarbeitung wird jedoch fortgesetzt.
- Wird der Befehl zu einem anderen Zeitpunkt empfangen, wird die Programmabarbeitung bis zur nächsten PRINT-Anweisung fortgesetzt, und die Datenausgabe wird zu diesem Zeitpunkt unterbrochen. Das Programm füllt den Ausgangspuffer, bis dieser voll ist.
- XON (**[CTRL–Q]**) ist für die Fortsetzung der Datenausgabe erforderlich.

Syntax:

[**CTRL-S**]

Beispiel:

```
> LIST
1  REM EXAMPLE PROGRAM
10 A = 1
20 DO
[CTRL-S]
.
.
.
[CTRL-Q]
30 A = A+1
40 PRINT A
50 WHILE A < 20

READY
>
```

In diesem Beispiel wird die Ausgabe mit der Tastenkombination [**CTRL-S**] unterbrochen und mit [**CTRL-Q**] fortgesetzt.

Control Q

Funktion:

Mit **[CTRL–Q]** wird ein durch **[CTRL–s]** unterbrochener LIST-Befehl bzw. PRINT-Ausgang fortgesetzt.

Syntax:

[**CTRL-Q**]

Beispiel:

```
> LIST
1  REM EXAMPLE PROGRAM
10 A = 1
20 DO
[CTRL-S]
.
.
.
[CTRL-Q]
30 A = A+1
40 PRINT A
50 WHILE A < 20

READY
>
```

In diesem Beispiel wird die Ausgabe durch die Tastenkombination [**CTRL-S**] unterbrochen und durch [**CTRL-Q**] fortgesetzt.

EDIT

Funktion:

Mit dem EDIT-Befehl wird der BASIC-Zeileneditor aufgerufen. Mit diesem Editor kann jeweils eine Zeile des aktuellen, im RAM-Speicher enthaltenen Programms editiert werden. Die Funktionen des BASIC-Editors sind in Tabelle 4.B aufgeführt.

Tabelle 4.B
Funktionen des BASIC-Editors

Operation	Funktion	Tastenanschläge
Verschieben	Verschiebung des Cursors nach rechts und links	[**Leertaste**] - positioniert den Cursor um ein Zeichen nach rechts [**Rücktaste**] - positioniert den Cursor um ein Zeichen nach links
Ersetzen	Ersetzen des Zeichens, das sich an der aktuellen Cursorposition befindet	Drücken Sie die Taste des Zeichens, welches das Zeichen an der aktuellen Cursorposition ersetzen soll.
Einfügen	Einfügen von Text an der Cursorposition **Wichtig**: Bei Verwendung der Einfügefunktion wird der Text auf der rechten Seite des Cursors ausgeblendet und erscheint erst wieder bei der erneuten Eingabe von [**Ctrl-A**]. Die Maximallänge einer Zeile beträgt 79 Zeichen.	[**Ctrl-A**] **Wichtig:** Zur Beendung der Einfügefunktion die Tastenkombination [**Ctrl-A**] erneut drücken
Löschen	Löschen des Zeichens an der Cursorposition	[**Ctrl-D**]
Beenden	Verlassen des Editors. Änderungen werden je nach Tastenkombination gespeichert oder nicht gespeichert.	[**Ctrl-Q**] - Verlassen des Editors. Die alte Zeile wird durch die editierte Zeile ersetzt. [**Ctrl-C**] - Verlassen des Editors. Die in der Zeile vorgenommenen Änderungen werden nicht gespeichert.
Kopieren	Kopieren der aktuellen Zeile und Einfügen direkt hinter der aktuellen Zeile. Der Cursor wird auf dem ersten Zeichen der neuen Zeile positioniert.	[**RETURN**]

Syntax:

EDIT

Beispiel:

```
>EDIT 150
```

Ergebnis: Die Programmzeile 150 wird angezeigt und kann nun editiert werden.

ERASE

Funktion:

Mit dem ERASE-Befehl wird das zuletzt mit dem PROG-Befehl im EEPROM gespeicherte BASIC-Programm gelöscht.

Syntax:

ERASE

Beispiel:

```
>ERASE

>ROM 13 ERASED
```

Ergebnis: Das letzte im EEPROM gespeicherte Programm (in diesem Beispiel ROM 13) wird gelöscht.

IDLE

Funktion:

Mit dem IDLE-Befehl wird das BASIC-Modul zwangsweise in den “Warte-auf-Interrupt”-Modus umgeschaltet. Die Programmabarbeitung wird so lange angehalten, bis eine ONTIME-Bedingung erkannt wird. Der ONTIME-Interrupt muß vor der Ausführung des IDLE-Befehls aktiviert werden, anderenfalls schaltet das BASIC-Modul in eine endlose Warteschleife. Wurde **[CTRL-C]** aktiviert, können Sie damit den IDLE-Befehl verlassen.

Syntax:

IDLE

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 TIME = 0
>20 CLOCK1
>30 ONTIME 2,70
>40 IDLE
>50 PRINT “END OF TEST!!!”
>60 END
>70 PRINT “TIMER INTERRUPT AT - ”,TIME,“SECONDS.”
>80 RETI

READY
>RUN

TIMER INTERRUPT AT - 2.005 SECONDS.
END OF TEST!!!
```

LIST

Funktion:

Mit dem LIST-Befehl kann das Programm auf dem Terminal angezeigt werden. Hinter der Zeilennummer sowie vor und hinter den Anweisungen werden Leerstellen eingefügt, um das Debugging von BASIC-Programmen zu erleichtern. Die Anzeige eines Programms auf dem Terminal kann durch **[CTRL–C]** jederzeit beendet werden. Mit den Befehlen **[CTRL–s]** und **[CTRL–Q]** kann die Anzeige unterbrochen und fortgesetzt werden.

Wichtig: Mit dem Befehl [**CTRL-C**] wird die Anzeige beendet, wenn dieser aktiviert ist. Mit dem Befehl [**CTRL-S**] wird die Anzeige unterbrochen, sofern die Software-Quittierung aktiviert ist.

Syntax:

LIST [Zeil-Nr]

LIST [Zeil-Nr] – [Zeil-Nr]

Im ersten Fall wird das Programm von der spezifizierten Zeilennummer [Zeil-Nr] bis zum Ende ausgegeben. Im zweiten Fall wird das Programm von der ersten spezifizierten Zeilennummer [Zeil-Nr] bis zur zweiten spezifizierten Zeilennummer [Zeil-Nr] ausgegeben.

Wichtig: Die beiden Zeilennummern müssen durch einen Bindestrich (–) getrennt werden.

Beispiel:

```
>LIST
1  REM EXAMPLE PROGRAM
10 PRINT “LOOP PROGRAM”
20 FOR I = 1 TO 3
30 PRINT I
40 NEXT I
50 END

READY
>LIST 30
1  REM EXAMPLE PROGRAM
30 PRINT I
40 NEXT I
50 END

READY
>LIST 20-40
1  REM EXAMPLE PROGRAM
20 FOR I = 1 TO 3
30 PRINT I
40 NEXT I
```

LIST@

Funktion:

Mit dem LIST@-Befehl wird das Programm auf dem an PRT1 angeschlossenen Gerät ausgegeben. Alle Hinweise zum LIST-Befehl gelten auch für den LIST@-Befehl. Sie müssen die Portparameter für PRT1 entsprechend dem Ausgabegerät konfigurieren. Diese Parameter (Baudrate, Parität, Stoppbits usw.) können mit dem MODE-Befehl eingestellt werden.

Syntax:

LIST@

Beispiele:

```
LIST@ :REM LISTS ALL LINES IN THE PROGRAM

LIST@10-20 :REM LISTS LINES 10 THROUGH 20
```

LIST#

Funktion:

Mit dem LIST#-Befehl wird das Programm auf dem an PRT2 angeschlossenen Gerät ausgegeben. Alle Hinweise zum LIST-Befehl gelten auch für den LIST#-Befehl. Sie müssen die Portparameter für PRT2 entsprechend dem Ausgabegerät konfigurieren. Diese Parameter (Baudrate, Parität, Stoppbits usw.) können mit dem MODE-Befehl eingestellt werden.

Syntax:

LIST#

Beispiel:

Siehe LIST@.

MODE

Funktion:

Mit dem MODE-Befehl werden die Portparameter für PRT1, PRT2 und DH485 spezifiziert. Die Portparameter für PRT1 und PRT2 sind in Tabelle 4.C aufgeführt.

Tabelle 4.C
Portparameter für PRT1 und PRT2

Portparameter	Einstellungen	Vorgabe-einstellung
Baudrate	300, 600, 1200, 2400, 4800, 9600, 19200	1200
Arg1 (Parität)	keine (N), gerade (E), ungerade (O)	N
Arg2 (Anzahl der Datenbits)	7 oder 8	8
Arg3 (Anzahl der Stoppbits)	1 oder 2	1
Arg4 (Handshake-Quittierung)	keine Handshake-Quittierung (N) Software-Handshake-Quittierung (S) Hardware-Handshake-Quittierung (H) Hardware- und Software-Handshake-Quittierung (B)	S
Arg5 (Speichertyp)	Datensicherung im Anwender-ROM und -RAM (E) Datensicherung im batteriegesicherten RAM (R)	R

Wichtig: Wird ein Argument (ausgenommen Portbezeichnung und Baudrate) nicht spezifiziert, wird dieses auf den zuletzt spezifizierten Wert eingestellt.

Tabelle 4.D enthält eine Auflistung der Portparameter für DH485.

Tabelle 4.D
Portparameter für DH485

Portparameter	Einstellungen	Vorgabe-einstellung
Baudrate	300, 600, 1200, 2400, 4800, 9600, 19200	19200
Arg1 (Adresse des Host-Netzknotens)	0 bis 31	0
Arg2 (Adresse des Modulnetzknotens)	1 bis 31	1
Arg3 (höchste Netzknotenadresse)	1 bis 31	31
Arg4 (nicht belegt)		
Arg5 (Speichertyp)	Datensicherung im Anwender-ROM und -RAM (E) Datensicherung im batteriegestützten RAM (R)	R

Wichtig: Wird ein Argument (ausgenommen Portbezeichnung) nicht definiert, wird dieses auf den zuletzt spezifizierten Wert eingestellt.

Syntax:

MODE(Portbezeichnung, Baudrate, Arg1, Arg2, Arg3, Arg4, Arg5)

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 MODE(DH485,19200,0,1,2,,R)
.
.
.
>25 MODE(PRT1,1200,N,8,,,)
```

Wichtig: Wird MODE als Anweisung verwendet, kann die Speichertyp-Option E nicht verwendet werden.

NEW

Funktion:

Mit dem NEW-Befehl wird das derzeit im RAM-Speicher enthaltene Programm gelöscht. Außerdem werden alle Variablen auf NULL gesetzt und alle Zeichenketten sowie alle durch BASIC veranlaßten Interrupts gelöscht. Die freilaufende Uhr, Zeichenkettenzuordnungen und die internen Stapelzeiger sind hiervon nicht betroffen. Mit dem NEW-Befehl können ein im RAM gespeichertes Programm und alle Variablen gelöscht werden.

Syntax:

NEW

Beispiel:

```
>NEW

>LIST

>READY
>
```

NULL

Funktion:

Mit dem NULL-Befehl können Sie feststellen, wieviele NULL-Zeichen (00H) vom BASIC-Modul nach einem Wagenrücklauf in einer PRINT-Anweisung ausgegeben werden. Nach der Initialisierung wird dieser Wert auf 0 gesetzt. Die meisten Drucker verfügen über einen RAM-Puffer, bei dem die Ausgabe der NULL-Zeichen nach einem Wagenrücklauf nicht erforderlich ist.

Syntax:

NULL[Ganzzahl]

PROG

Wichtig: Bevor Sie versuchen, den EEPROM-Speicher zu programmieren, lesen Sie bitte die Abschnitte PROG, PROG1 und PROG2 in diesem Kapitel.

Funktion:

Mit dem PROG-Befehl wird das im RAM-Speicher enthaltene aktuelle Programm in den residenten EEPROM-Speicher übertragen. UVPROM-Speicher können mit dem BASIC-Modul nicht programmiert werden.

Wichtig: Wählen Sie das zu speichernde Programm, bevor Sie den PROG-Befehl ausführen. Das RAM-Programm wird vom BASIC-Modul nicht automatisch in den EEPROM-Speicher kopiert. Wenn bei der EEPROM-Programmierung ein Fehler eintritt, wird die Meldung **`ERROR: Programming sequence failure`** angezeigt.

Nach der Eingabe von **`PROG`** zeigt das BASIC-Modul die Nummer des im EEPROM FILE enthaltenen Programms an. Die Programmierung dauert nur einige Sekunden.

Syntax:

PROG

Beispiel:

```
>LIST
1  REM EXAMPLE PROGRAM
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
40 END

READY
>PROG

ROM 12

Programming sequence successful.

READY
>ROM 12

>LIST
1  REM EXAMPLE PROGRAM
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
40 END

READY
>
```

Ergebnis: Das in diesem Beispiel gerade in den EEPROM-Speicher übertragene Programm ist das 12. gespeicherte Programm.

Wichtig: Wird der verfügbare EEPROM-Speicherplatz überschritten, kann die Programmierung erst nach dem Löschen der Daten fortgesetzt werden. Löschen Sie das zuletzt im EEPROM gespeicherte Programm mit dem ERASE-Befehl. Vor der EEPROM-Programmierung sollten Sie mit CALL 81 oder CALL 82 den verfügbaren Speicherplatz feststellen. Hinweise zu den Aufrufen 81 und 82 sind in Kapitel 5 enthalten.

PROG1

Wichtig: Bevor Sie versuchen, den EEPROM-Speicher zu programmieren, lesen Sie bitte die Abschnitte PROG, PROG1 und PROG2 in diesem Kapitel.

Funktion:

Mit dem PROG1-Befehl werden die Daten der drei Ports sowie MTOP-Daten im residenten EEPROM gespeichert. Beim Einschalten liest das BASIC-Modul diese Daten ab und initialisiert den MTOP-Opertor sowie alle drei seriellen Ports. Die Einschaltmeldung wird unmittelbar nach Abschluß der Rücksetzsequenz an das Terminal gesendet. Wenn sich die Baudrate des Terminals ändert, müssen Sie den EEPROM-Speicher neu programmieren, um die Kompatibilität zwischen Modul und Terminal sicherzustellen. Zur erneuten Programmierung müssen Sie die entsprechenden Port- bzw. MTOP-Daten ändern und anschließend den PROG1-Befehl erneut ausführen.

Syntax:

PROG1

Beispiel:

```
READY
>PROG1

Programming sequence successful.
```

PROG2

Wichtig: Bevor Sie versuchen, einen EEPROM-Speicher zu programmieren, lesen Sie bitte die Abschnitte PROG, PROG1 und PROG2 in diesem Kapitel. Beachten Sie, daß der Inhalt des RAM-Speichers nicht mit dem PROG2-Befehl an den EEPROM-Speicher übertragen wird. Mit dem PROG2-Befehl wird das erste Programm im EEPROM bei jeder Inbetriebnahme geladen.

Funktion:

Der PROG2-Befehl entspricht, mit Ausnahme der im folgenden aufgeführten Besonderheiten, dem PROG1-Befehl. Anstelle des Einschaltens und der Aktivierung des Befehlsmodus beginnt das Modul sofort mit der Ausführung des ersten im residenten EEPROM gespeicherten Programms. Anderenfalls führt es das im RAM gespeicherte Programm aus.

Mit dem PROG2-Befehl kann ein Programm nach dem Einschalten des Moduls ausgeführt werden (RUN), ohne daß eine Verbindung zum Terminal vorhanden ist. Das Sichern der PROG2-Daten entspricht der Eingabe einer Befehlssequenz ROM 1, RUN. Mit dieser Funktion kann außerdem eine besondere Initialisierungssequenz in BASIC geschrieben und eine anwendungsspezifische Einschaltmeldung für bestimmte Anwendungen generiert werden. Das erste Programm im Speichermodul wird durch den PROG2-Befehl nicht geändert.

Wichtig: Wenn die Standardkommunikation für PRT1 über JW4 gewählt ist, wird mit dem PROG2-Befehl nicht veranlaßt, daß das BASIC-Modul nach der Inbetriebnahme ein Programm ausführt. Weitere Hinweise sind in Kapitel 3 des Design- und Integrationshandbuchs für das BASIC-Modul SLC 500 (Publikationsnummer 1746-6.1DE) enthalten.

Die nachfolgende Abbildung zeigt den Betrieb des BASIC-Moduls nach einem Einschaltzustand und unter Verwendung der Befehle PROG1 und PROG2 bzw. des batteriegestützten RAM-Speichers.

Syntax:

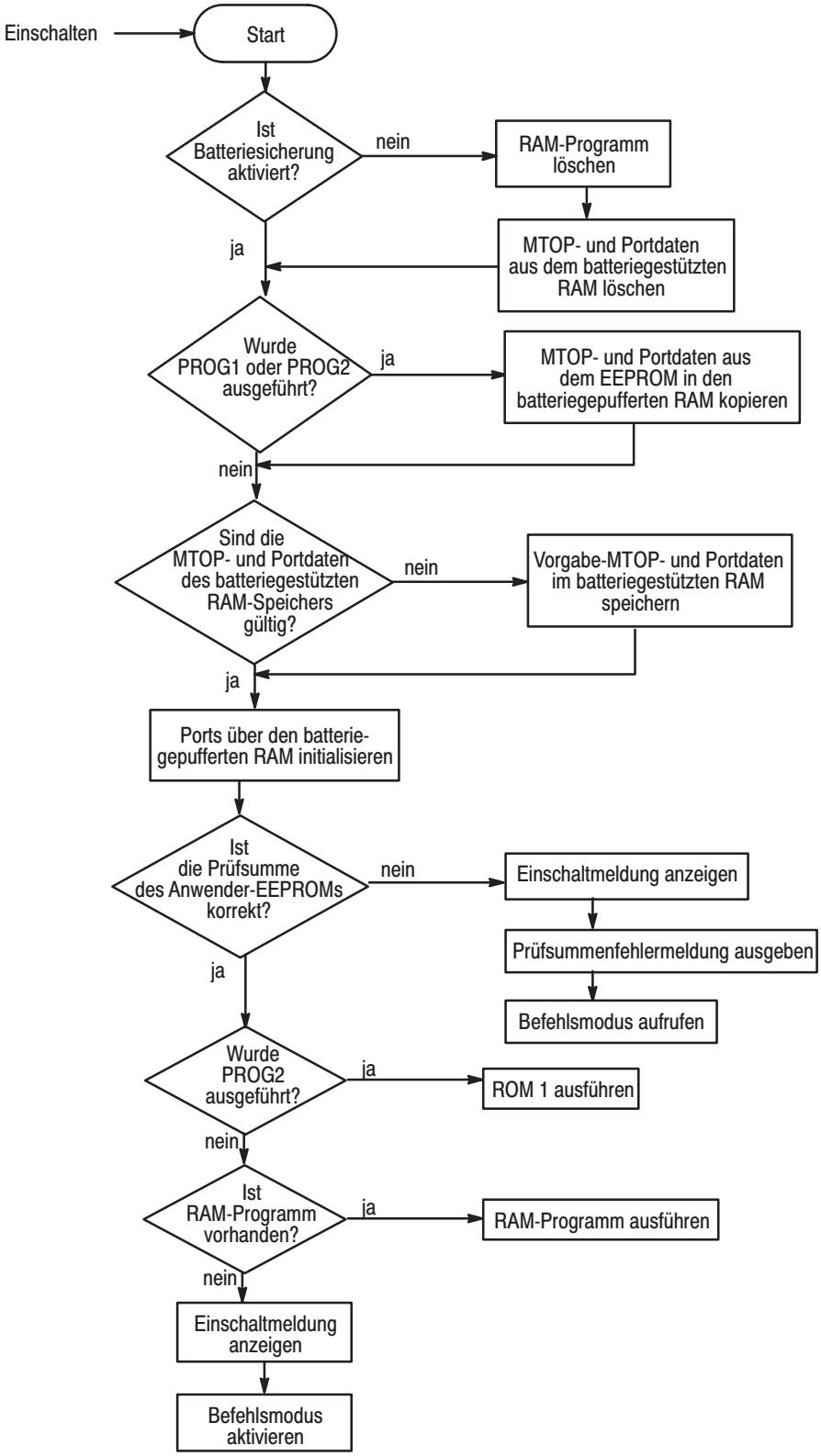
PROG2

Beispiel:

```
READY
>PROG2

Programming sequence successful.
```

Abbildung 4.1
Funktionsweise von PROG1 und PROG2



RAM

Funktion:

Mit dem RAM-Befehl wird der Interpreter des BASIC-Moduls angewiesen, das aktuelle Programm aus dem RAM-Speicher zu wählen. Das aktuelle Programm wird mit einem LIST-Befehl angezeigt und bei Eingabe von **RUN** ausgeführt.

Wichtig: Die RAM-Speicherkapazität ist auf 24 KB begrenzt. Berechnen Sie den verfügbaren Anwender-RAM anhand der folgenden Formel:

Verfügbarer Anwender-RAM = MTOP-H

H = LEN + S + 6*A + 8*V + 512

wobei gilt:

LEN = Systemsteuerwert, der die Länge des aktuellen RAM-Programms enthält

S = Anzahl der den Zeichenketten zugewiesenen Bytes (erster Wert im STRING-Befehl)

A = Summe aller (Datenfeldgrößen +1)

V = Summe aller verwendeten Variablennamen (einschließlich jedes Datenfeldnamens)

Syntax:

RAM

Beispiel:

```
READY
>RAM
```

REM

Funktion:

Mit dem REM-Befehl wird eine Kommentarzeile im BASIC-Programm spezifiziert. Durch das Einfügen von Kommentarzeilen wird ein Programm verständlicher. Programmzeilen, die mit einem REM-Befehl beginnen, dürfen nicht mit einem Doppelpunkt (:) beendet werden. REM-Befehle können in einer Programmzeile hinter einem Doppelpunkt (:) stehen. Somit kann in jede Zeile ein Kommentar eingefügt werden.

Wichtig: REM-Befehle erhöhen die Programmausführungszeit. Sie sollten deshalb gezielt angewandt oder am Programmende positioniert werden, wo sie die Programmausführungszeit nicht beeinflussen. Verwenden Sie REM-Befehle nicht in häufig aufgerufenen Schleifen und Unterprogrammen.

Syntax:

REM

Beispiel:

```
>10 REM DIES IST EINE KOMMENTARZEILE
>20 NEW : REM DIES IST EBENFALLS EINE KOMMENTARZEILE
```

REN

Funktion:

Mit dem REN-Befehl werden Programmzeilen neu numeriert.

Syntax:

REN[neue Nummer],[alte Nummer],[Inkrement]

Beispiele:

```
REN
```

Ergebnis: Das gesamte Programm wird neu numeriert. Die erste neue Zeilennummer ist 10. Die Zeilennummern werden jeweils um 10 inkrementiert.

```
REN 20
```

Ergebnis: Das gesamte Programm wird neu numeriert. Die erste neue Zeilennummer ist 10. Die Zeilennummern werden jeweils um 20 inkrementiert.

```
REN 300,50
```

Ergebnis: Das gesamte Programm wird neu numeriert. Die erste neue Zeilennummer ist 300. Die Zeilennummern werden jeweils um 50 inkrementiert.

```
REN 1000,900,20
```

Ergebnis: Das Programm wird ab Zeile 900 neu numeriert. Zeilennummer 900 wird nun Zeilennummer 1000. Alle folgenden Zeilennummern werden jeweils um 20 inkrementiert.

ROM

Funktion:

Mit dem ROM-Befehl wird der Interpreter des BASIC-Moduls angewiesen, das aktuelle Programm aus dem EEPROM- bzw. UVPROM-Speicher zu wählen. Das aktuelle Programm wird mit dem LIST-Befehl angezeigt und bei Eingabe von **RUN** ausgeführt.

Wichtig: Das BASIC-Modul kann je nach Größe der Programme und der EEPROM-Speicherkapazität bis zu 255 Programme im EEPROM speichern. Die Programme werden zum Abruf und zur Ausführung im EEPROM in einer Sequenzzeichenkette gespeichert, die als EEPROM-File bezeichnet wird.

Wenn Sie ROM [Ganzzahl] eingeben, wählt das BASIC-Modul das betreffende Programm aus dem EEPROM und lädt dieses als aktuelles Programm. Wenn nach dem ROM-Befehl keine Ganzzahl eingegeben wird (z.B. **ROM**), greift das Modul standardmäßig auf ROM 1 zu. Da die Programme im EEPROM der Reihe nach gespeichert sind, können Sie mit der Ganzzahl hinter dem ROM-Befehl das Programm wählen, das ausgeführt bzw. angezeigt werden soll. Wenn Sie versuchen, ein nicht vorhandenes Programm zu wählen (z.B., wenn Sie **ROM 8** eingeben, obwohl nur sechs Programme im EEPROM gespeichert sind), wird die Fehlermeldung **ERROR: PROM MODE** angezeigt.

Im ROM-Modus überträgt das Modul kein Programm vom EEPROM in den RAM-Speicher. Wenn Sie versuchen, im ROM-Modus ein Programm durch Eingabe einer Zeilennummer zu ändern, erscheint die Fehlermeldung **ERROR: PROM MODE**. Ein Programm kann zum Editieren mit dem XFER-Befehl vom EEPROM in den RAM-Speicher übertragen werden. Wenn Sie versuchen, eine Zeile eines RAM-Programms zu editieren, wird keine Fehlermeldung angezeigt.

Wichtig: Bei der Übertragung von Programmen aus dem EEPROM in den RAM-Speicher gehen die zuvor im RAM enthaltenen Daten verloren.

Da ein Programm *nicht* mit dem ROM-Befehl in den RAM-Speicher übertragen wird, können im ROM- und im RAM-Speicher gleichzeitig verschiedene Programme enthalten sein. Vom Befehlsmodus aus können Sie zwischen beiden Modi hin- und herschalten. Im Run-Modus erfolgt die Umschaltung mit den Aufrufen 70, 71 und 72. Wurde das Programm im EEPROM gespeichert, kann der gesamte RAM-Speicher als Variablenspeicher verwendet werden. Der Systemsteuerwert MTOP bezieht sich immer auf den RAM-Speicher. Der Systemsteuerwert LEN bezieht sich auf das derzeit gewählte Programm im RAM- oder ROM-Speicher.

Syntax:

ROM[Ganzzahl]

Beispiel:

```
READY
>ROM1
```

RROM

Funktion:

Mit dem RROM-Befehl wird der Interpreter des BASIC-Moduls angewiesen, das aktuelle Programm aus dem EEPROM bzw. UVPROM zu wählen und es anschließend auszuführen. Dieser Befehl entspricht der Eingabesequenz **ROM** – **RUN**.

Wichtig: Das BASIC-Modul kann je nach Größe der Programme und der EEPROM-Speicherkapazität bis zu 255 Programme im EEPROM speichern. Die Programme werden zum Abruf und zur Ausführung im EEPROM in einer Sequenzzeichenkette gespeichert, die als EEPROM-File bezeichnet wird.

Wenn Sie PROM [Ganzzahl] eingeben, wählt das BASIC-Modul das betreffende Programm aus dem EEPROM, lädt dieses als aktuelles Programm und beginnt mit der Ausführung. Wenn nach dem PROM-Befehl keine Ganzzahl eingegeben wird (z.B. **PROM**), greift das Modul standardmäßig auf PROM 1 zu. Da die Programme im EEPROM der Reihe nach gespeichert sind, können Sie mit der Ganzzahl hinter dem PROM-Befehl das Programm wählen, das ausgeführt bzw. angezeigt werden soll. Wenn Sie versuchen, ein nicht vorhandenes Programm zu wählen (z.B., wenn Sie **PROM 8** eingeben, obwohl nur sechs Programme im EEPROM gespeichert sind), wird die Fehlermeldung **ERROR: PROM MODE** angezeigt.

Im PROM-Modus überträgt das Modul kein Programm vom EEPROM in den RAM-Speicher. Wenn Sie versuchen, im ROM-Modus ein Programm durch Eingabe einer Zeilennummer zu ändern, erscheint die Fehlermeldung **ERROR: PROM MODE**. Ein Programm kann zum Editieren mit dem XFER-Befehl vom EEPROM in den RAM-Speicher übertragen werden. Wenn Sie versuchen, eine Zeile eines RAM-Programms zu editieren, wird keine Fehlermeldung angezeigt.

Wichtig: Bei der Übertragung von Programmen aus dem EEPROM in den RAM-Speicher gehen die zuvor im RAM enthaltenen Daten verloren.

Da ein Programm mit dem PROM-Befehl *nicht* in den RAM-Speicher übertragen wird, können im ROM- und im RAM-Speicher gleichzeitig verschiedene Programme enthalten sein. Vom Befehlsmodus aus können Sie zwischen beiden Modi hin- und herschalten. Im Run-Modus erfolgt die Umschaltung mit den Aufrufen 70, 71 und 72. Wurde das Programm im EEPROM gespeichert, kann der gesamte RAM-Speicher als Variablenspeicher verwendet werden. Der Systemsteuerwert MTOP bezieht sich immer auf den RAM-Speicher. Der Systemsteuerwert LEN bezieht sich auf das derzeit gewählte Programm im RAM- oder ROM-Speicher.

Syntax:

RROM[Ganzzahl]

Beispiel:

```
READY
>RROM
```

RUN

Funktion:

Mit dem RUN-Befehl werden alle Variablen auf Null gesetzt, alle durch BASIC veranlaßten Interrupts gelöscht, und die Programmausführung wird an der ersten Zeilennummer des gewählten Programms gestartet. Der Interpreter des BASIC-Moduls kann nur mit dem RUN-Befehl und der GOTO-Anweisung in den Run-Modus geschaltet werden. Die Programmabarbeitung kann jederzeit durch Eingabe von **[CTRL–C]** auf dem Terminal beendet werden.

Abweichungen: Bei einigen BASIC-Interpretern darf eine Zeilennummer hinter dem RUN-Befehl eingegeben werden (z.B. **RUN 100**). Im BASIC-Modul ist dies nicht erlaubt.

Die Ausführung beginnt mit der ersten Zeilennummer. Um eine Funktion ähnlich der des RUN-Befehls [Zeil-Nr] zu erhalten, verwenden Sie die GOTO[Zeil-Nr]-Anweisung im Direktmodus. Näheres ist in Kapitel 7 unter GOTO enthalten.

Syntax:

RUN

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 PRINT I
>30 NEXT I
>40 END
>RUN

  1
  2
  3

READY
>
```

SNGLSTP

Funktion:

Mit dem SNGLSTP-Befehl wird die Einzelschritt-Programmausführung eingeleitet. Wenn die mit diesem Befehl spezifizierte Programmnummer Null ist, wird die Einzelschritt-Programmausführung deaktiviert. Anderenfalls wird vor jeder Zeile des Programms ein Fixpunkt gesetzt. Starten Sie das Programm, indem Sie den RUN-Befehl eingeben. Geben Sie nach jedem Stopp **CONT** ein, um die nächste Zeile abzuarbeiten. An jedem Fixpunkt können Sie Variablen überprüfen oder zuordnen. SNGLSTP ist nur in Programmen funktionsfähig, die vom RAM-Speicher aus ausgeführt werden. Die Abarbeitung eines im EEPROM gespeicherten Programms wird mit diesem Befehl nicht gestoppt.

Syntax:

SNGLSTP[Ganzzahl]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 5
>20 PRINT I
>30 NEXT I
>40 PRINT “PASSED FOR - NEXT LOOP”
>50 PRINT “THIS IS THE END”
>60 END

READY
>SNGLSTP 20

SINGLE STEP ENABLED

READY
>RUN

STOP - LINE 20
READY
>CONT

1

STOP - LINE 30
READY
>CONT

STOP - LINE 20
READY
>CONT

2

STOP - LINE 30
READY
```

```
>CONT


STOP - LINE 20
READY
>CONT

3

STOP - LINE 30
READY
>SNGLSTP 0

SINGLE STEP DISABLED

READY
>CONT

4
5
PASSED FOR - NEXT LOOP
THIS IS THE END

READY
>
```

VER

Funktion:

Mit dem VER-Befehl wird die Einschaltmeldung des BASIC-Moduls ausgegeben, welche die aktuelle Version der Firmware anzeigt.

Syntax:

VER

Beispiel:

```
>VER
SLC 500 BASIC Module-Catalog Number 1746-BAS
Firmware release: 1.00
Allen-Bradley Company, Copyright 1991
All rights reserved

>
```

XFER

Funktion:

Mit dem XFER-Befehl wird das gewählte Programm vom ROM- in den RAM-Speicher übertragen und der RAM-Modus gewählt. Nach der Ausführung des XFER-Befehls können Sie das Programm wie jedes andere RAM-Programm editieren.

Wichtig: Mit dem XFER-Befehl werden bestehende RAM-Programme gelöscht.

Syntax:

XFER

Beispiel:

```
READY
>XFER
```

Kapitel 5

Aufrufe der Befehlszeile

In diesem Kapitel werden die Aufrufe beschrieben, die jeweils eine Funktion im BASIC-Modul veranlassen. Diese Aufrufe können nicht im BASIC-Programm ausgeführt werden, *sondern* werden auf der Befehlszeile eingegeben. Eine Auflistung der Aufrufe und der jeweiligen Mnemonik ist in Tabelle 5.A aufgeführt.

Tabelle 5.A
Kapitelinhalt

CALL 73 – Deaktivierung der RAM-Speicher-Batteriepufferung

Funktion:

Mit CALL 73 wird die Batteriepufferung des RAM-Speichers deaktiviert. Nach der Ausführung dieses Aufrufs wird auf dem Host-Terminal die Meldung **`Battery Backup Disabled`** angezeigt. Die Deaktivierung der RAM-Batteriepufferung ermöglicht eine löschende Rücksetzung. Beim Ausschalten der Versorgungsspannung des BASIC-Moduls wird der RAM-Speicher gelöscht und die Batteriepufferung erneut aktiviert.

Syntax:

CALL 73

Beispiel:

```
>CALL 73

Battery Backup Disabled

>REM TURNING POWER OFF, THEN BACK ON
```

CALL 74 - Aktivierung der RAM-Speicher-Batteriepufferung

Funktion:

Mit CALL 74 wird die Batteriepufferung des RAM-Speichers aktiviert. Nach der Ausführung dieses Aufrufs wird auf dem Host-Terminal die Meldung **`Battery Backup Enabled`** angezeigt. Die Batteriepufferung des RAM-Speichers wird beim Einschalten des BASIC-Moduls aktiviert und wird erst bei einer Ausführung des Aufrufs 73 oder bei erschöpfter Batterie deaktiviert.

Syntax:

CALL 74

Beispiel:

```
>CALL 74

Battery Backup Enabled
```

CALL 77 – Geschützter Variablenspeicher

Wichtig: Ändern Sie CALL 77 nur im Befehlsmodus, um eine störungsfreie Betriebsweise zu gewährleisten.

Funktion:

Mit CALL 77 wird der obere Bereich des RAM-Speichers als geschützter Variablenspeicher reserviert. Die Werte werden gesichert, wenn die Batteriepufferung eingeschaltet ist. Die Sicherung der Werte erfolgt mit dem ST@-Befehl, und der Abruf der Werte wird mit dem LD@-Befehl ausgeführt. Jede gespeicherte Variable erfordert sechs Speicherbytes.

Sie müssen die Anzahl der gespeicherten Variablen sechsmal vom MTOP-Wert subtrahieren und damit den verfügbaren RAM-Speicher verkleinern. Dieser Wert wird mit dem PUSH-Befehl als neue MTOP-Adresse auf dem Stapel abgelegt. Alle zugehörigen Variablenzeiger werden neu konfiguriert. Führen Sie diesen Befehl nur im Befehlsmodus aus.

Wichtig: Stellen Sie sicher, daß die ST@-Adresse nicht über die MTOP-Adresse geschrieben wird, weil dadurch der Wert einer Variablen oder einer Zeichenkette geändert werden könnte. Die niedrigste zulässige Einstellung für MTOP ist 4096 (1000H).

Wichtig: Mit CALL 77 wird die Zuordnung des Zeichenkettenspeichers und -inhaltes rückgängig gemacht. Deshalb muß dieser Aufruf vor der Abarbeitung der STRING-Anweisung ausgeführt werden.

Syntax:

PUSH [neue MTOP-Adresse]
CALL 77

Beispiel: (Speichern von zwei Variablen)

```
>PRINT MTOP
24575
>PRINT MTOP–12
24563
>PUSH 24563:REM NEW MTOP ADDRESS
>CALL 77

>1   REM EXAMPLE PROGRAM
>10  K = 678*PI
>20  L = 520
>30  PUSH K
>40  ST@ 24575 : REM STORE K IN PROTECTED AREA
>50  PUSH L
>60  ST@ 24569 : REM STORE L IN PROTECTED AREA
>70  REM TO RETRIEVE PROTECTED VARIABLES
>80  LD@ 24575 : REM REMOVE K FROM PROTECTED AREA
>90  POP K
>100 LD@ 24569 : REM REMOVE L FROM PROTECTED AREA
>110 POP L
>120 REM USE LD@ AFTER POWER LOSS AND BATTERY BACK–UP IS
USED
```

CALL 81 – Überprüfung und Beschreibung des Anwenderspeichermoduls

Funktion:

Mit CALL 81 wird das Anwenderspeichermodul vor dem Einbrennen eines Programms in das Speichermodul überprüft. Diese Routine:

- legt die Anzahl der Speichermodulprogramme fest.
- legt die Anzahl der freien Bytes im Speichermodul fest.
- legt die Anzahl der Bytes des RAM-Programms fest.
- gibt eine Meldung aus, die signalisiert, ob das Speichermodul über die für das RAM-Programm erforderliche Speicherkapazität verfügt.
- überprüft die Prüfsumme des Speichermoduls, wenn ein Programm gefunden wird.
- gibt bei einem Prüfsummenfehler eine Warnmeldung aus.

Wichtig: Mit CALL 81 wird ein defektes Speichermodul nicht erkannt.

PUSH- und POP-Anweisungen werden nicht benötigt.

Syntax:

CALL 81

Beispiel:

```
>CALL 81

Number of BASIC programs in (E)EPROM.........  3
Available bytes to end of user (E)EPROM.......  7944
Available bytes to beginning of assembly pgm..  3848
Length of BASIC program in RAM...............  76

Program will fit in (E)EPROM.

READY
>
```

CALL 82 – Überprüfung der Belegung des Anwenderspeichermoduls

Funktion:

Mit CALL 82 wird das Anwenderspeichermodul überprüft und die Speicherbelegung der BASIC-Programme angezeigt. Mit Hilfe dieses Aufrufs kann der Programmierer feststellen, an welcher Stelle freier Speicherplatz verfügbar ist und wieviel Speicherkapazität im Modul vorhanden ist. PUSH- und POP-Anweisungen sind nicht erforderlich.

Syntax:

CALL 82

Beispiel:

```
>CALL 82

 8010H --  805CH --> ROM 1
 805DH --  80A9H --> ROM 2
 80AAH --  80F6H --> ROM 3
 80F7H --  FFFFH --> UNUSED
>
```

CALL 101 – Hochladen des Anwenderspeichercodes an den Host

Funktion:

Mit CALL 101 wird der Code des Anwenderspeichermoduls an das Host-Terminal hochgeladen. Dieser Aufruf erfordert zwei PUSH-Anweisungen, jedoch keine POP-Anweisungen. Bei der ersten PUSH-Anweisung handelt es sich um die Startadresse, bei der zweiten PUSH-Anweisung um die Endadresse. Mit diesem Aufruf werden die Daten innerhalb des Adreßbereichs in das Intel™ Hex-Format umgewandelt und anschließend über den Programmierport ausgegeben. Stimmen die Adressen nicht überein, erscheint eine Fehlermeldung.

Syntax:

PUSH [Startadresse]
PUSH [Endadresse]
CALL 101

Beispiel:

```
>PUSH 8000 : PUSH 804FH : CALL 101

:108000003107021327CC3313276607005FFF473081
:108010005509000A8B41E034290D1000149C3130C1
:108020002C32302C33302C34300D0A001EA049EA9B
:1080300030A6330D0900289B41E049290D06003286
:1080400097490D0A003CA04AEA4FA6330D090046A5
:00000001F
>
```

CALL 103 – Ausdruck des PRT1-Ausgangspuffers und -Zeigers

Funktion:

Mit CALL 103 wird der vollständige Inhalt des Ausgangspuffers, einschließlich Adresse, vorderem Zeiger und Anzahl der Zeichen im Puffer auf einem an den Programmierport angeschlossenen Datensichtgerät ausgegeben. PUSH- und POP-Anweisungen werden nicht benötigt.

Verwenden Sie diese Daten bei der Fehlersuche. Der Inhalt der Puffer wird hiervon nicht beeinflußt.

Syntax:

CALL 103

Beispiel:

```
>CALL 103
                                        PRT1 Output Queue

6D00H 3AH 31H 30H 38H 30H 34H 30H 30H 30H 39H 37H 34H 39H 30H 44H 30H
6D10H 41H 30H 30H 33H 43H 41H 30H 34H 41H 45H 41H 34H 46H 41H 36H 33H
6D20H 33H 30H 33H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H 38H
6D30H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 33H 48H 20H 34H 38H
6D40H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H 38H
6D50H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH 20H
6D60H 36H 44H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 34H 38H
6D70H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH 20H
6D80H 36H 44H 37H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 32H
6D90H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H
6DA0H 48H 20H 33H 34H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H
6DB0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 34H
6DC0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 32H
6DD0H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H
6DE0H 48H 20H 33H 34H 48H 0DH 0AH 20H 36H 44H 43H 30H 48H 20H 34H 38H
6DF0H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 38H 48H 20H 34H 34H

  Output queue front pointer is:  6D29H
```

CALL 104 – Ausdruck des PRT1-Eingangspuffers und -Zeigers

Funktion:

Mit CALL 104 wird der vollständige Inhalt des Eingangspuffers, einschließlich Adresse, vorderem Zeiger und Anzahl der Zeichen im Puffer auf einem an den Programmierport angeschlossenen Datensichtgerät ausgegeben. PUSH- und POP-Anweisungen werden nicht benötigt.

Verwenden Sie diese Daten bei der Fehlersuche. Der Inhalt der Puffer wird hiervon nicht beeinflußt.

Syntax:

CALL 104

Beispiel:

```
>CALL 104
                                        PRT1 Input Queue

6C00H 33H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 7FH 7FH 7FH 7FH 7FH 7FH
6C10H 7FH 7FH 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 7FH 7FH
6C20H 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 0DH 0DH 0DH 0DH 0DH 0DH
6C30H 0DH 0DH 0DH 45H 58H 41H 4DH 7FH 7FH 7FH 7FH 52H 45H 4DH 20H 45H
6C40H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH 20H 50H 41H 47H 45H 20H
6C50H 36H 2DH 37H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 0DH 52H 4DH 41H
6C60H 4EH 54H 20H 7FH 7FH 7FH 54H 20H 44H 41H 54H 41H 0DH 52H 45H 4DH
6C70H 20H 54H 48H 45H 52H 45H 20H 49H 53H 20H 4EH 4FH 20H 52H 45H 41H
6C80H 4CH 20H 52H 45H 53H 50H 4FH 4EH 53H 45H 20H 57H 48H 49H 43H 48H
6C90H 20H 57H 49H 4CH 4CH 20H 53H 48H 4FH 57H 20H 55H 50H 20H 49H 4EH
6CA0H 20H 41H 4EH 20H 45H 58H 41H 4DH 50H 4CH 45H 0DH 0DH 0DH 0DH 0DH
6CB0H 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH 20H
6CC0H 50H 41H 47H 45H 20H 36H 2DH 36H 0DH 50H 55H 53H 48H 20H 38H 30H
6CD0H 30H 30H 48H 3AH 50H 7FH 7FH 20H 70H 55H 53H 7FH 7FH 7FH 3AH 20H
6CE0H 50H 55H 53H 48H 20H 38H 30H 7FH 30H 34H 46H 48H 20H 3AH 43H 41H
6CF0H 4CH 4CH 20H 31H 30H 31H 0DH 0DH 0DH 43H 41H 4CH 4CH 20H 31H 30H

Input queue front pointer is:  6C5DH
```

CALL 109 – Ausdruck des Argumentstapels

Funktion:

Mit CALL 109 werden neun Werte des oberen Argumentstapelbereichs am Terminal ausgegeben. PUSH- und POP-Anweisungen sind nicht erforderlich. Verwenden Sie diese Daten bei der Fehlersuche. Der Inhalt des Argumentstapels bzw. des Stapelzeigers wird hiervon nicht beeinflußt.

Syntax:

CALL 109

Beispiel:

```
>CALL 109

 1C9H 00H 00H 00H 00H 00H 00H
 1CFH 00H 00H 00H 00H 00H 00H
 1D5H 00H 00H 00H 00H 00H 00H
 1DBH 41H 67H 50H 00H 00H 7EH
 1E1H 83H 75H 00H 00H 00H 7CH
 1E7H 13H 04H 00H 00H 00H 7CH
 1EDH 32H 84H 70H 00H 00H 85H
 1F3H 32H 76H 80H 00H 00H 85H
 1F9H 00H 00H 00H 00H 00H 00H

Argument stack pointer is:  01FEH
```

CALL 110 – Ausdruck des PRT2-Ausgangspuffers und -Zeigers

Funktion:

Mit CALL 110 wird der vollständige Inhalt des Ausgangspuffers, einschließlich Adresse, vorderem Zeiger und Anzahl der Zeichen im Puffer auf einem an den Programmierport angeschlossenen Datensichtgerät ausgegeben. PUSH- und POP-Anweisungen werden nicht benötigt.

Verwenden Sie diese Daten bei der Fehlersuche. Der Inhalt der Puffer wird hiervon nicht beeinflußt.

Syntax:

CALL 110

Beispiel:

```
>CALL 110
                                        PRT2 Output Queue

6F00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F10H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F20H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F30H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F40H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F50H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F60H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F70H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F80H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F90H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FA0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FB0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FC0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FD0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FE0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FF0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H

Output queue front pointer is:  6F00H
```

CALL 111 – Ausdruck des PRT2-Eingangspuffers und -Zeigers

Funktion:

Mit CALL 111 wird der vollständige Inhalt des Eingangspuffers, einschließlich Adresse, vorderem Zeiger und Anzahl der Zeichen im Puffer auf einem an den Programmierport angeschlossenen Datensichtgerät ausgegeben. PUSH- und POP-Anweisungen werden nicht benötigt.

Verwenden Sie diese Daten bei der Fehlersuche. Der Inhalt der Puffer wird hiervon nicht beeinflußt.

Syntax:

CALL 111

Beispiel:

```
>CALL 111
                                        PRT2 Input Queue

6E00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E10H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E20H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E30H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E40H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E50H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E60H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E70H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E80H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E90H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EA0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EB0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EC0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6ED0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EE0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EF0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H

Input queue front pointer is:  6E00H
```

Kapitel 6

Zuordnungsfunktionen

In diesem Kapitel werden die Befehle beschrieben und veranschaulicht, die zur Speicherzuordnung, Rücksetzung des Datenspeichers und der Variablenwerte in einem BASIC-Programm oder von der Befehlszeile aus eingesetzt werden. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 6.A enthalten.

Tabelle 6.A
Kapitelinhalt

CLEAR

Funktion:

Mit der CLEAR-Anweisung werden alle Variablen auf 0 gesetzt und alle durch BASIC ausgelösten Interrupts und Stapel zurückgesetzt. Dies bedeutet, daß nach der Ausführung der CLEAR-Anweisung eine ONTIME-Anweisung ausgeführt werden muß, bevor das Modul die internen Zeitwerk-Interrupts bestätigt. Die Fehlereingrenzung mit Hilfe der ONERR-Anweisung erfolgt erst nach Ausführung einer ONERR[zeil nr]-Anweisung.

Die durch die CLOCK1-Anweisung aktivierte freilaufende Uhr wird von der CLEAR-Anweisung nicht beeinflußt. CLOCK0 ist die einzige Modulanweisung, mit der die freilaufende Uhr deaktiviert werden kann.

Auch wird der den Zeichenketten zugewiesene Speicher durch die CLEAR-Anweisung nicht zurückgesetzt. Deshalb brauchen Sie die Anweisung STRING [ausdr], [ausdr] nicht einzugeben, um den Zeichenketten nach der Ausführung der CLEAR-Anweisung erneut einen Speicherplatz zuzuweisen. Die CLEAR-Anweisung wird im allgemeinen zum Löschen aller Variablen benutzt.

Syntax:

CLEAR

Beispiel:

```
>CLEAR

>LIST
1  REM EXAMPLE PROGRAM
10 DIM A(4)
20 DATA 10,20,30,40
30 FOR I=0 TO 3
40 READ A(I)
50 NEXT I
60 FOR J=O TO 3
70 PRINT A(J)
80 NEXT J

READY
>PRINT A(1),I,J
 0  0  0

>RUN

 10
 20
 30
 40

READY
>PRINT A(1),I,J
 20  4  4

>CLEAR

>PRINT A(1),I,J
 0  0  0
```

CLEARI

Funktion:

Mit der CLEARI-Anweisung werden alle durch BASIC ausgelösten Interrupts gelöscht. Der ONTIME-Interrupt wird nach der Ausführung der CLEARI-Anweisung deaktiviert.

Die durch die CLOCK1-Anweisung aktivierte freilaufende Uhr wird von der CLEARI-Anweisung nicht beeinflußt. CLOCK0 ist die einzige Modulanweisung, mit der die freilaufende Uhr deaktiviert werden kann.

Mit dieser Anweisung können gewählte ONTIME-Interrupts in bestimmten Abschnitten des BASIC-Programms deaktiviert werden. Sie müssen die ONTIME-Anweisung erneut ausführen, um die jeweiligen Interrupts zu aktivieren.

Wichtig: Bei der Ausführung des LIST-Befehls erscheint die CLEARI-Anweisung als CLEARI.

Syntax:

CLEARI

Beispiel:

```
READY
>CLEARI
```

CLEARS

Funktion:

Mit der CLEARS-Anweisung werden alle Stapel zurückgesetzt. Die Steuer- und Argumentstapel sowie die internen Stapel werden auf ihre Initialisierungswerte zurückgesetzt. Wenn ein Fehler in einem Unterprogramm eintritt, können die Stapel mit diesem Befehl zurückgesetzt werden.

Wichtig: Bei der Ausführung des LIST-Befehls erscheint die CLEARS-Anweisung als CLEARS.

Syntax:

CLEARS

Beispiel:

```
READY
>CLEARS
```

DATA

Funktion:

Mit der DATA-Anweisung werden die mit einer READ-Anweisung abgerufenen Ausdrücke spezifiziert. Wenn eine Zeile mehrere Ausdrücke enthält, *müssen* diese durch ein Komma getrennt werden.

Durch jede READ-Anweisung wird veranlaßt, daß der nächste Ausdruck der DATA-Anweisung bewertet und der Variablen in der READ-Anweisung zugeordnet wird. Sie können DATA-Anweisungen an einer beliebigen Stelle im Programm anordnen. Sie werden nicht ausgeführt und verursachen keine Fehler. Das System setzt voraus, daß DATA-Anweisungen verkettet sind. Sie erscheinen deshalb als eine einzige lange DATA-Anweisung. Wenn alle Daten gelesen wurden und eine neue READ-Anweisung ausgeführt wird, stoppt das Programm, und auf dem Terminalbildschirm wird die Fehlermeldung **`ERROR: NO DATA – IN LINE XX`** angezeigt. Das Modul wird in den Befehlsmodus zurückgeschaltet.

Syntax:

DATA

Beispiel:

```
>LIST
1  REM EXAMPLE PROGRAM
10 DIM A(4)
20 DATA 10,ASC(A),ASC(C),35.627
30 FOR I=0 TO 3
40 READ A(I)
50 NEXT I
60 FOR J=O TO 3
70 PRINT A(J)
80 NEXT J


READY
>RUN

 10
 65
 67
 35.627
```

Wichtig: In einer DATA-Anweisung darf der CHR-Operator nicht verwendet werden.

DIM

Funktion:

Mit der DIM-Anweisung wird ein Speicherbereich für Matrizen reserviert. Zu Beginn setzt das System voraus, daß dieser Bereich gleich Null ist. Matrizen im BASIC-Modul müssen eindimensional sein, und die Größe des dimensionierten Datenfelds darf 254 Elemente nicht überschreiten.

Eine in einem Programm bereits dimensionsierte Variable darf nicht erneut dimensioniert werden. Anderenfalls wird ein Datenfeldgrößenfehler verursacht, und das Modul schaltet in den Befehlsmodus um.

Wenn Sie eine Datenfeldvariable verwenden, die nicht mit einer DIM-Anweisung dimensioniert wurde, wird der Datengröße durch das BASIC-Programm der Vorgabewert 10 zugewiesen. Bei der Ausführung eines RUN- oder NEW-Befehls sowie einer CLEAR-Anweisung werden alle Datenfelder auf Null gesetzt.

Die Anzahl der einem Datenfeld zugewiesenen Bytes entspricht dem Sechsfachen der Datenfeldgröße plus 1 (6 * (Datenfeldgröße + 1)). Beispiel: Datenfeld A (100) erfordert 606 Speicherbytes. Die Größe eines dimensionierten Datenfeldes ist im allgemeinen durch die Speichergröße begrenzt.

Syntax:

DIM

Beispiele:

Mit einer DIM-Anweisung können mehrere Variablen dimensioniert werden.

```
>1  REM EXAMPLE PROGRAM
>10 DIM A(25), C(15), A1(20)
```

Fehler bei dem Versuch, ein Datenfeld erneut zu dimensionieren:

```
>1  REM EXAMPLE PROGRAM
>10 A(5) = 10 : REM BASIC ASSIGNS DEFAULT OF 10 TO ARRAY A
>20 DIM A(5) : REM ARRAY RE-DIMENSION ERROR
>

READY
>RUN

ERROR: ARRAY SIZE - IN LINE  20

20     DIM A(5) :  REM ARRAY RE-DIMENSION ERROR
---------------X
READY
>
```

LET

Funktion:

Mit der LET-Anweisung wird dem Wert eines Ausdrucks eine Variable zugeordnet.

Syntax:

LET [var] = [ausdr]

Beispiele:

```
>1  REM EXAMPLE PROGRAM
>10 LET A = 10*SIN(C)/100
```

```
>1  REM EXAMPLE PROGRAM
>10 LET A = A +1
```

Beachten Sie bitte, daß das in der LET-Anweisung verwendete Gleichheitszeichen kein Äquivalenzoperator, sondern ein Ersatzoperator ist. Die Anweisung sollte folgendermaßen gelesen werden: A wird durch A plus 1 ersetzt. Das Wort LET ist immer optional (Beispiel: **`LET A = 2`** entspricht **`A = 2`**).

Wenn das Wort LET weggelassen wird, spricht man von einer sogenannten IMPLIZIERTEN LET-Anweisung. Das Wort LET kennzeichnet sowohl eine LET-Anweisung als auch eine IMPLIZIERTE LET-Anweisung.

Mit der LET-Anweisung können auch Zeichenkettenvariablen zugeordnet werden:

```
LET $(1)="DIES IST EINE ZEICHENKETTE"
```

oder

```
LET $(2)=$(1)
```

Vor der Zuordnung von Zeichenketten müssen Sie die Anweisung STRING [ausdr], [ausdr] ausführen. Anderenfalls wird ein Speicherzuordnungsfehler verursacht, und das Modul schaltet in den Befehlsmodus zurück.

RESTORE

Funktion:

Mit der RESTORE-Anweisung wird der interne Lesezeiger an den Anfang der Daten zurückgesetzt, so daß diese erneut gelesen werden können.

Syntax:

RESTORE

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 READ A,C
>30 PRINT A,C
>40 NEXT I
>50 RESTORE
>60 READ A,C
>70 PRINT A,C
>80 DATA 10,20,10/2,20/2,SIN(PI),COS(PI)

READY
>RUN

 10        20
 5         10
 0         -1
 10        20
```

Kapitel 7

Steuerfunktionen

In diesem Kapitel werden die Befehle beschrieben, die im BASIC-Programm oder von der Befehlszeile aus ausgeführt werden, um die interne Uhr bzw. den Ablauf des BASIC-Programms zu steuern. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 7.A enthalten.

Tabelle 7.A
Kapitelinhalt

CLOCK1

Funktion:

Mit der CLOCK1-Anweisung wird die freilaufende Uhr im BASIC-Modul aktiviert. Der Sonderfunktionsoperator TIME wird nach der Ausführung der CLOCK1-Anweisung alle fünf Millisekunden einmal erhöht. Die CLOCK1-Anweisung erzeugt über ein internes ZEITWERK alle fünf Millisekunden einen Interrupt. Aus diesem Grund verfügt der Sonderfunktionsoperator TIME über eine Auflösung von fünf Millisekunden. Er führt Zählungen zwischen 0 und 65535,995 Sekunden aus. Wenn der Zählwert 65535,995 Sekunden erreicht, findet ein Überlauf des TIME-Operators statt, und dieser startet erneut bei Null. Aufgrund der mit der CLOCK1-Anweisung zusammenhängenden Interrupts werden Modulprogramme mit etwa 99,6% der normalen Geschwindigkeit abgearbeitet. Dies bedeutet, daß die Bearbeitung der Interrupts für die freilaufende Uhr etwa 0,4% der gesamten CPU-Zeit erfordert.

Wichtig: Hierbei wurde der zusätzliche Aufwand für die Ausführung des ON-TIME-Anwenderinterrupts noch nicht berücksichtigt.

Syntax:

CLOCK1

Beispiel:

```
>NEW

>1   REM EXAMPLE PROGRAM
>10  TIME = 0
>15  DBY(71) = 0
>20  CLOCK1
>30  ONTIME 2,100
>40  DO
>50  WHILE TIME < 10
>60  END
>100 PRINT “TIMER INTERRUPT AT – ”,TIME,“ SECONDS”
>110 ONTIME TIME+2,100
>120 RETI

READY
>RUN
TIMER INTERRUPT AT –  2.01  SECONDS
TIMER INTERRUPT AT –  4.015  SECONDS
TIMER INTERRUPT AT –  6.01  SECONDS
TIMER INTERRUPT AT –  8.01  SECONDS
TIMER INTERRUPT AT –  10.01  SECONDS
```

CLOCK0

Funktion:

Mit der Anweisung CLOCK0 (Null) wird die freilaufende Uhr im BASIC-Modul deaktiviert oder ausgeschaltet. Nach der Ausführung von CLOCK0 führt der Sonderfunktionsoperator TIME keine weiteren Inkrementierungen aus. CLOCK0 ist die einzige Modulanweisung, mit der die freilaufende Uhr deaktiviert werden kann. Mit CLEAR und CLEARI wird *nicht* die freilaufende Uhr, sondern nur ihr entsprechender ONTIME-Interrupt deaktiviert.

Wichtig: CLOCK1 und CLOCK0 stehen mit der Uhrzeit-/Kalenderfunktion in keinem Zusammenhang.

Syntax:

CLOCK0

Beispiel:

```
READY
>CLOCK0
```

DO-WHILE

Funktion:

Mit der DO-WHILE-Anweisung wird die Schleifensteuerung in einem Modulprogramm konfiguriert. Die Funktionsweise dieser Anweisung ähnelt der Anweisung DO-UNTIL [rel ausdr]. Alle Anweisungen zwischen DO und WHILE [rel ausdr] werden so lange ausgeführt, wie der der WHILE-Anweisung folgende relationale Ausdruck wahr ist. DO-WHILE-Anweisungen können verschachtelt werden.

Im Steuerstapel (C-Stapel) werden alle Daten gespeichert, die mit der Schleifensteuerung in Zusammenhang stehen (Beispiel: DO-WHILE, DO-UNTIL, FOR-NEXT und BASIC-Subroutinen). Der Steuerstapel umfaßt 157 Bytes. DO-WHILE- und DO-UNTIL-Schleifen sowie GOSUB-Befehle belegen drei Bytes und FOR-NEXT-Schleifen 17 Bytes des Steuerstapels.

Wichtig: Bei einer übermäßigen Verschachtelung werden die Grenzen des Steuerstapels überschritten, wodurch eine Fehlermeldung angezeigt und das Modul in den Befehlsmodus geschaltet wird.

Syntax:

DO-WHILE [rel ausdr]

Beispiele:

Einfache DO-WHILE-Schleife

```
>NEW

>1  REM EXAMPLE PROGRAM
>10 DO
>20 A = A + 1
>30 PRINT A
>40 WHILE A < 4
>50 PRINT “DONE”
>60 END

READY
>RUN

 1
 2
 3
 4
DONE

READY
>
```

Verschachtelte DO-WHILE-Schleife

```
>NEW

>1  REM EXAMPLE PROGRAM
>10 A=0 : C=0
>20 DO
>30 A=A+1
>40 DO
>45 C=C+1
>50 PRINT A,C,A*C
>60 WHILE C<>3
>70 C=0
>80 WHILE A<4
>90 END

READY
>RUN
 1  1  1
 1  2  2
 1  3  3
 2  1  2
 2  2  4
 2  3  6
 3  1  3
 3  2  6
 3  3  9

READY
>
```

DO-UNTIL

Funktion:

Mit der DO-UNTIL-Anweisung wird die Schleifensteuerung in einem Modulprogramm konfiguriert. Alle Anweisungen zwischen DO und UNTIL [rel ausdr] werden so lange ausgeführt, wie der der UNTIL-Anweisung folgende relationale Ausdruck wahr ist. DO-UNTIL-Anweisungen können verschachtelt werden.

Im Steuerstapel (C-Stapel) werden alle Daten gespeichert, die mit der Schleifensteuerung in Zusammenhang stehen (Beispiel: DO-WHILE, DO-UNTIL, FOR-NEXT und BASIC-Subroutinen). Der Steuerstapel umfaßt 157 Bytes. DO-WHILE- und DO-UNTIL-Schleifen sowie GOSUB-Befehle belegen drei Bytes und FOR-NEXT-Schleifen 17 Bytes des Steuerstapels.

Wichtig: Bei einer übermäßigen Verschachtelung werden die Grenzen des Steuerstapels überschritten, wodurch eine Fehlermeldung angezeigt und das Modul in den Befehlsmodus geschaltet wird.

Syntax:

DO-UNTIL [rel ausdr]

Beispiele:

Einfache DO-UNTIL-Schleife

```
>1  REM EXAMPLE PROGRAM
>10 A=0
>20 DO
>30 A=A+1
>40 PRINT A
>50 UNTIL A=4
>60 PRINT “DONE”
>70 END
>RUN
```

Verschachtelte DO-UNTIL-Schleife

```
>1  REM EXAMPLE PROGRAM
>10 DO
>20 A=A+1
>30 DO
>40 C=C+1
>50 PRINT A,C,A*C
>60 UNTIL C=3
>70 C=0
>80 UNTIL A=3
>90 END
RUN
```

END

Funktion:

Mit der END-Anweisung wird die Programmausführung beendet. Bei Verwendung dieser Anweisung ist CONT nicht funktionsfähig, und auf dem Terminalbildschirm wird die Fehlermeldung **`ERROR : CAN'T CONTINUE`** angezeigt. Zur ordnungsgemäßen Beendigung eines Programms sollten Sie immer eine END-Anweisung programmieren.

Syntax:

END

Beispiel:

Programmabschluß mit END-Anweisung

```
>1  REM PROGRAMMBEISPILE
>10 FOR I = 1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END

READY
>RUN

 1  2  3  4
READY
>
```

FOR-TO-(STEP)-NEXT

Funktion:

Mit der FOR-TO-(STEP)-NEXT-Anweisung werden Programmschleifen konfiguriert und gesteuert.

Im Steuerstapel (C-Stapel) werden alle Daten gespeichert, die mit der Schleifensteuerung in Zusammenhang stehen (Beispiel: DO-WHILE, DO-UNTIL, FOR-NEXT und BASIC-Subroutinen). Der Steuerstapel umfaßt 157 Bytes. DO-WHILE- und DO-UNTIL-Schleifen sowie GOSUB-Befehle belegen drei Bytes und FOR-NEXT-Schleifen 17 Bytes des Steuerstapels.

Wichtig: Bei einer übermäßigen Verschachtelung werden die Grenzen des Steuerstapels überschritten, wodurch eine Fehlermeldung angezeigt und das Modul in den Befehlsmodus geschaltet wird.

Syntax:

FOR [ausdr] TO [ausdr] STEP [ausdr]
.
.
.
NEXT [ausdr]

Beispiele:

```
>1  REM EXAMPLE PROGRAM
>5  E=0 : C=10 : D=2
>10 FOR A=E TO C STEP D
>20 PRINT A
>30 NEXT A
>40 END
>RUN
```

```
0
2
4
6
8
10

READY

>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END

READY
>RUN

 1  2  3  4
```

Da im ersten Beispiel E=0, C=10, D=2 und die PRINT-Anweisung in Zeile 20 sechsmal ausgeführt wird, werden für A folgende Werte ausgegeben: 0, 2, 4, 6, 8 und 10. “A” repräsentiert den Namen des Index- bzw. Schleifenzählers. Der Wert “E” entspricht dem beginnnenden Wert des Index. “C” ist der Grenzwert des Index und “D” das Inkrement für den Index.

Werden die STEP-Anweisung und Wert “D” nicht programmiert, wird als Inkrement standardmäßig der Wert 1 verwendet. STEP ist also eine optionale Anweisung. Die NEXT-Anweisung bewirkt den Rücksprung an den Schleifenbeginn und die Addition des Wertes “D” zum aktuellen Indexwert. Dieser wird anschließend mit dem Indexgrenzwert “C” verglichen.

Ist der Index kleiner als oder gleich dem Grenzwert, erfolgt nach der FOR-Anweisung ein Rücksprung zur Anweisung. Ein negativer Schrittwert (FOR I = 100 TO 1 STEP–1) ist im BASIC-Modul erlaubt. Die NEXT-Anweisung steht immer vor der entsprechenden Variablen. FOR-NEXT-Schleifen dürfen höchstens neunmal verschachtelt werden.

```
>1  REM EXAMPLE PROGRAM
>10 FOR I=1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END
>RUN
>1 2 3 4


READY
```

```
>1  REM EXAMPLE PROGRAM
>10 FOR I=0 TO 8 STEP 2
>20 PRINT I
>30 NEXT I
>40 END
>RUN
0
2
4
6
8

READY
```

GOTO

Funktion:

Mit der GOTO-Anweisung wird die Steuerung an die spezifizierte Zeilennnummer ([zeil nr]) übertragen.

Syntax:

GOTO [zeil nr]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>50 GOTO 100
```

Wenn Zeile 100 vorhanden ist, wird die Programmabarbeitung durch die Anweisung in dieser Zeile fortgesetzt. Wenn Zeile 100 nicht vorhanden ist, erscheint auf dem Terminalbildschirm die Fehlermeldung **`ERROR:  INVALID LINE NUMBER`**, und das BASIC-Modul wird in den Befehlsmodus geschaltet.

Im Gegensatz zum RUN-Befehl werden der Variablenspeicher und die Interrupts mit der GOTO-Anweisung vom Befehlsmodus aus nicht gelöscht. Wird die GOTO-Anweisung jedoch im Befehlsmodus nach der Änderung einer Zeile ausgeführt, löscht das Modul den Variablenspeicher und alle durch BASIC ausgelösten Interrupts.

IF-THEN-ELSE

Funktion:

Mit der IF-THEN-ELSE-Anweisung wird eine bedingte Prüfung konfiguriert.

Syntax:

IF [rel ausdr] THEN gültige Anweisung ELSE gültige Anweisung

Beispiele:

```
>1  REM EXAMPLE PROGRAM
>10 IF A =100 THEN A=0 ELSE A=A+1
```

Wenn nach der Ausführung von Zeile 10 “A” gleich 100 ist, dann wird der Variablen A der Wert 0 zugeordnet. Wenn A ungleich 100 ist, wird der Variablen A der Wert A+1 zugeordnet. Wenn die Steuerung mit der IF-Anweisung an verschiedene Zeilennummern übertragen werden soll, ist die GOTO-Anweisung nicht erforderlich. In den folgenden Beispielen wird jeweils das gleiche Ergebnis erzielt.

```
>20 IF INT(A)<10 THEN GOTO 100 ELSE GOTO 200
          oder
>20 IF INT(A)<10 THEN 100 ELSE 200
```

Die THEN-Anweisung kann durch eine beliebige gültige BASIC-Anweisung ersetzt werden:

```
>30 IF A<>10 THEN PRINT A ELSE 10
>30 IF A<>10 PRINT A ELSE 10
```

Nach THEN und ELSE können mehrere Anweisungen ausgeführt werden, die jedoch durch einen Doppelpunkt voneinander getrennt werden müssen.

Beispiel:

```
>30 IF A<>10 THEN PRINT A : GOTO 150 ELSE 10
>30 IF A<>10 PRINT A : GOTO 150 ELSE 10
```

In diesen Beispielen werden sowohl PRINT A als auch GOTO 150 ausgeführt, wenn A ungleich 10 ist. Wenn A gleich 10 ist, wird die Steuerung an Zeile 10 übertragen.

Die ELSE-Anweisung ist nicht erforderlich. Wenn sie weggelassen wird, wird die nächste Anweisung ausgeführt.

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>20 IF A=10 THEN 40
>30 PRINT A
```

In diesem Beispiel wird Zeile 40 ausgeführt, wenn A gleich 10 ist. Wenn A ungleich 10 ist, wird Zeile 30 ausgeführt.

NEXT

Funktion:

Mit der NEXT-Anweisung wird ein Rücksprung an den Beginn der FOR-TO-(STEP)-NEXT-Schleife sowie die Addition des Indexinkrementswertes zum Index veranlaßt. Der aktuelle Indexwert wird anschließend mit dem Indexgrenzwert verglichen, um festzustellen, ob eine weitere Schleife abgearbeitet werden muß.

Syntax:

NEXT

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>5  E=0 : C=10 : D=2
>10 FOR A=E TO C STEP D
>20 PRINT A
>30 NEXT A
>40 END
>RUN

0
2
4
6
8
10

READY
>

>NEW

>1  REM EXAMPLE PROGRAM
>10 FOR I = 0 TO 8 STEP 2
>20 PRINT I
>30 NEXT I
>40 END

>RUN

 0
 2
 4
 6
 8
```

ON-GOTO

Funktion:

Mit der ON-GOTO-Anweisung wird die Steuerung an die durch die GOTO-Anweisung spezifizierte Zeile(n) übertragen, wenn der Wert des Ausdrucks hinter der ON-Anweisung im BASIC-Programm enthalten ist.

Syntax:

ON [ausdr] GOTO [zeil nr]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 ON Q GOTO 100,200,300
```

Die Steuerung wird an Zeile 100 übertragen, wenn Q gleich 0 ist. Wenn Q gleich 1 ist, erfolgt ein Sprung zu Zeile 200, und wenn Q gleich 2 ist, ein Sprung zu Zeile 300 usw. Alle Hinweise zur GOTO-Anweisung gelten in gleicher Weise für die ON-GOTO-Anweisung. Wenn Q kleiner als 0 ist, erscheint die Fehlermeldung **`ERROR: BAD ARGUMENT`**, und das BASIC-Modul wird in den Befehlsmodus geschaltet. Wenn Q größer als die in der GOTO-Anweisung spezifizierte Zeilennummer ist, wird die Fehlermeldung **`ERROR: BAD SYNTAX`** angezeigt. Die ON-GOTO-Anweisung ermöglicht eine bedingte Verzweigung innerhalb des BASIC-Programms.

Kapitel 8

Ausführungssteuerungs- und Interruptfunktionen

In diesem Kapitel sind die Befehle beschrieben und dargestellt, die innerhalb eines BASIC-Programms und von der Befehlszeile aus zur Steuerung des Datenflusses und der Programmübertragung zwischen dem ROM- und dem RAM-Speicher verwendet werden. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 8.A enthalten.

Tabelle 8.A
Kapitelinhalt

CALL 16 – Aktivierung eines DF1-Datenpaketinterrupts

Funktion:

Mit CALL 16 wird die DF1-Datenpaketinterrupt-Funktion aktiviert. Ein Argument wird mit einer PUSH-Anweisung übertragen. Mit der POP-Anweisung werden keine Argumente übertragen. Das Eingangsargument ist die beginnende BASIC-Zeilennummer der Interrupt-Routine, an die ein Sprung ausgeführt werden soll, wenn ein gültiges DF1-Datenpaket im Puffer des PRT2-Ports empfangen wird. Das Datenpaket sollte innerhalb der Interrupt-Routine abgearbeitet werden. Durch die Ausführung eines RETI-Befehls innerhalb der Routine wird ein Rücksprung an die Stelle im Programm veranlaßt, die vor Eintreten des Interrupts ausgeführt wurde. Dieser Befehl ist nur dann wirksam, wenn das DF1-Protokoll aktiviert ist (CALL 108). Außerdem muß die Brücke JW4 so eingestellt sein, daß DF1 für den PRT2-Port aktiviert ist.

Wenn dieser Aufruf aktiviert ist, überprüft der Prozessor den PRT2-Port am Ende jeder BASIC-Zeile, um festzustellen, ob eine DF1-Nachricht empfangen wurde.

Wenn CALL 16 aktiviert ist und durch CALL 122 oder CALL 123 der Empfang eines DF1-Datenpakets veranlaßt wird, erhalten Sie zwar den DF1-Datenpaketinterrupt, das DF1-Datenpaket wird aber aus dem Eingangspuffer entfernt.

Interrupts werden deaktiviert, wenn sich das BASIC-Modul im Befehlsmodus befindet. Standardmäßig ist CALL 16 beim Aufruf des Run-Modus deaktiviert und muß jedesmal, wenn in den Run-Modus umgeschaltet wird, erneut aktiviert werden.

Syntax:

PUSH [BASIC-Zeilennummer]
CALL 16

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PACKET INTERRUPT
>20 PUSH 800: REM LINE NUMBER OF START OF DF1 INTERRUPT ROUTINE
>30  CALL 16
>800 (BEGINNING OF INTERRUPT ROUTINE)
 :   (PROCESS THE PACKET)
>850 RETI
```

CALL 17 – Deaktivierung eines DF1-Datenpaketinterrupts

Funktion:

Mit CALL 17 wird die mit CALL 16 aktivierte DF1-Datenpaketinterrupt-Funktion deaktiviert. Diese Routine verfügt über keine Eingangs- und Ausgangsargumente.

Syntax:

CALL 17

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM DISABLE DF1 PACKET INTERRUPT ENABLED WITH CALL 16
>20 CALL 17
```

CALL 20 – Aktivierung eines Prozessorinterrupts

Funktion:

Mit CALL 20 wird ermöglicht, daß der Prozessor einen Interrupt im BASIC-Modul auslöst. Ein Argument wird mit einer PUSH-Anweisung übertragen. Mit der POP-Anweisung werden keine Argumente übertragen. Das Argument der PUSH-Anweisung ist die beginnende BASIC-Zeilennummer der Interrupt-Routine, an die ein Sprung ausgeführt werden soll, wenn Wort 0, Bit 15 der CPU-Ausgangsdatentafel von einem niedrigen Wert auf einen hohen Wert übergeht. Das BASIC-Modul stellt diesen Übergang automatisch fest und veranlaßt einen Sprung in die Interruptroutine. Durch die Ausführung eines RETI-Befehls innerhalb der Routine wird ein Rücksprung an die Stelle im Programm veranlaßt, die vor Eintreten des Interrupts ausgeführt wurde.

Am Ende jeder BASIC-Zeile überprüft das BASIC-Modul Wort 0, Bit 15, des CPU-Ausgangsfiles und erzeugt einen Interrupt, wenn das Bit auf 1 übergeht. Die CPU-Einheit muß das Interrupt-Anforderungsbit mindestens zehn Millisekunden lang auf 0 halten, bevor ein Interrupt angefordert wird. Das Bit muß mindestens eine Abfrage lang gesetzt (1) sein, damit es vom BASIC-Modul erkannt werden kann.

Wenn vor der vollständigen Abarbeitung des Interrupts ein neuer Interrupt festgestellt wird, wird der neue Interrupt als anstehend gekennzeichnet, wobei immer nur ein Interrupt anstehen kann.

Interrupts werden deaktiviert, wenn sich das BASIC-Modul im Befehlsmodus befindet. Standardmäßig ist CALL 20 beim Aufruf des Run-Modus deaktiviert und muß jedesmal, wenn in den Run-Modus umgeschaltet wird, erneut aktiviert werden.

Syntax:

PUSH [BASIC-Zeilennummer]
CALL 20

Beispiel:

```
>1    REM EXAMPLE PROGRAM
>10   REM ENABLE PROCESSOR INTERRUPTS
>20   PUSH 1000 : REM LINE NUMBER OF START OF PROCESSOR
      INTERRUPT ROUTINE
>30   CALL 20
>1000 (BEGINNING OF THE PROCESSOR INTERRUPT ROUTINE)
 :
>1050 RETI
```

CALL 21 – Deaktivierung eines Prozessorinterrupts

Funktion:

Mit CALL 21 wird die mit CALL 20 aktivierte Prozessorinterrupt-Funktion deaktiviert. Diese Routine verfügt über keine Eingangs- und Ausgangsargumente.

Syntax:

CALL 21

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM DISABLE PROCESSOR INTERRUPTS ENABLED WITH CALL 20
>20 CALL 21
```

CALL 26 – Erteilung eines Interrupts an den SLC-Prozessor

Funktion:

Mit CALL 26 wird ein Interrupt an einen Prozessor SLC 5/02™ und höher generiert. Mit der PUSH-Anweisung wird kein Argument übertragen, während mit der POP-Anweisung ein Argument übertragen wird. Das Argument der POP-Anweisung ist der Status des SLC-Prozessors. Bei der Ausführung dieses Aufrufs wird vom BASIC-Modul ein E/A-Ereignisinterrupt erteilt, der den normalen Betriebszyklus des Prozessors unterbricht, um eine spezifizierte Subroutine abzufragen. Durch diesen Interrupt wird der SLC-Prozessor veranlaßt, den in der Steckplatzkonfiguration des BASIC-Moduls enthaltenen File der Interrupt-Subroutine (ISR-numerierter File) auszuführen. Das BASIC-Modul verweilt in dieser Aufrufroutine, bis vom Prozessor eine Interruptbestätigung eingeht. Jedesmal, wenn der SLC-Prozessor unterbrochen werden soll, muß CALL 26 im BASIC-Programm ausgeführt werden.

Dieser Aufruf ist nur dann funktionsfähig, wenn sich der SLC-Prozessor im Run-Modus befindet.

Nachdem das BASIC-Modul diesen Aufruf erteilt, kann es bis zu fünf Millisekunden dauern, bevor der Interrupt ausgeführt wird.

Die POP-Anweisung enthält den Status des SLC-Prozessors:

- 0 = SLC-Prozessor bestätigt den Interrupt, hat jedoch die Interrupt-Routine möglicherweise noch nicht ausgeführt.
- 1 = SLC-Prozessor brach den Interrupt ab.
- 2 = SLC-Prozessor befindet sich nicht im Run-Modus.
- 3 = Interrupts werden vom SLC-500-Prozessor mit fester Hardwarekonfiguration und vom SLC-5/01™-Prozessor nicht unterstützt.

Syntax:

CALL 26
POP[SLC-Prozessorstatus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE BASIC MODULE INTERRUPT TO THE SLC PROCESSOR
    TO      EXECUTE THE ISR FILE
>20 CALL 26
>30 POP S : REM SLC PROCESSOR STATUS
```

CALL 38 – Erweiterter ONERR-Neustart

Funktion:

Mit CALL 38 wird die Art der durch die ONERR-Funktion eingegrenzten und bearbeiteten Fehler erweitert. Ein Argument wird mit einer PUSH-Anweisung übertragen. Mit der POP-Anweisung werden keine Argumente übertragen. Mit diesem Aufruf kann das Modul einen Sprung in eine Fehlerbehandlungsroutine veranlassen, wenn ein arithmetischer Überlauf, eine Division durch Null oder ein ungültiges Argument festgestellt wird. Bei allen weiteren Fehlern wird das BASIC-Modul in den Befehlsmodus geschaltet und die Ausführung des Programms gestoppt. Wenn CALL 38 aktiviert ist, wird bei allen Fehlern, mit Ausnahme von hardwarespezifischen Störungen (Watchdog-, RAM-Fehler usw.) die in der ONERR-Funktion spezifizierte Routine aufgerufen, ohne daß das Modul in den Befehlsmodus geschaltet wird. Mit dieser Routine kann der Fehler beseitigt und die normale Programmabarbeitung wieder aufgenommen werden. Wenn durch einen Fehler ein Neustart verursacht wird, werden die Stapel gelöscht und die Variablen und Ports nicht neu initialisiert. Dieser Aufruf ist nur dann wirksam, wenn im Programm der ONERR-Befehl ausgeführt wird.

Wie im folgenden dargestellt, wird mit der PUSH-Anweisung festgelegt, ob diese erweiterte ONERR-Funktion aktiviert oder deaktiviert ist:

- 0 = Deaktivierung des erweiterten ONERR-Neustarts
- 1 (oder ein anderer Wert) = Aktivierung des erweiterten Neustarts

Dieser Aufruf wird zurückgesetzt, wenn das BASIC-Modul in den Befehlsmodus geschaltet wird. Er muß jedesmal, wenn der Run-Modus aufgerufen wird, erneut aktiviert werden.

Syntax:

PUSH [0 oder 1]
CALL 38

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE EXPANDED ONERR FUNCTION
>20  ONERR 160
>30  PUSH 1
>40  CALL 38
>50  CALL 53: REM GET DATA FROM OUTPUT IMAGE
>60  PUSH 201: REM ADDRESS OF SECOND WORD IN BUFFER
>70  CALL 14: REM GET DATA FROM INPUT BUFFER
>80  POP X: REM VALUE FROM INPUT BUFFER
>90  A=(X*2.499733)-8191.625
>100 PUSH A: REM RESULT OF ABOVE CALCULATION
>110 PUSH 201: REM WORD NUMBER OF BASIC OUTPUT BUFFER
>120 CALL 24: REM BASIC FLOATING POINT TO 16-BIT SIGNED
     INTEGER
>130 CALL 54: REM OUTPUT BUFFER TO SLC INPUT FILE
>140 POP Y: REM SLC PROCESSOR STATUS
>150 IF (Y<>0) THEN PRINT "PROCESSOR NOT IN RUN MODE"
>160 GOTO 50
>170 PRINT "ERROR CODE WAS",XBY(257) : REM BEGINNING OF
     ONERR       ROUTINE
>180 GOTO 50
```

Im vorausgehenden Beispiel fehlt eine POP-Anweisung für CALL 53. Dadurch wird ein Fehler des A-Stapels verursacht, was normalerweise dazu führen würde, daß der Prozessor in den Befehlsmodus umgeschaltet wird. Bei Eintreten dieses Fehlers wird der Fehlercode angezeigt und die Abarbeitung des Programms fortgesetzt.

CALL 70 – Sprung aus einem ROM- in ein RAM-Programm

Funktion:

Mit CALL 70 wird der Sprung aus einem laufenden ROM-Programm an den Beginn eines RAM-Programms während der Programmabarbeitung ermöglicht. Die Übertragung von Argumenten in PUSH- und POP-Anweisungen ist nicht erforderlich.

Wichtig: Die erste Zeile des RAM-Programms wird nicht ausgeführt. Es empfiehlt sich, diese Zeile deshalb als Kommentar zu schreiben.

Syntax:

CALL 70

Beispiel:

```
READY
>LIST
1  REM EXAMPLE PROGRAM
10 REM SAMPLE ROM PROGRAM FOR CALL 70
20 PRINT "NOW EXECUTING ROM 5"
30 CALL 70 :  REM GO EXECUTE RAM
40 END

READY
>RUN

NOW EXECUTING ROM 5
NOW EXECUTING RAM

READY
>LIST
1  REM EXAMPLE PROGRAM
10 REM SAMPLE RAM PROGRAM FOR CALL 70
20 PRINT "NOW EXECUTING RAM"
30 END

READY
```

CALL 71 – Sprung aus einem ROM-/RAM-Programm in ein ROM-Programm

Funktion:

Mit CALL 71 wird der Sprung aus einem laufenden ROM- oder RAM-Programm an den Beginn eines ROM-Programms ermöglicht. Ein Argument (die Nummer des ROM-Programms) wird mit der PUSH-Anweisung spezifiziert. Die POP-Anweisung ist nicht erforderlich. Ist die ROM-Nummer nicht vorhanden, erscheint eine Fehlermeldung (ungültiges Programm) und das Modul wird in den Befehlsmodus zurückgeschaltet.

Wichtig: Die erste Zeile des ROM-Programms wird nicht ausgeführt. Es empfiehlt sich, diese Zeile deshalb als Kommentar zu schreiben.

Syntax:

PUSH [Nummer des ROM-Programms]
CALL 71

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM THIS ROUTINE WILL CALL AND EXECUTE A ROM ROUTINE
>20 INPUT “ENTER ROM ROUTINE TO EXECUTE”,N
>30 PUSH N
>40 CALL 71
>50 END

>RUN

ENTER ROM ROUTINE TO EXECUTE 4
```

An dieser Stelle erfolgt die Ausführung von ROM 4, sofern dieses Programm vorhanden ist. Ist die angeforderte ROM-Routine nicht vorhanden, erscheint die folgende Anzeige:

```
PROGRAM NOT FOUND.
READY
>
```

CALL 72 – Rücksprung zur RAM-/ROM-Routine

Funktion:

Mit CALL 72 wird der Rücksprung zu der ROM- bzw. RAM-Routine ermöglicht, mit der die aktuelle ROM- bzw. RAM-Routine aufgerufen wurde. Die Abarbeitung beginnt in der Zeile, die unmitelbar auf die Zeile mit dem CALL-Befehl folgt. Die Spezifikation von Argumenten mit der PUSH- bzw. POP-Anweisung ist nicht erforderlich. Diese Routine kann nicht verschachtelt werden. Die Programmsteuerung geht an die Zeile zurück, die unmittelbar auf den CALL-Befehl des vorhergehenden Programms folgt.

Wichtig: In der ROM- bzw. RAM-Routine muß eine folgende Zeile vorhanden sein; anderenfalls können unvorhersehbare Ereignisse auftreten, die den Inhalt des RAM-Speichers zerstören können. Sie sollten deshalb sicherstellen, daß nach CALL 70 bzw. 71 mindestens eine END-Anweisung programmiert ist.

Syntax:

CALL 72

Beispiel:

Programm in ROM 1:

```
>1  REM EXAMPLE PROGRAM
>10 REM SAMPLE PROG FOR CALL 72
>20 PRINT “NOW EXECUTING ROM 1”
>30 PUSH 3
>40 CALL 71 : REM EXECUTE ROM 3 THEN RETURN
>50 PRINT “EXECUTING ROM 1 AGAIN”
>60 END
```

Programm in ROM 3:

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “NOW EXECUTING ROM 3”
>20 CALL 72
>30 END
```

Bei Wahl von ROM 1:

```
>RUN

NOW EXECUTING ROM 1
NOW EXECUTING ROM 3
EXECUTING ROM 1 AGAIN

READY
>
```

GOSUB

Funktion:

Mit der GOSUB-Anweisung wird das BASIC-Modul veranlaßt, die Programmsteuerung an die Zeile zu übertragen, deren Zeilennummer [Zeil-Nr] hinter ger GOSUB-Anweisung spezifiziert wurde. Darüber hinaus speichert GOSUB die Adresse der ihr folgenden Anweisung auf dem Steuerstapel. Somit können Sie mit einer RETURN-Anweisung einen Rücksprung zu der Anweisung ausführen, die auf die zuletzt ausgeführte GOSUB-Anweisung folgt. GOSUB-Anweisungen können bis zu neunmal verschachtelt werden.

Der Steuerstapel (C-Stapel) speichert alle mit der Schleifensteuerung zusammenhängende Daten (Beispiel: DO-WHILE, DO-UNTIL, FOR-NEXT und BASIC-Subroutinen). Der Steuerstapel umfaßt 157 Bytes. DO-WHILE- und DO-UNTIL-Schleifen sowie GOSUB-Befehle verwenden drei Bytes und FOR-NEXT-Schleifen 17 Bytes des Steuerstapels.

Wichtig: Bei einer übermäßigen Verschachtelung werden die Grenzen des Steuerstapels überschritten. Dies verursacht einen Fehler, und das Modul wird in den Befehlsmodus geschaltet.

Syntax:

GOSUB [Zeil-Nr]

Beispiele:

Einfache Subroutine

```
READY
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 5
>20  GOSUB 100
>30  NEXT I
>40  END
>100 PRINT I
>110 RETURN

READY
>RUN

 1
 2
 3
 4
 5

READY
>NEW
```

Verschachtelte Subroutine

```
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 3
>20  GOSUB 100
>30  NEXT I
>40  END
>100 REM USER SUBROUTINE HERE
>105 PRINT I,
>110 GOSUB 200
>120 RETURN
>200 REM START OF NESTED SUBROUTINE
>210 PRINT I*I
 220 RETURN

READY
>RUN

 1  1
 2  4
 3  9

READY
>
```

ONERR

Funktion:

Mit der ONERR-Anweisung können möglicherweise auftretende arithmetische Fehler während der Programmausführung bearbeitet werden. Mit dieser Anweisung werden nur Fehler eingegrenzt, die auf einen arithmetischen Überlauf, Division durch Null oder ein ungültiges Argument zurückzuführen sind. Alle weiteren Fehler werden nicht eingegrenzt und veranlassen die Umschaltung des BASIC-Moduls in den Befehlsmodus. Tritt ein arithmetischer Fehler nach der Ausführung der ONERR-Anweisung auf, veranlaßt der Modulinterpreter, daß die Programmsteuerung an die Zeilennummer [zeil nur] nach der ONERR-Anweisung übertragen wird. Die Handhabung von Fehlern erfolgt in der auf die jeweilige Anwendung abgestimmten Weise. Während eines Eingangsbefehls eingegebene ungültige Daten werden von ONERR nicht eingegrenzt, sondern führen zur Fehlermeldung **`TRY AGAIN`** oder **`EXTRA IGNORED`**. Eine ausführlichere Beschreibung der ONERR-Anweisung ist auf Seite 8-6 unter “CALL 38” enthalten.

Nach der Ausführung der ONERR-Anweisung können Sie durch die Überprüfung der externen Speicheradresse 257 (101H) feststellen, um welchen Fehlertyp es sich handelt.

Die Fehlercodes lauten:

- **FEHLERCODE = 10 – DIVISION DURCH NULL**
- **FEHLERCODE = 20 – ARITHM. ÜBERLAUF oder UNTERLAUF**
- **FEHLERCODE = 40 – UNGÜLTIGES ARGUMENT**

Mit Hilfe einer XBY(257)-Anweisung können Sie diese Adresse untersuchen.

Syntax:

ONERR [Zeil-Nr]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  ONERR 500
>20  FOR I = 5 TO  0 STEP –1
>30  PRINT 1/I
>40  NEXT I
>50  END
>500 PRINT “ERROR CODE WAS ”,XBY(257)
>510 END

READY
>RUN

 .2
 .25
 .33333333
 .5
 1
ERROR CODE WAS  10

READY
>
```

In diesem Beispiel kann die END-Anweisung durch eine GOTO-Anweisung ersetzt werden, um ein vom Anwender programmiertes Fehlerbeseitigungsverfahren zu implementierten.

ON-GOSUB

Funktion:

Mit der ON-GOSUB-Anweisung wird die Steuerung in die durch die GOSUB-Anweisung spezifizierte(n) Zeile(n) übertragen, wenn der Wert des Ausdrucks hinter der ON-Anweisung im BASIC-Programm abgearbeitet wird.

Syntax:

ON [Ausdr] GOSUB [Zeil-Nr], [Zeil-Nr],...[Zeil-Nr]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 ON Q GOSUB 100,200,300
```

Wenn Q gleich 0 ist, wird die Steuerung auf die Zeilennummer 100 übertragen. Wenn Q gleich 1 ist, wird die Steuerung auf Zeilennummer 200 übertragen. Wenn Q gleich 2 ist, wird die Steuerung auf Zeilennummer 300 übertragen usw. Alle Hinweise zur GOSUB-Anweisung gelten in gleicher Weise für die ON-Anweisung. Wenn Q kleiner als 0 ist, wird die Fehlermeldung **ERROR: BAD ARGUMENT** angezeigt. Wenn Q größer als die in der GOSUB-Anweisung spezifizierte Anzahl von Zeilennummern ist, erscheint die Fehlermeldung **ERROR: BAD SYNTAX.** Die ON-GOSUB-Anweisung ermöglicht “bedingte Verzweigungen” innterhalb des BASIC-Programms.

ONTIME

Funktion:

Mit der Anweisung ONTIME [Ausdr], [Zeil-Nr] wird die Inkompatibilität zwischen dem Zeitwerk bzw. den Zählern des Mikroprozessors und dem BASIC-Modul ausgeglichen. Das BASIC-Modul kann eine Zeile innerhalb von wenigen Millisekunden ausführen, während das Zeitwerk/die Zähler des Mikroprozessors in Mikrosekunden arbeiten. Die ONTIME-Anweisung generiert einen Interrupt, sobald der Sonderfunktionsoperator TIME größer als oder gleich dem Ausdruck nach der ONTIME-Anweisung ist.

Nur der Ganzzahlanteil von TIME wird mit dem Ganzzahlanteil des Ausdrucks, der die Sekunden angibt, verglichen. Dieser Vergleich erfolgt am Ende (CR oder :) jeder BASIC-Zeile. Der Interrupt ermöglicht die zwangsweise Ausführung der GOSUB-Anweisung und damit den Sprung in die Zeile, die in der ONTIME-Anweisung hinter dem Ausdruck [Ausdr] durch [Zeil-Nr] spezifiziert wurde.

Ein Eingangsbefehl bzw. eine CALL-Routine wird mit der ONTIME-Anweisung nicht unterbrochen. Da in der ONTIME-Anweisung der Sonderfunktionsoperator TIME verwendet wird, müssen Sie die CLOCK1-Anweisung ausführen, damit ONTIME funktionsfähig wird. Wenn CLOCK1 nicht ausgeführt wird, erfolgt keine Inkrementierung des Sonderfunktionsoperators TIME.

Syntax:

ONTIME [Ausdr], [Zeil-Nr]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  TIME = 0
>15  DBY(71) = 0
>20  CLOCK1
>30  ONTIME 2,100
>40  DO
>50  WHILE TIME < 10
>60  CLOCK0
>70  END
>100 PRINT "TIMER INTERRUPT AT - ",TIME, " SECONDS"
>110 ONTIME TIME+2,100
>120 RETI

READY
>RUN

TIMER INTERRUPT AT -  2.01  SECONDS
TIMER INTERRUPT AT -  4.005  SECONDS
TIMER INTERRUPT AT -  6.015  SECONDS
TIMER INTERRUPT AT -  8.01  SECONDS
TIMER INTERRUPT AT -  10.01  SECONDS
```

In diesem Beispiel weicht die ausgegebene Zeit um 0,01 Sekunden von der Soll-Ausgabezeit ab. Dies ist auf das im Beispiel verwendete Terminal zurückzuführen, dessen Baudrate 19200 beträgt, was bei der Ausgabe zu einer Verzögerung von 0,01 Sekunden führt.

Um den ONTIME-Interrupt während eines Bruchteils einer Sekunde auszuführen, müssen Sie DBY(71) = X verwenden, wobei gilt: X = 0 bis 200. Jeder Zählwert stellt ein Zeitintervall von fünf Millisekunden dar.

PUSH

Funktion:

Mit der PUSH-Anweisung werden ein bzw. mehrere arithmetische Ausdrücke auf dem Argumentstapel des BASIC-Moduls gespeichert. Diese Anweisung bewertet den arithmetischen Ausdruck bzw. die arithmetischen Ausdrücke hinter der PUSH-Anweisung und legt diese der Reihe nach auf dem Argumentstapel ab.

Zusammen mit der POP-Anweisung ermöglicht diese Anweisung, daß Parameter an CALL-Routinen übertragen werden. Außerdem können mit diesen beiden Anweisungen Parameter an BASIC-Subroutinen übertragen und Variablen mit dem SWAP-Befehl ausgetauscht werden. Der letzte mit der PUSH-Anweisung auf den Argumentstapel übertragene Wert wird mit der POP-Anweisung zuerst vom Stapel gelesen.

Mit einer PUSH-Anweisung, die mehrere Ausdrücke ([Ausdr], [Ausdr],.[Ausdr]) enthält, können mehrere Ausdrücke an den Argumentstapel übertragen werden. Jedem Ausdruck muß ein Komma folgen. Der letzte mit der PUSH-Anweisung an den Argumentstapel übertragene Wert ist der letzte Ausdruck [Ausdr] der PUSH-Anweisung.

Wichtig: Der Argumentstapel kann maximal 33 Fließkommazahlen enthalten, bevor ein Überlauf eintritt.

Syntax:

PUSH [Ausdr], [Ausdr],.[Ausdr]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 A = 10
>20 C = 20
>30 PRINT “A = ”,A,“ AND C = ” C
>40 PUSH A,C
>50 POP A,C
>60 PRINT “A = ”,A,“  AND C = ”,C
>70 END

READY
>RUN

A =  10  AND C =  20
A =  20  AND C =  10

READY
>

>NEW

>1  REM EXAMPLE PROGRAM
>10 PUSH 0
>20 CALL 14
>30 POP W
>40 PRINT W
>50 END

READY
>RUN

 0

READY
>
```

POP

Funktion:

Mit der POP-Anweisung werden Werte aus dem Argumentstapel des BASIC-Moduls gelesen. Der oberste Wert des Argumentstapels wird der hinter der POP-Anweisung spezifizierten Variablen zugeordnet (die Werte werden vom Argumentstapel gelesen) (Beispiel: Inkrementierung der Werte um 6). Mit der PUSH-Anweisung werden Werte auf den Argumentstapel übertragen.

Wichtig: Wird eine POP-Anweisung ausgeführt, und zuvor wurde kein Wert auf den Argumentstapel übertragen, tritt ein A-Stapel-Fehler ein, und das BASIC-Modul wird in den Befehlsmodus geschaltet.

Mit einer POP-Anweisung, die mehrere Ausdrücke ([Ausdr], [Ausdr],.[Ausdr]) enthält, können mehrere Ausdrücke aus dem Argumentstapel gelesen werden. Jedem Ausdruck muß ein Komma folgen.

Syntax:

POP [var], [var],.....[var]

Beispiel:

Siehe PUSH-Anweisung.

Mit der PUSH- und der POP-Anweisung können durch GLOBALE Variablen auftretende Probleme minimiert werden. Diese werden dadurch verursacht, daß im Hauptprogramm und in allen Subroutinen des Hauptprogramms die gleichen Variablennamen verwendet werden (z.B. GLOBALE VARIABLEN). Wenn in einer Subroutine und im Hauptprogramm unterschiedliche Variablen verwendet werden müssen, können Sie eine bestimmte Anzahl von Variablen neu zuordnen (z.B. A=Q), bevor eine GOSUB-Anweisung ausgeführt wird.

Wenn Sie bestimmte Variablennamen speziell für Subroutinen (S1, S2) reservieren und die Variablen, wie im vorhergehenden Beispiel gezeigt, auf den Argumentstapel übertragen, können Sie die durch GLOBALE Variablen verursachten Probleme im BASIC-Modul vermeiden.

In der PUSH- und der POP-Anweisung können sowohl dimensionierte Variablen – A(4) und S1(12) – als auch Skalarvariablen verwendet werden. Dies ist besonders dann sinnvoll, wenn beim Aufruf von CALL-Routinen große Datenmengen mit PUSH- bzw. POP-Anweisungen übertragen werden müssen.

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>40 FOR I=1 TO 64
>50 PUSH I
>60 CALL 10
>70 POP A(I)
>80 NEXT I
```

RETI

Funktion:

Mit der RETI-Anweisung wird ein Interrupt (ONTIME, CALL 16 oder CALL 20), der in einem Programm des BASIC-Moduls abgearbeitet wird, verlassen. Die Funktionsweise der RETI-Anweisung entspricht im wesentlichen der der RETURN-Anweisung, mit der Ausnahme, daß auch ein Software-Interrupt-Flag gelöscht wird, so daß Interrupts erneut quittiert werden. Wird die RETI-Anweisung in der Interrupt-Routine nicht ausgeführt, werden alle zukünftigen Interrupts ignoriert.

Syntax:

RETI

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  TIME=0 : CLOCK1 : ONTIME 2, 100 : DO
>20  WHILE TIME<10 : END
>100 PRINT “TIMER INTERRUPT AT -”, TIME,“ SECONDS”
>110 ONTIME TIME+2, 100 : RETI
>RUN

TIMER INTERRUPT AT - 2.045 SECONDS
TIMER INTERRUPT AT - 4.045 SECONDS
TIMER INTERRUPT AT - 6.045 SECONDS
TIMER INTERRUPT AT - 8.045 SECONDS
TIMER INTERRUPT AT - 10.045 SECONDS

READY
```

RETURN

Funktion:

Mit der RETURN-Anweisung wird ein Rücksprung zu der Anweisung ausgeführt, die auf die zuletzt ausgeführte GOSUB-Anweisung folgt. Um einen Überlauf im Steuerstapel zu vermeiden, sollten Sie für jede GOSUB-Anweisung eine RETURN-Anweisung programmieren. Dies bedeutet, daß eine durch die GOSUB-Anweisung aufgerufene Subroutine eine weitere Subroutine mit einer weiteren GOSUB-Anweisung aufrufen kann.

Syntax:

RETURN

Beispiele:

Einfache Subroutine

```
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 5
>20  GOSUB 100
>30  NEXT I
>40  END
>100 PRINT I
>110 RETURN

READY
>RUN

 1
 2
 3
 4
 5

READY
>
```

Verschachtelte Subroutine

```
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 5
>20  GOSUB 100
>30  NEXT I
>40  END
>100 PRINT I,
>110 GOSUB 200
>120 RETURN
>200 PRINT I*I,
>210 GOSUB 300
>220 RETURN
>300 PRINT I*I*I
>310 RETURN

READY
>RUN

 1  1  1
 2  4  8
 3  9  27
 4  16  64
 5  25  125

READY
>
```

STOP

Funktion:

Mit der STOP-Anweisung wird die Programmausführung an spezifizierten Programmpunkten angehalten, um Variablen zu überprüfen oder zu ändern. Die Fortsetzung der Programmausführung erfolgt mit dem CONT-Befehl. Die STOP-Anweisung ermöglicht eine einfache Fehlersuche innerhalb eines Programms.

Syntax:

STOP

Beispiel:

```
1
STOP – IN LINE 40

>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 100
>20 PRINT I
>30 STOP
>40 NEXT I

READY
>RUN

 1
STOP – IN LINE  40
READY
>CONT

 2
STOP – IN LINE  40
READY
>CONT

 3
STOP – IN LINE  40
READY
>CONT

 4
STOP – IN LINE  40
READY
>
```

Beachten Sie bitte, daß nach der Ausführung der STOP-Anweisung die auf die STOP-Anweisung folgende Zeile und nicht die Zeile, die die STOP-Anweisung enthält, angezeigt wird.

Kapitel 9

Mathematische und Backplane-Umwandlungsfunktionen

In diesem Kapitel werden Befehle beschrieben und veranschaulicht, die die Umwandlung von Ganzzahl- und BASIC-Fließkommawerten ermöglichen. Darüber hinaus ist eine Beschreibung und Darstellung der Befehle enthalten, die zur Übertragung von Daten vom Ausgangspuffer des BASIC-Moduls an das Eingangsabbild des Prozessors bzw. zur Übertragung von Daten aus dem Ausgangsabbild an den Eingangspuffer des BASIC-Moduls verwendet werden. Diese Befehle können in einem BASIC-Programm und von der Befehlszeile aus programmiert werden. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 9.A enthalten.

Tabelle 9.A
Kapitelinhalt

CALL 14 – 16-Bit-Ganzzahl mit Vorzeichen in BASIC-Fließkommawert

Funktion:

Mit CALL 14 wird eine in einer SLC-500-Steuerung verwendete 16-Bit-Ganzzahl mit Vorzeichen in eine BASIC-Fließkommazahl umgewandelt. Das Eingangsargument ist die Adressennummer (0 bis 207) des umzuwandelnden Wortes im Eingangspuffer des BASIC-Moduls. Das Ausgangsargument ist der umgewandelte Wert.

Syntax:

PUSH [Nummer des Wortes im Eingangspuffer des BASIC-Moduls]
CALL 14
POP [umgewandelter Wert]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>20 PUSH 0 : REM CONVERT 1ST WORD OF BASIC INPUT BUFFER
>30 CALL 14 : REM DO 16–BIT SIGNED TO F.P. CONVERSION
>40 POP W : REM GET CONVERTED VALUE
>50 PRINT W
>RUN

 0

READY
>
```

CALL 15 – 16-Bit-Ganzzahl ohne Vorzeichen in BASIC-Fließkommawert

Funktion:

Mit CALL 15 wird eine in einer SLC-500-Steuerung verwendete 16-Bit-Ganzzahl ohne Vorzeichen in eine BASIC-Fließkommazahl umgewandelt. Das Eingangsargument ist die Adressennummer (0 bis 207) des umzuwandelnden Wortes im Eingangspuffer des BASIC-Moduls. Das Ausgangsargument ist der umgewandelte Wert.

Syntax:

PUSH [Nummer des Wortes im Eingangspuffer des BASIC-Moduls]
CALL 15
POP [umgewandelter Wert]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>50 PUSH 9 : REM CONVERT 10TH WORD OF BASIC INPUT BUFFER
>60 CALL 15 : REM DO 16–BIT UNSIGNED INTEGER TO
    F.P. CONVERSION
>70 POP W : REM GET CONVERTED VALUE
>80 PRINT W
>RUN

 0

READY
>
```

CALL 24 – BASIC-Fließkommawert in 16-Bit-Ganzzahl mit Vorzeichen

Funktion:

Mit CALL 24 wird eine BASIC-Fließkommazahl in eine 16-Bit-Ganzzahl mit Vorzeichen umgewandelt und das Ergebnis im Ausgangspuffer des BASIC-Moduls abgelegt. Der erste mit der PUSH-Anweisung übertragene Wert ist die Datenvariable. Der zweite mit der PUSH-Anweisung übertragene Wert ist die Adressennummer (0 bis 207) des Wortes im Ausgangspuffer des BASIC-Moduls.

Wichtig: Wenn Sie versuchen, Daten in das Wort 200 des BASIC-Ausgangspuffers zu schreiben, erscheint eine Fehlermeldung, und das BASIC-Modul wird in den Befehlsmodus zurückgeschaltet. Die Bits des Wortes 200 sind belegt.

Der Bruchteil des BASIC-Fließkommawertes wird abgeschnitten. Wenn der Wert der BASIC-Fließkommazahl kleiner als -32768 ist, wird der Wert -32768 im Ausgangspuffer des BASIC-Moduls gespeichert. Wenn der Wert größer als +32767 ist, wird der Wert +32767 im Ausgangspuffer des BASIC-Moduls gespeichert. Als Programmierer müssen Sie vor der Umwandlung den Wertebereich überprüfen.

Syntax:

PUSH [umzuwandelnder Wert]
PUSH [Nummer des Wortes im Ausgangspuffer des BASIC-Moduls]
CALL 24

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 W = 17
>40 PUSH W : REM THE VALUE TO BE CONVERTED
>50 PUSH 0 : REM 1ST WORD OF BASIC OUTPUT BUFFER
>60 CALL 24 : REM DO THE F.P. TO 16–BIT SIGNED CONVERSION

READY
>
```

CALL 25 – BASIC-Fließkommawert in 16-Bit-Binärwert

Funktion:

Mit CALL 25 wird ein BASIC-Fließkommawert zwischen 0 und 65535 in den entsprechenden 16-Bit-Binärwert umgewandelt. Das Ergebnis wird anschließend im Ausgangspuffer des BASIC-Moduls gespeichert. Der erste mit der PUSH-Anweisung übertragene Wert ist der umzuwandelnde Wert. Der zweite mit der PUSH-Anweisung übertragene Wert ist die Adressennummer (0 bis 207) des Wortes im Ausgangspuffer des BASIC-Moduls.

Wichtig: Wenn Sie versuchen, Daten in das Wort 200 des BASIC-Ausgangspuffers zu schreiben, erscheint eine Fehlermeldung, und das BASIC-Modul wird in den Befehlsmodus zurückgeschaltet. Die Bits des Wortes 200 sind belegt.

Der Bruchteil des BASIC-Fließkommawertes wird abgeschnitten. Wenn der Wert der BASIC-Fließkommazahl kleiner als 0 ist, wird der Wert 0 im Ausgangspuffer des BASIC-Moduls gespeichert. Wenn der Wert größer als +65535 ist, wird der Wert +65535 im Ausgangspuffer des BASIC-Moduls gespeichert. Als Programmierer müssen Sie vor der Umwandlung den Wertebereich überprüfen.

Syntax:

PUSH [umzuwandelnder Wert]
PUSH [Nummer des Wortes im Ausgangspuffer des BASIC-Moduls]
CALL 25

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 9E+1 : REM THE VALUE TO BE CONVERTED
>50 PUSH 0 : REM 1ST WORD OF BASIC OUTPUT BUFFER
>60 CALL 25 : REM DO F.P. TO 16–BIT BINARY CONVERSION
>
READY
>
```

Kapitel 10

Uhrzeit-/Kalenderfunktionen

Dieses Kapitel enthält eine Beschreibung und Darstellung der Befehle, die zur Einstellung und Anzeige der Echtzeituhr-/Kalenderfunktion in einem BASIC-Programm bzw. von der Befehlszeile aus programmiert werden können. Die entsprechende Mnemonik ist in Tabelle 10.A aufgeführt.

Tabelle 10.A
Kapitelinhalt

CALL 40 – Einstellung der Uhrzeit

Funktion:

Mit CALL 40 wird die Uhrzeit eingestellt:

- H = Stunden (0 bis 23; nur eine 24-Stunden-Uhr ist verfügbar)
- M = Minuten (0 bis 59)
- S = Sekunden (0 bis 59)

Syntax:

PUSH [Stunden]
PUSH [Minuten]
PUSH [Sekunden]
CALL 40

Beispiel:

Stellen Sie die Systemuhr auf 13.35 Uhr ein (nur eine 24-Stunden-Uhr ist verfügbar)

```
>1  REM EXAMPLE PROGRAM
>10 H = 13 : M = 35 : S = 00
>20 REM HOURS = 13, MINUTES = 35, SECONDS = 00
>30 PUSH H,M,S
>40 CALL 40
READY
>RUN
READY
>
```

CALL 41 – Einstellung des Datums

Funktion:

Mit CALL 41 wird das Datum eingestellt:

- D = Tag
- M = Monat
- Y = Jahr

Mit der PUSH-Anweisung werden drei Werte in den Stapel eingespeichert. Die POP-Anweisung ist nicht erforderlich.

Syntax:

PUSH [Tag]
PUSH [Monat]
PUSH [Jahr]
CALL 41

Beispiel:

Stellen sie das Datum auf den 16. Juni 1991 ein.

```
>1   REM EXAMPLE PROGRAM
>5   STRING 100,20
>10  H=13 : M=35 : S=00
>20  REM HOURS = 13, MINUTES = 35, SECONDS = 00
>30  PUSH H,M,S
>40  CALL 40
>60  D=16 : MO=6 : Y=91
>70  PUSH D,MO,Y
>80  CALL 41
>90  PUSH 3
>100 CALL 42
>120 PUSH 0
>130 CALL 43
>140 PRINT $(0)

READY
>RUN

16-JUN-91 13:35:00
```

CALL 42 – Einstellung des Wochentages

Funktion:

Mit CALL 42 wird der Wochentag eingestellt, wobei Sonntag der erste und Samstag der 7. Wochentag ist.

Syntax:

PUSH [Wochentag]
CALL 42

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 3: CALL 42:REM DAY IS TUESDAY.
```

CALL 43 – Abruf einer Datum-/Uhrzeit-Zeichenkette

Funktion:

Mit CALL 43 wird das aktuelle Datum und die aktuelle Uhrzeit als Zeichenkette eingelesen. Zur Ausgabe des Datums und der Uhrzeit im Format (TT-MM-JJ HH:MM:SS) müssen Sie mit der PUSH-Anweisung die Nummer der Zeichenkette, in dem die Daten gespeichert werden sollen, festlegen. Sie müssen der Zeichenkette mindestens 18 Zeichen zuordnen. Aus diesem Grund muß die maximale Länge für alle Zeichenketten mindestens 18 Zeichen betragen.

Syntax:

PUSH [Zeichenkettennummer]
CALL 43

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1: CALL 43: REM PUT DATE/TIME IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

16-JUN-91 13:35:00

READY
>
```

CALL 44 – Abruf einer numerischen Datumsausgabe

Funktion:

Mit CALL 44 wird das aktuelle Datum vom Argumentstapel in numerischer Form (drei Ziffern) eingelesen. Für diese Routine ist kein Eingangsargument erforderlich, und es werden drei Variablen ausgegeben. Das Datum wird mit der POP-Anweisung vom Stapel gelesen und in der Reihenfolge Tag – Monat – Jahr zugewiesen.

Syntax:

CALL 44
POP [Tag]
POP [Monat]
POP [Jahr]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM DATE RETRIEVE – NUMERIC EXAMPLE
>20 CALL 44 : REM INVOKE THE UTILITY ROUTINE
>30 POP D,M,Y : REM GET THE DATA FROM THE ARGUMENT STACK
>40 PRINT “CURRENT DATE IS ”,Y,M,D
>50 END

READY
>RUN

CURRENT DATE IS  91  6  19

READY

>
```

CALL 45 – Abruf einer Zeit-Zeichenkette

Funktion:

Mit CALL 45 wird die aktuelle Uhrzeit in Form einer Zeichenkette (HH:MM:SS) abgerufen. Zur Zuordnung der Uhrzeit müssen Sie die Nummer der Zeichenkette mit der PUSH-Anweisung festlegen. Der Zeichenkette müssen mindestens acht Zeichen zugeordnet werden.

Syntax:

PUSH [Zeichenkettennummer]
CALL 45

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1 : CALL 45 : REM PUT TIME IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

06:40:49

READY
>
```

CALL 46 – Abruf einer numerischen Zeitausgabe

Funktion:

Mit CALL 46 wird die aktuelle Uhrzeit in numerischer Form abgerufen. Sie müssen dem Argumentstapel drei Variablen mit dem POP-Befehl entnehmen. Es sind keine Eingangsargumente erforderlich. Die Uhrzeit wird mit der POP-Anweisung aus dem Stapel gelesen und in der Reihenfolge Stunde – Minute – Sekunde zugeordnet.

Syntax:

CALL 46
POP [Stunde]
POP [Minute]
POP [Sekunde]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM TIME IN VARIABLES EXAMPLE : REM GET THE WALL CLOCK
    TIME
>20 CALL 46
>30 POP H,M,S
>40 PRINT  "CURRENT TIME IS ",H,M,S
>50 END

READY
>RUN

CURRENT TIME IS  6  43  7

READY
>
```

CALL 47 – Abruf einer Wochentag-Zeichenkette

Funktion:

Mit CALL 47 wird der aktuelle Wochentag in Form einer Zeichenkette mit drei Zeichen abgerufen. Zur Ausgabe des Wochentages müssen Sie die Nummer der Zeichenkette mit Hilfe der PUSH-Anweisung zuweisen. Einer Zeichenkette müssen mindestens drei Zeichen zugeordnet werden. Die folgenden Zeichenketten werden ausgegeben: SUN, MON, TUE, WED, THU, FRI und SAT.

Syntax:

PUSH [Zeichenkettennummer]
CALL 47

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 0 :CALL 47
>30 PRINT “TODAY IS ”,$(0)

READY
>RUN

TODAY IS FRI

READY
>
```

CALL 48 – Abruf der numerischen Wochentag-Ausgabe

Funktion:

Mit CALL 48 wird der aktuelle Wochentag vom Argumentstapel mit der POP-Anweisung als Zahl (Beispiel: `Sonntag=1, Samstag=7`) abgerufen. Diese Zahl kann mit der POP-Anweisung einer Variablen zugeordnet werden.

Syntax:

CALL 48
POP [Wochentag]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM DAY OF WEEK RETRIEVE – NUMERIC EXAMPLE
>20 CALL 48 : REM INVOKE UTILITY TO GET D.O.W.
>30 POP D
>40 PRINT D
>50 END

READY
>RUN

 5

READY
>
```

CALL 52 – Abruf einer Datum-Zeichenkette

Funktion:

Mit CALL 52 wird das aktuelle Datum in Form einer Zeichenkette (TT-MM-JJ) ausgegeben. Zur Zuordnung des Datums müssen Sie die Nummer der Zeichenkette mit der PUSH-Anweisung übertragen. Der Zeichenkette müssen mindestens neun Zeichen zugeordnet werden.

Syntax:

PUSH [Zeichenkettennummer]
CALL 52

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1 : CALL 52 : REM PUT DATE IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

16-JUN-91

READY
>
```

Kapitel 11

Statusfunktionen

Dieses Kapitel enthält eine Beschreibung und Darstellung der Befehle, die zur Überwachung des BASIC-Modulstatus programmiert werden können. Ferner sind Befehle zur Konfiguration des DF1-Treibers im BASIC-Programm enthalten oder können von der Befehlszeile aus eingegeben werden. Die entsprechende Mnemonik ist in Tabelle 11.A aufgeführt.

Tabelle 11.A
Kapitelinhalt

CALL 36 – Abruf der Zeichenanzahl in den PRT2-Puffern

Funktion:

Mit CALL 36 wird die Zeichenanzahl im jeweils gewählten Puffer des PRT2-Ports abgerufen.

Der zu prüfende Puffer muß mit einer PUSH-Anweisung spezifiziert werden:

- PUSH 1 für den Eingangspuffer
- PUSH 0 für den Ausgangspuffer

Zur Ermittlung der Zeichenanzahl ist eine POP-Anweisung erforderlich.

Syntax:

PUSH [gewählter Puffer]
CALL 36
POP [Zeichenanzahl]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM EXAMINES THE OUTPUT BUFFER
>20 CALL 36
>30 POP X : REM GET THE NUMBER OF CHARACTERS
>40 PRINT “NUMBER OF CHARACTERS IN OUTPUT BUFFER IS”,X
>50 END
READY
>RUN
NUMBER OF CHARACTERS IN OUTPUT BUFFER IS 0
READY
>
```

CALL 51 – Überprüfung des CPU-Ausgangsabbildpuffers

Funktion:

Mit CALL 51 wird überprüft, ob der im BASIC-Modul vorhandene Ausgangsabbildpuffer der SLC-500-Steuerung seit der letzten Überprüfung aktualisiert wurde. (In diesem Fall ist unter "aktualisieren" zu verstehen, daß die Daten von der CPU-Einheit auch dann an diese Puffer übertragen wurden, wenn deren Wert unverändert ist.) Diese Routine erfordert kein Eingangs- und ein Ausgangsargument.

Das Ausgangsargument besitzt den Wert:

- 0, wenn der Logikprozessor keine Daten in den Ausgangsabbildpuffer geschrieben hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt auftrat).
- 1, wenn der Logikprozessor Daten in den Ausgangsabbildpuffer geschrieben hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt auftrat).
- 2, wenn der Logikprozessor diese Funktion nicht unterstützt (z.B. beim SLC-5/01-Prozessor)

Syntax:

CALL 51
POP [Status des Ausgangsabbildpuffers]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>120 CALL 51 : REM WAIT ON SLC
>130 POP S
>140 IF (S = 2) THEN PRINT "THE SLC DOES NOT SUPPORT THIS
     FUNCTION"
>150 IF (S = 2) THEN STOP
>160 IF (S = 0) THEN GOTO 120
>170 PRINT "OUTPUT IMAGE HAS BEEN UPDATED"

READY
>RUN

THE SLC DOES NOT SUPPORT THIS FUNCTION
STOP – IN LINE  160
READY
```

CALL 55 – Überprüfung des CPU-Eingangsabbildpuffers

Funktion:

Mit CALL 55 wird überprüft, ob der im BASIC-Modul vorhandene Eingangsabbildpuffer der SLC-500-Steuerung seit der letzten Überprüfung abgelesen wurde. Diese Routine erfordert kein Eingangs- und ein Ausgangsargument.

Das Ausgangsargument besitzt den Wert:

- 0, wenn der Logikprozessor keine Daten aus dem Eingangsabbildpuffer gelesen hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt auftrat).
- 1, wenn der Logikprozessor Daten aus dem Eingangsabbildpuffer gelesen hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt auftrat).
- 2, wenn der Logikprozessor diese Funktion nicht unterstützt (z.B. beim SLC-5/01-Prozessor)

Syntax:

CALL 55
POP [Status des Eingangsabbildpuffers]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>120 CALL 55 :  REM WAIT ON SLC
>130 POP S
>140 IF (S=2) THEN  PRINT “THE SLC DOES NOT SUPPORT THIS
     FUNCTION”
>150 IF (S=2) THEN  STOP
>160 IF (S=0) THEN  GOTO 120
>170 PRINT “INPUT IMAGE HAS BEEN READ”

READY
>RUN

INPUT IMAGE HAS BEEN READ

READY
>
```

CALL 58 – Überprüfung des M0-Files

Funktion:

Mit CALL 58 wird geprüft, ob der im BASIC-Modul befindliche Modulfile M0 seit der letzten Überprüfung aktualisiert wurde. (In diesem Fall ist unter “aktualisieren” zu verstehen, daß die Daten von der CPU-Einheit auch dann an diese Puffer übertragen wurden, wenn deren Wert unverändert ist.) Diese Routine erfordert kein Eingangs- und ein Ausgangsargument.

Das Ausgangsargument besitzt den Wert:

- 0, wenn der Logikprozessor keine Daten in den Modulfile M0 geschrieben hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde
- 1, wenn der Logikprozessor Daten in den Modulfile M0 geschrieben hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt auftrat).
- 2, wenn der Logikprozessor diese Funktion nicht unterstützt (z.B. beim SLC-5/01-Prozessor).

Syntax:

CALL 58
POP[Schreibstatus des Modulfiles M0]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>120 CALL 58 : REM START WAITING ON M0 UPDATE
>130 POP S
>140 IF (S = 2) THEN PRINT “PROCESSOR DOES NOT SUPPORT THIS
       FUNCTION”
>150 IF (S = 2) THEN STOP
>160 IF (S = 0) THEN GOTO 120
>170 PUSH 64
>180 CALL 56
>190 POP A
>200 PRINT “CALL 56 OUTPUT IS ”,A


READY
>RUN

CALL 56 OUTPUT IS  0
```

CALL 59 – Überprüfung des M1-Files

Funktion:

Mit CALL 59 wird überprüft, ob der im BASIC-Modul vorhandene Modulfile M1 seit der letzten Überprüfung gelesen wurde. Diese Routine erfordert kein Eingangs- und ein Ausgangsargument.

Das Ausgangsargument besitzt den Wert:

- 0, wenn der Logikprozessor keine Daten aus dem Modulfile M1 gelesen hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt auftrat).
- 1, wenn der Logikprozessor Daten aus dem Modulfile M1 gelesen hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt auftrat).
- 2, wenn der Logikprozessor diese Funktion nicht unterstützt (z.B. beim SLC-5/01-Prozessor).

Syntax:

CALL 59
POP [Lesestatus des Modulfiles M1]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>100 PUSH 64
>110 CALL 56 :  REM COPY BASIC OUTPUT BUFFER TO M1
>120 POP A
>130 IF (A=2) THEN  PRINT “SLC DOES NOT SUPPORT THIS
     FUNCTION”
>140 IF (A=2) THEN  STOP
>150 IF (A<>0)THEN GOTO 110
>160 CALL 59 :  REM START WAITING NOW
>170 POP S
>180 IF (S=2) THEN  PRINT “SLC DOES NOT SUPPORT THIS
     FUNCTION”
>190 IF (S=2)THEN STOP
>200 IF (S=0) THEN  GOTO 170
>210 PRINT “CALL 59 OUTPUT IS ”,S

READY
>RUN

CALL 59 OUTPUT IS  1
```

CALL 75 – Überprüfung des CPU-Status der SCL-500-Steuerung

Funktion:

Mit CALL 75 wird der Modus (Run/Program/Test) des SLC-Prozessors überprüft. Es sind keine PUSH-Anweisungen und eine POP-Anweisung erforderlich.

Die POP-Werte des SLC-5/01-Betriebsmodus lauten:

- 0 = SLC-Prozessor im Run-Modus
- 1 = SLC-Prozessor nicht im Run-Modus

Die POP-Werte des SLC-5/02-Betriebsmodus lauten:

- 0 = SLC-Prozessor im Run-Modus
- 1 = SLC-Prozessor im Program-Modus
- 2 = SLC-Prozessor im Test-Modus

Syntax:

CALL 75
POP [Prozessormodus]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>100 CALL 75
>110 POP S
>120 IF (S=0) THEN PRINT “SLC IS IN RUN MODE”
>130 IF (S=1) THEN PRINT “SLC IS NOT IN RUN MODE”
>140 IS (S=2) THEN PRINT “SLC IS IN TEST MODE”

READY
>RUN

SLC IS IN RUN MODE

READY
>
```

CALL 80 – Überprüfung der Batterie

Funktion:

Mit CALL 80 wird der Batteriezustand des BASIC-Moduls überprüft. Wird nach der Ausführung von CALL 80 in der POP-Anweisung der Wert 0 ausgegeben, ist der Batteriezustand in Ordnung. Bei Ausgabe des Wertes 1 ist die Batteriespannung niedrig.

Syntax:

CALL 80
POP [Batteriestatus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 80
>20 POP C
>30 IF (C<>0) THEN PRINT ”BATTERY LOW!”
>40 END

READY
>RUN

BATTERY LOW!
```

CALL 86 – Überprüfung des dezentralen Schreibstatus des DH-485-Schnittstellenfiles

Funktion:

Mit CALL 86 wird überprüft, ob der gemeinsame DH-485-Schnittstellenfile im BASIC-Modul seit der letzten Überprüfung aktualisiert wurde. Diese Routine erfordert kein Eingangs- und ein Ausgangsargument.

Das Ausgangsargument entspricht dem Wert:

- 0, wenn ein Gerät im seriellen DH-485-Kommunikationsverbund keine Daten in den gemeinsamen seriellen DH-485-Schnittstellenfile geschrieben hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt eintrat).
- 1, wenn ein Gerät im seriellen DH-485-Kommunikationsverbund Daten in den gemeinsamen seriellen DH-485-Schnittstellenfile geschrieben hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt eintrat).

Syntax:

CALL 86
POP [dezentraler Schreibstatus des DH-485-Schnittstellenfiles]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>100 CALL 86 : REM CHECK FILE STATUS
>110 POP X : REM GET THE STATUS
>120 IF(X<>1) THEN GOTO 100 : REM WAIT ON THE DATA

READY
>
```

CALL 87 – Überprüfung des dezentralen Lesestatus des DH-485-Schnittstellenfiles

Funktion:

Mit CALL 87 wird überprüft, ob der gemeinsame DH-485-Schnittstellenfile im BASIC-Modul seit der letzten Überprüfung von einem Gerät im seriellen DH-485-Kommunikationsverbund gelesen wurde. Diese Routine erfordert kein Eingangs- und ein Ausgangsargument.

Das Ausgangsargument entspricht dem Wert:

- 0, wenn ein Gerät keine Daten aus dem gemeinsamen DH-485-Schnittstellenfile gelesen hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt eintrat).
- 1, wenn ein Gerät im seriellen DH-485-Kommunikationsverbund Daten aus dem gemeinsamen DH-485-Schnittstellenfile gelesen hat, nachdem dieser Aufruf zuletzt ausgeführt oder das BASIC-Modul eingeschaltet wurde (unabhängig davon, welcher Zustand zuletzt eintrat).

Syntax:

CALL 87
POP [dezentraler Lesestatus des DH-485-Schnittstellenfiles]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>100 CALL 87 : REM CHECK FILE STATUS
>110 POP X : REM GET THE STATUS
>120 IF (X<>1) GOTO 100: REM WAIT ON DATA TO BE READ

READY
>
```

CALL 95 – Abruf der Zeichenanzahl aus den PRT1-Puffern

Funktion:

Mit CALL 95 wird die Zeichenanzahl im jeweils gewählten Puffer des PRT1-Ports abgerufen.

Der zu prüfende Puffer muß mit einer PUSH-Anweisung spezifiziert werden:

- PUSH 1 für den Eingangspuffer
- PUSH 0 für den Ausgangspuffer

Zur Ermittlung der Zeichenanzahl ist eine POP-Anweisung erforderlich.

Syntax:

PUSH [gewählter Puffer]
CALL 95
POP [Zeichenanzahl]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM EXAMINES THE OUTPUT BUFFER
>20 CALL 95
>30 POP X : REM GET THE NUMBER OF CHARACTERS
>40 PRINT “NUMBER OF CHARACTERS IN PRT1 OUTPUT BUFFER IS ”,X
>50 END

READY
>RUN

NUMBER OF CHARACTERS IN PRT1 OUTPUT BUFFER IS  0

READY
>
```

CALL 97 – Aktivierung des DTR-Signals am PRT2-Port

Funktion:

Mit CALL 97 wird das DTR-Signal (Datenterminal bereit) des PRT2-Ports aktiviert. Dieses Signal ist beim Einschalten standardmäßig aktiviert. Mit diesem Aufruf wird das DTR-Signal erneut aktiviert, wenn es zuvor mit CALL 98 deaktiviert wurde.

Syntax:

CALL 97

Beispiel:

```
>10 REM EXAMPLE PROGRAM
>20 CALL 97 : REM ENABLE DTR SIGNAL


READY
>
```

CALL 98 – Deaktivierung des DTR-Signals am PRT2-Port

Funktion:

Mit CALL 98 wird das DTR-Signal (Datenterminal bereit) des PRT2-Ports deaktiviert.

Syntax:

CALL 98

Beispiel:

```
>10 REM EXAMPLE PROGRAM
>20 CALL 98 : REM DISABLE DTR SIGNAL


READY
>
```

CALL 108 – Aktivierung der DF1-Treiberkommunikation

Funktion:

Mit CALL 108 wird die DF1-Treiberkommunikation über den PRT2-Port aktiviert.

Wichtig: DF1 kann nur aktiviert werden, wenn sich die Brücke JW4 des BASIC-Moduls in der richtigen Position befindet. Weitere Informationen sind im Design- und Integrationshandbuch des SLC 500 BASIC-Moduls (Publikationsnummer 1746-6.1DE) enthalten.

Diese Routine verfügt über sechs Eingangsargumente und kein Ausgangsargument. Das erste Eingangsargument spezifiziert den Betriebscode, der den Betriebsmodus des DF1-Treibers angibt. Der Betriebscode spezifiziert folgende DF1-Parameter:

- Vollduplex- oder Halbduplex-Slave-Betrieb
- Wahl der Erkennung doppelter Datenpakete (DPD)
- Wahl des BCC- oder CRC-Fehlerprüfverfahrens
- Aktivierung eingebetteter Antworten (ER) oder automatisch erkannter eingebetteter Antworten (ADER). Wenn ADER gewählt wird, werden die aktivierten Antworten erst nach dem Empfang einer eingebetteten Antwort übertragen. Hierbei wird vorausgesetzt, daß ein Kommunikationsgerät, das eine ER-Antwort sendet, diese auch empfangen kann. (Dies gilt jedoch nur für den Vollduplex-Betrieb.)
- Wahl der Modem-Handshake-Quittierung:
  - Halbduplex-Optionen:
    kein Handshake (NHS)
    Halbduplex-Modem ohne Dauerträgersignal (HDMwoCC)
    Halbduplex-Modem mit Dauerträgersignal (HDMwCC)
  - Vollduplex-Optionen:
    kein Handshake (NHS)
    Vollduplex-Modem (FDM)

Die zulässigen Werte für den Betriebscode sind 0 bis 11 für den Halbduplexbetrieb und 16 bis 31 für den Vollduplexbetrieb. Tabelle 11.B zeigt die zulässigen Werte für die Halbduplex-Betriebscodes und die entsprechenden Modi. Eine Auflistung der gültigen Werte für die Vollduplex-Betriebscodes und die entsprechenden Modi sind in Tabelle 11.C aufgeführt.

Ein besonderer Betriebscode-Bereich (32 – 43) ist ebenfalls zulässig. Diese Codes entsprechen im wesentlichen den Codes 0 – 11, mit der Ausnahme, daß die EOT-Datenpakete (Ende der Übertragung) unterdrückt werden. Dieser Betrieb stellt eine Abweichung des DF1-Standardprotokolls dar und sollte nur verwendet werden, wenn möglichst wenige Übertragungen von einem Slave-Modul ausgeführt werden sollen. Bei Verwendung einer dieser Optionen reagiert der DF1-Treiber auf die Stationsaufforderung (ENQ) eines DF1-Masters erst dann, wenn ein Datenpaket übertragen wird.

Wichtig: Weitere Portparameter, z.B. Baudrate, Anzahl der Stoppbits und Parität, werden mit Hilfe des MODE-Befehls gewählt, bevor DF1 aktiviert wird. Die hier gewählte Modem-Quittierung hat bis zur Deaktivierung von DF1 eine höhere Priorität gegenüber dem Handshake-Parameter des MODE-Befehls.

Tabelle 11.B
DF1-Halbduplex-Betriebscodes

Betriebs-code	Entsprechender Betriebsmodus	Besonderer Betriebscode (entspricht 0 – 11, mit der Ausnahme, daß EOT unterdrückt wird)
0	NHS, DPD deaktiviert, BCC-Fehlerprüfung	32
1	NHS, DPD aktiviert, BCC-Fehlerprüfung	33
2	NHS, DPD deaktiviert, CRC-Fehlerprüfung	34
3	NHS, DPD aktiviert, CRC-Fehlerprüfung	35
4	HDMwoCC, DPD deaktiviert BCC-Fehlerprüfung	36
5	HDMwoCC, DPD aktiviert, BCC-Fehlerprüfung	37
6	HDMwoCC, DPD deaktiviert, CRC-Fehlerprüfung	38
7	HDMwoCC, DPD aktiviert, CRC-Fehlerprüfung	39
8	HDMwCC, DPD deaktiviert, BCC-Fehlerprüfung	40
9	HDMwCC, DPD aktiviert, BCC-Fehlerprüfung	41
10	HDMwCC, DPD deaktiviert, CRC-Fehlerprüfung	42
11	HDMwCC, DPD aktiviert, CRC-Fehlerprüfung	43

Tabelle 11.C
DF1-Vollduplex-Betriebscodes

Betriebs-code	Entsprechender Betriebsmodus
16	NHS, ER, DPD deaktiviert, BCC-Fehlerprüfung
17	NHS, ER, DPD aktiviert, BCC-Fehlerprüfung
18	NHS, ER, DPD deaktiviert, CRC-Fehlerprüfung
19	NHS, ER, DPD aktiviert, CRC-Fehlerprüfung
20	NHS, ADER, DPD deaktiviert, BCC-Fehlerprüfung
21	NHS, ADER, DPD aktiviert, BCC-Fehlerprüfung
22	NHS, ADER, DPD deaktiviert, CRC-Fehlerprüfung
23	NHS, ADER, DPD aktiviert, CRC-Fehlerprüfung
24	FDM, ER, DPD deaktiviert, BCC-Fehlerprüfung
25	FDM, ER, DPD aktiviert, BCC-Fehlerprüfung
26	FDM, ER, DPD deaktiviert, CRC-Fehlerprüfung
27	FDM, ER, DPD aktiviert, CRC-Fehlerprüfung
28	FDM, ADER, DPD deaktiviert, BCC-Fehlerprüfung
29	FDM, ADER, DPD aktiviert, BCC-Fehlerprüfung
30	FDM, ADER, DPD deaktiviert, CRC-Fehlerprüfung
31	FDM, ADER, DPD aktiviert, CRC-Fehlerprüfung

Halbduplex-Modemsteuerung ohne Handshake

Die durch die Betriebscodes 0 bis 3 gewählte Halbduplex-Modemsteuerung ohne Handshake-Betrieb hat die folgenden Eigenschaften:

- Die RTS-Ausgangsleitung wird während der Übertragung aktiviert, eine RTS-Einschalt- bzw. -Ausschaltverzögerung findet jedoch nicht statt.
- Die DTR-Ausgangsleitung wird nicht vom DF1-Treiber manipuliert. Während der DF1-Kommunikation sollten Sie das DTR-Signal im BASIC-Programm aktivieren.
- Die CTS- und DSR-Eingangsleitungen werden *nicht* überwacht und beeinflussen die Übertragung und den Empfang nicht.
- Durch eine Sendeüberwachung ist gewährleistet, daß Sender-Interrupts rechtzeitig generiert werden. Bei einem Zeitablauf wird DF1_Status auf den Codewert 5 gesetzt, wenn zuvor ein Datenpaket übertragen wurde. Ferner wird die RTS-Verbindung bei einem Zeitablauf sofort unterbrochen.

Die durch die Betriebscodes 4 bis 7 gewählte Halbduplex-Modemsteuerung ohne Dauerträgersignal hat die folgenden Eigenschaften:

Wichtig: Um einen störungsfreien Betrieb zu gewährleisten, muß die Datenträgerbestimmungsleitung (DCD) des Modems an den DSR-Eingang des PRT2-Ports angeschlossen werden.

- Die RTS-Ausgangsleitung wird nur während der Übertragung aktiviert. Die Übertragung des Pakets selbst wird nach der durch den RTS-Einschaltverzögerungsparameter spezifizierten Verzögerung gestartet. Dabei wird vorausgesetzt, daß der CTS-Eingang bereits aktiv ist. Nach Abschluß der Übertragung und Ablauf der durch den RTS-Ausschaltverzögerungsparameter spezifizierten Verzögerung wird RTS deaktiviert.

- Die Übertragung selbst beginnt erst, wenn der CTS-Eingang aktiv ist. Durch die Übertragung ist gewährleistet, daß Sender-Interrupts rechtzeitig generiert werden. Bei einem Zeitablauf wird DF1_Status auf den Codewert 5 gesetzt, wenn zuvor ein Datenpaket übertragen wurde. Ferner wird die RTS-Verbindung bei einem Zeitablauf sofort unterbrochen.

- Wenn die DTR-Verbindung nicht bereits aktiv ist, wird sie bei der Aktivierung des DF1-Treibers in Betrieb genommen. Auch nach der Deaktivierung des DF1-Treibers bleibt diese Verbindung *weiterhin aktiv* und kann mit einem BASIC-Aufruf deaktiviert werden.

- Empfangene Zeichen werden nur bei aktivierter DCD-Verbindung akzeptiert. Wird die DCD-Verbindung während der byteweisen Übertragung des Datenpakets deaktiviert, wird der Empfang des Datenpaketes abgebrochen.

  Im Gegensatz zur Übertragung mit Dauerträgersignal findet auch zwischen Paketen keine kontinuierliche DCD-Überwachung statt. Deshalb wird die DTR-Leitung in keinem Fall deaktiviert.

Halbduplex-Modemsteuerung mit Dauerträgersignal

Die durch die Betriebscodes 8 bis 11 gewählte Halbduplex-Modemsteuerung mit Dauerträgersignal hat die folgenden Eigenschaften:

Wichtig: Um einen störungsfreien Betrieb zu gewährleisten, muß die Datenträgerbestimmungsleitung (DCD) des Modems an den DSR-Eingang des PRT2-Ports angeschlossen werden.

- Die RTS-Ausgangsleitung wird nur während der Übertragung aktiviert. Die Übertragung des Pakets selbst wird nach der durch den RTS-Einschaltverzögerungsparameter spezifizierten Verzögerung gestartet. Dabei wird vorausgesetzt, daß der CTS-Eingang bereits aktiv ist. Nach Abschluß der Übertragung und Ablauf der durch den RTS-Ausschaltverzögerungsparameter spezifizierten Verzögerung wird RTS deaktiviert.

- Die Übertragung selbst beginnt erst, wenn der CTS-Eingang aktiv ist. Durch die Übertragung ist gewährleistet, daß Sender-Interrupts rechtzeitig generiert werden. Bei einem Zeitablauf wird DF1_Status auf den Codewert 5 gesetzt, wenn zuvor ein Datenpaket übertragen wurde. Ferner wird die RTS-Verbindung bei einem Zeitablauf sofort unterbrochen.

- Wenn die DTR-Verbindung nicht bereits aktiv ist, wird sie bei der Aktivierung des DF1-Treibers in Betrieb genommen. Die Verbindung wird nur dann unterbrochen, wenn das DCD-Signal deaktiviert wird (siehe folgenden Abschnitt). Sie können diese Verbindung mit einem BASIC-Aufruf deaktivieren, unabhängig davon, ob der DF1-Treiber deaktiviert oder weiterhin aktiv ist.

- Zum Empfang eines Datenpakets wird das DCD-Signal (über die DSR-Eingangsleitung) überwacht. Wenn das DCD-Signal nicht bereits bei der Aktivierung des DF1-Treibers aktiv ist, wird es sofort bei seiner Aktivierung erkannt. Das DCD-Signal wird nun alle 5 ms überprüft, um sicherzustellen, daß es aktiv bleibt. Bei einer Deaktivierung des DCD-Signals wartet der Treiber 10 Sekunden lang auf eine erneute Aktivierung. Wird das DCD-Signal innerhalb dieses Zeitraums nicht wieder aktiv, wird die DTR-Ausgangsleitung für einen Zeitraum von 5 bis 10 ms unterbrochen.

  Darüber hinaus werden empfangene Zeichen nur bei aktivierter DCD-Leitung akzeptiert. Wird die DCD-Verbindung während der byteweisen Übertragung des Datenpakets deaktiviert, wird der Empfang des Datenpakets abgebrochen.

Vollduplex ohne Handshake

Das durch die Betriebscodes 16 bis 23 gewählte Vollduplexformat ohne Handshake-Betrieb hat die folgenden Eigenschaften:

- Die RTS-Ausgangsleitung wird bei der Aktivierung des DF1-Treibers aktiviert und bleibt bis zur Deaktivierung des DF1-Treibers aktiv.

- Die DTR-Ausgangsleitung wird vom DF1-Treiber nicht manipuliert. Während der DF1-Kommunikation sollten Sie das DTR-Signal im BASIC-Programm aktivieren.

- Die CTS- und DSR-Eingangsleitungen werden *nicht* überwacht und beeinflussen die Übertragung und den Empfang nicht.

- Durch eine Sendeüberwachung ist gewährleistet, daß Sender-Interrupts rechtzeitig generiert werden. Bei einem Zeitablauf wird DF1_Status auf den Codewert 5 gesetzt, wenn zuvor ein Datenpaket übertragen wurde. Das RTS-Signal wird bei einem Zeitablauf *nicht* deaktiviert.

Vollduplex-Modem (FDM)

Das durch die Betriebscodes 24 bis 31 gewählte Vollduplex-Modem (FDM) hat die folgenden Eigenschaften:

- Die RTS-Ausgangsleitung wird bei der Aktivierung des DF1-Treibers aktiviert und bleibt bis zur Deaktivierung des DF1-Treibers aktiv.

- Die Übertragung selbst beginnt erst, wenn der CTS-Eingang aktiv ist. Durch eine Sendeüberwachung ist gewährleistet, daß Sender-Interrupts rechtzeitig generiert werden. Bei einem Zeitablauf wird DF1_Status auf den Codewert 5 gesetzt, wenn zuvor ein Datenpaket übertragen wurde. Das RTS-Signal wird bei einem Zeitablauf *nicht* deaktiviert.

- Wenn die DTR-Verbindung nicht bereits aktiv ist, wird sie bei der Aktivierung des DF1-Treibers in Betrieb genommen. Die Verbindung wird nur dann aktiv, wenn das DCD-Signal deaktiviert wird (siehe folgenden Abschnitt). Das DTR-Signal bleibt auch bei deaktiviertem DF1-Treiber aktiv. Sie können diese Verbindung mit einem BASIC-Aufruf deaktivieren.

- Zum Empfang eines Datenpakets wird das DCD-Signal über die DSR-Eingangsleitung überwacht. Wenn das DCD-Signal nicht bereits bei der Aktivierung des DF1-Treibers aktiv ist, wird es sofort bei seiner Aktivierung erkannt. Das DCD-Signal wird nun alle 5 ms überprüft, um sicherzustellen, daß es aktiv bleibt. Bei einer Deaktivierung des DCD-Signals wartet der Treiber 10 Sekunden lang auf eine erneute Aktivierung. Wird das DCD-Signal innerhalb dieses Zeitraums nicht wieder aktiv, wird die DTR-Ausgangsleitung für einen Zeitraum von 5 bis 10 ms deaktivert.

  Darüber hinaus werden empfangene Zeichen nur bei aktivierter DCD-Leitung akzeptiert. Wird die DCD-Verbindung während der byteweisen Übertragung des Datenpakets deaktiviert, wird der Empfang des Pakets abgebrochen.

Im Halbduplex-Modus spezifiziert das zweite Eingangsargument den Zeitablauf der Abfrage und im Vollduplex-Modus den Zeitablauf der Bestätigung (ACK). Der Zeitablauf der Abfrage spezifiziert in Inkrementen von 5 ms, wie lange der DF1-Master wartet, bevor er eine Abfrage ausführt. Bei Überschreitung der Zeit wird die Sendeanforderung ignoriert. Der PUSH-Wert 0 kennzeichnet einen Abfrage-Zeitablauf von 0 ms. Der Zeitablauf der Bestätigung (ACK) spezifiziert in Inkrementen von 5 ms, wie lange das Programm auf eine Bestätigung (ACK) bzw. auf eine NAK-Meldung wartet, bevor eine Anforderung (ENQ) gesendet wird. Der zulässige Bereich für den Zeitablauf der Bestätigung beträgt 2 bis 65535.

Im Halbduplex-Modus spezifiziert das dritte Eingangsargument die Anzahl der Nachrichtenwiederholungen und im Vollduplex-Modus die Anzahl der auszuführenden Wiederholungen einer Anforderung (ENQ). Die Nachrichtenwiederholung spezifiziert, wie oft das System versucht, eine Nachricht zu senden, bevor es aufgibt und die Übertragung als gescheitert gekennzeichnet wird. Der PUSH-Wert 0 gibt an, daß nur ein Versuch unternommen wird. Wird dieser vom Master nicht bestätigt, gilt der Versuch als gescheitert. Die Wiederholung einer Anforderung (ENQ) spezifiziert die Anzahl der gesendeten ENQ-Signale, bevor die Übertragung eines Datenpakets als gescheitert gilt. Der zulässige Bereich für beide Parameter beträgt 0 bis 254.

Im Halbduplex-Modus spezifiziert das vierte Eingangsargument die RTS-Einschaltverzögerung und im Vollduplex-Modus die Anzahl der empfangenen Wiederholungsversuche bei einer NAK-Meldung. Die RTS-Einschaltverzögerung spezifiziert in Inkrementen von 5 ms die Verzögerung zwischen der Aktivierung einer Sendeanforderung (RTS) und dem Beginn der Übertragung. Dieses Argument wird nur verwendet, wenn im ersten Eingangsargument HDMwCC oder HDMwoCC gewählt wurde. Der gültige Bereich für die RTS-Einschaltverzögerung beträgt 0 bis 65535. Die empfangenen Wiederholungsversuche bei einer NAK-Meldung spezifizieren die Anzahl der Paketübertragungswiederholungen, die aufgrund der NAK-Meldung durchgeführt werden. Der gültige Bereich für Wiederholungsversuche bei Empfang einer NAK-Meldung liegt zwischen 0 und 254.

Das fünfte Eingangsargument bestimmt die RTS-Ausschalt-verzögerung. Diese spezifiziert in Inkrementen von 5 ms die Verzögerung zwischen dem Ende einer Übertragung und der Deaktivierung der Sendeanforderung (RTS). Der gültige Bereich für die RTS-Ausschaltverzögerung liegt zwischen 0 und 65499. Dieses Argument wird nur verwendet, wenn im ersten Eingangsargument HDMwCC oder HDMwoCC gewählt wurde. Ferner wird dieses Eingangsargument nur im Halbduplex-Modus verwendet. Im Vollduplex-Modus muß mit der PUSH-Anweisung der Wert NULL eingegeben werden.

Das sechste Eingangsargument spezifiziert die Adresse des BASIC-Moduls, auf die der DF1-Treiber beim Empfang der Aufforderungen eines dezentralen DF1-Gerätes reagiert. Die gültigen Werte liegen zwischen 0 und 254. Dieses Eingangsargument wird im Halbduplex- und Vollduplex-Modus verwendet.

Syntax:

PUSH [Betriebscode]
PUSH [Abfrage-Zeitablauf oder ACK-Zeitablauf]
PUSH [Nachrichten- oder ENQ-Übertragungswiederholungen]
PUSH [RTS-Einschaltverzögerung oder Wiederholungsversuche bei Empfang einer NAK-Meldung]
PUSH [RTS-Ausschaltverzögerung oder NULL-Wert]
PUSH [DF1-Adresse des BASIC-Moduls]
CALL 108

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 5 : REM HDMWOCC, ENABLE DPD, BCC ERROR CHECKING
>20 PUSH 200 : REM WAIT 1 SECOND TO BE POLLED BY MASTER
>30 PUSH 2 : REM PERFORM 2 RETRIES
>40 PUSH 4 : REM 20 MS RTS ON DELAY
>50 PUSH 4 : REM 20 MS RTS OFF DELAY
>60 PUSH 10 : REM BASIC MODULE ADDRESS OF 10
>70 CALL 108
>80 END
```

CALL 113 – Deaktivierung der DF1-Treiberkommunikation

Funktion:

Mit CALL 113 wird die DF1-Treiberkommunikation deaktiviert. Diese Routine besitzt keine Eingangs- und Ausgangsargumente. Mit diesem Aufruf wird die DF1-Kommunikation auch im Falle einer seriellen Übertragung eines Datenpakets sofort beendet. Das Anwenderprogramm sollte so geschrieben werden, daß alle Übertragungen vor der Ausführung von CALL 113 beendet werden. Mit diesem Aufruf werden der Sende- und Empfangspuffer von PRT2 gelöscht.

Syntax:

CALL 113

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 113
>20 END
```

CALL 120 – Löschen der Eingangs- und Ausgangspuffer des BASIC-Moduls

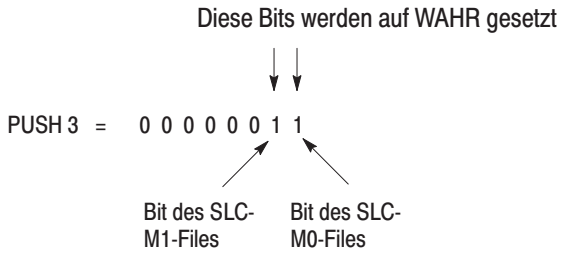
Funktion:

Mit CALL 120 wird der Eingangs- und Ausgangspuffer des BASIC-Moduls gelöscht. Diese Routine verfügt über ein Eingangs- und kein Ausgangsargument. Das Eingangsargument ist ein 8-Bit-Wort, das jeweils dem Eingangs- und Ausgangspuffer des BASIC-Moduls entspricht (siehe folgende Tabelle):

Tabelle 11.D
Informationen zum Löschen der Eingangs- und Ausgangspuffer

Bit	Entsprechender Dezimalwert	Eingangs- und Ausgangspufferbereiche des BASIC-Moduls
0	1	SLC-M0- File
1	2	SLC-M1-File
2	4	SLC-Ausgangsdatentafel
3	8	SLC-Eingangsdatentafel
4	16	gemeinsamer Schnittstellen-Eingangsfile
5	32	gemeinsamer Schnittstellen-Ausgangsfile
6		nicht belegt
7		nicht belegt

Sie müssen den entsprechenden Dezimalwert der zu löschenden Eingangs- und Ausgangspufferbereiche mit der PUSH-Anweisung festlegen. Um beispielsweise die SLC-M0- und SLC-M1-Files zu löschen, müssen Sie in der PUSH-Anweisung den Wert 3 spezifizieren. Das BASIC-Modul setzt dann die Bits 0 und 1 auf wahr und löscht die SLC-M0- und SLC-M1-Filebereiche des Eingangs- und des Ausgangspuffers.



Syntax:

PUSH [entsprechender Dezimalwert]
CALL 120

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 3 : REM CLEAR SLC MO AND M1 FILES
>20 CALL 120
>30 END
```

CALL 121 – Abruf der Programmkennummer des SLC-Prozessors

Funktion:

Mit CALL 121 wird die Kennummer des aktiven Programms im SLC-Prozessor abgerufen. Diese Routine verfügt über kein Eingangs- und ein Ausgangsargument. Das Ausgangsargument ist ein Ganzzahlwert zwischen 0 und 65536. Dieser Wert entspricht der Kennummer des aktiven Programms des SLC-Prozessors.

Syntax:

CALL 121
POP [Programmkennummer]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 121
>20 POP X : REM GET SLC PROCESSOR PROGRAM ID
>30 END
```

Kapitel 12

Ausgangsfunktionen

Dieses Kapitel enthält eine Beschreibung und Darstellung der Befehle, die eine Übertragung der im BASIC-Modul enthaltenen Daten an die externen Ports PRT1, PRT2 und DH485 ermöglichen. Diese Befehle können im BASIC-Programm oder von der Befehlszeile aus ausgeführt werden. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 12.A enthalten.

Tabelle 12.A
Kapitelinhalt

Funktion	Mnemonik	Seite
Ausgabe eines Hexadezimalwertes mit Nullpunktunterdrückung an PRT2	PH0.#	12-42
Ausgabe eines Hexadezimalwertes mit Nullpunktunterdrückung an PRT1	PH0.@	12-42
Ausgabe eines Hexadezimalwertes ohne Nullpunktunterdrückung an das Terminal	PH1.	12-42
Ausdruck eines Hexadezimalwertes ohne Nullpunktunterdrückung an PRT2	PH1.#	12-42
Ausgabe eines Hexadezimalwertes ohne Nullpunktunterdrückung an PRT1	PH1.@	12-42
Ausgabe von Variablen, Zeichenketten bzw. Literalen an das Terminal. P. ist die Abkürzung für “PRINT”	PRINT	12-39
Ausgabe an den PRT2-Port	PRINT#	12-40
Ausgabe an den PRT1-Port	PRINT@	12-40
Ausgabe eines Wagenrücklaufs	PRINT CR	12-40
Ausgabe von Leerstellen	PRINT SPC( )	12-41
Ausgabe von Tabulatorzeichen	PRINT TAB( )	12-41
Ausgabe numerischer Werte in wissenschaftlicher Schreibweise	PRINT USING(Fx)	12-41
Ausgabe numerischer Werte in dezimaler Schreibweise	PRINT USING(#.#)	12-41
Wiederherstellung des vorgegebenen Ausgabemodus	PRINT USING(0)	12-42
Speichern einer Variablen	ST@	12-43

CALL 23 – Datenübertragung von den CPU-Files an Port 1 oder 2

Funktion:

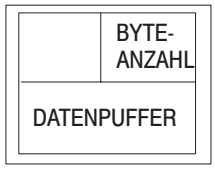
Mit CALL 23 werden Daten des CPU-Ausgangsdatenfiles bzw. des CPU-M0-Files direkt an den seriellen Port und/oder an eine Zeichenkette eines BASIC-Moduls übertragen. Zuerst wird das niederwertige Byte und anschließend das hochwertige Byte bzw. zuerst das hochwertige Byte und anschließend das niederwertige Byte an den Port des BASIC-Moduls übertragen. Darüber hinaus können die Daten zum Abruf durch das BASIC-Programm in einer Zeichenkette gespeichert werden.

Das Datenpackformat für alle durch CALL 23 aktivierten Ports wird durch die Byteübertragungsfolge (zuerst das niederwertige und dann das hochwertige Byte oder zuerst das hochwertige und anschließend das niederwertige Byte) des zuletzt ausgeführten Aufrufs 23 bzw. 22 festgelegt.

Das niederwertige Byte des ersten Wortes im Quellfile enthält die Zeichenanzahl (Byteanzahl) der übertragenen Daten. Wenn diese Byteanzahl größer als der gewählte File ist, wird nur die maximale Byteanzahl des Files übertragen. Das hochwertige Byte des ersten Wortes wird nicht belegt.

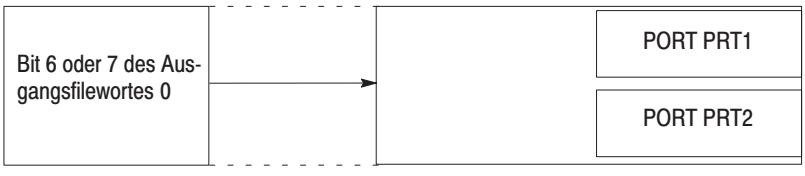
Führen Sie CALL 23 aus, um die Datentransferparameter zu konfigurieren. Nach der Ausführung dieses Aufrufs erhält das BASIC-Modul Daten aus dem Quellfile und überträgt sie an PRT1, PRT2 oder an eine interne Zeichenkette. Mit den Bits des Eingangs- und des Ausgangsabbildfiles (Wort 0, Bits 6 und 7) des Steckplatzes, in dem sich das BASIC-Modul befindet, wird die Übertragung eingeleitet und die Beendigung der Übertragung signalisiert. Dieser Vorgang ist im folgenden näher beschrieben:

1. Der Datenpuffer wird im Strompfadlogikprogramm des SLC-Prozessors erstellt. Der SLC-Prozessor bestimmt anschließend die Byteanzahl des zu übertragenden Files und speichert diesen Wert im niederwertigen Byte des ersten zu übertragenden Wortes. Der zu übertragende Datenfile umfaßt dieses Wort sowie die Daten.



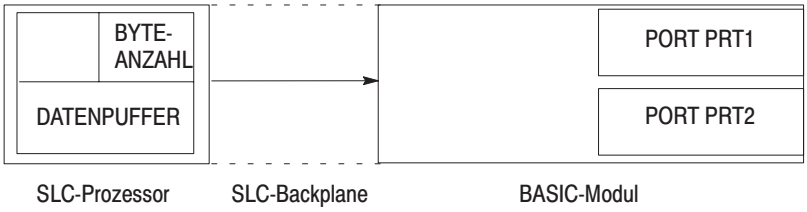
SLC-Prozessor

2. Das Strompfadlogikprogramm des SLC-Prozessors muß Bit 6 bzw. Bit 7 des Ausgangsfilewortes 0 setzen, um dem BASIC-Modul zu signalisieren, daß gültige Daten verfügbar sind. Bit 6 signalisiert, daß die Daten von PRT1 abgerufen werden können, und Bit 7 weist darauf hin, daß die Daten von PRT2 abgerufen werden können.

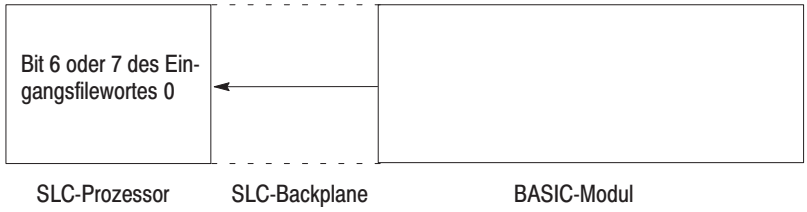


SLC-Prozessor SLC-Backplane BASIC-Modul

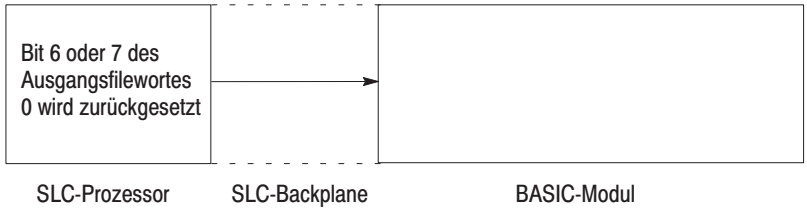
3. Das BASIC-Modul überträgt die Daten vom SLC-Prozessor automatisch an den spezifizierten Zielport (PRT1 oder PRT2).



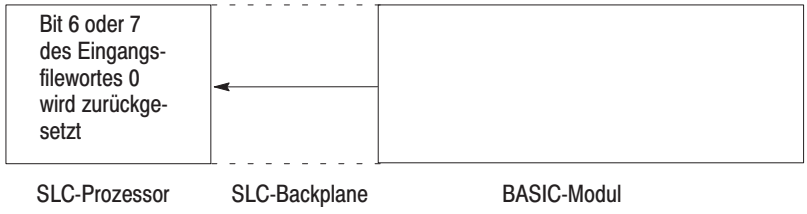
4. Das BASIC-Modul setzt Bit 6 bzw. Bit 7 des Eingangsfilewortes 0, um dem SLC-Prozessor mitzuteilen, daß die Übertragung erfolgreich durchgeführt wurde.



5. Das Strompfadlogikprogramm des SLC-Prozessors setzt Bit 6 bzw. Bit 7 des Ausgangsfilewortes 0 zurück.



6. Das BASIC-Modul setzt Bit 6 bzw. Bit 7 des Eingangsfilewortes 0 an demselben Ende des Abfragezyklus zurück, an dem Bit 6 bzw. Bit 7 des Ausgangsfilewortes 0 zurückgesetzt wurde.



Die Datenübertragung wird auf diese Weise fortgesetzt, bis der Aufruf für den Port mit anderen Eingangsparametern ausgeführt wird. In diesem Fall wird der vorherige CALL 23 dieses Ports automatisch deaktiviert, und der neue CALL 23 wird wirksam. Mehrfache Ausführungen von CALL 23 für denselben Port erfolgen nicht gleichzeitig. Port 1 und Port 2 können jedoch gleichzeitig aktiviert werden, indem CALL 23 für jeden Port separat erteilt wird.

Dieser Aufruf verfügt über fünf Eingangsargumente und ein Ausgangsargument.

Mit dem ersten Eingangsargument wird das Datenziel gewählt. Dies kann die Portnummer (1 oder 2) und/oder die interne Zeichenkette sein:

- 0 = Deaktivierung von CALL 23 für alle aktiven Ports und Zeichenketten, die durch vorhergehende Ausführungen von CALL 23 aktiviert wurden
- 1 = nur interne Zeichenkette
- 2 = serieller Port 1
- 3 = interne Zeichenkette und serieller Port 1
- 4 = serieller Port 2
- 5 = interne Zeichenkette und serieller Port 2

Wenn eine interne Zeichenkette (1, 3 oder 5) gewählt wird, enthält das erste Zeichen der Zeichenkette den Bytezählwert. Das zweite Zeichen (Übertragungsnummer) wird inkrementiert, um dem BASIC-Modul zu signalisieren, daß neue Daten in der Zeichenkette enthalten sind. Der Wert dieses Zeichens geht nach 255 wieder auf 0 über. Die Daten des Quellpuffers beginnen ab dem dritten Zeichen der Zeichenkette.

Das zweite Eingangsargument ist die Wahl der Datenquelle. Diese kann der CPU-Ausgangsdatenfile oder der CPU-M0-File sein:

- 0 = CPU-Ausgangsabbildfile
- 1 = CPU-M0-File

Das dritte Eingangsargument ist der Wortversatz im CPU-Ausgangsabbildfile bzw. M0-File. Dieser Versatz zeigt auf das Wort, das den Bytewert der gültigen Daten im File enthält. Das zweite Wort des CPU-Files enthält die ersten zwei Zeichen der zu übertragenden Daten. Wenn der CPU-Ausgangsabbildfile gewählt wird, darf dieser Versatz nicht Wort 0 sein, da dieses Wort für die mit diesem Aufruf verwendeten Handshake-Bits reserviert ist. Bei Wahl des Ausgangsabbildes ist deshalb das erste für den Bytezählwert verfügbare Wort das zweite Wort (Wort 1).

Das vierte Eingangsargument ist die Nummer der internen Zeichenkette. Wenn mit dem zweiten Eingangsargument nicht die interne Zeichenkette gewählt wird, wird der Wert dieses Eingangsargumentes ignoriert (muß jedoch trotzdem in der PUSH-Anweisung enthalten sein). Wenn die Daten die Zeichenkettenlänge überschreiten, werden die restlichen Daten abgeschnitten.

Mit dem fünften Eingangsargument wird die Byteübertragung gewählt, wobei die folgenden Werte Gültigkeit haben:

- 0 = die von der CPU-Einheit übertragenen Datenbytes werden bei der Übertragung an den Port bzw. an die Zeichenkette des BASIC-Moduls *nicht* vertauscht. Je Wort wird zuerst das niederwertige und anschließend das hochwertige Byte übertragen. Das niederwertige Byte des ersten Wortes im Quellpuffer enthält den Bytezählwert.

- 1 = die von der CPU-Einheit übertragenen Datenbytes werden bei der Übertragung an den Port bzw. an die Zeichenkette des BASIC-Moduls ausgetauscht. Je Wort wird zuerst das hochwertige und anschließend das niederwertige Byte übertragen. Das erste Wort ist nicht vom Austauschvorgang betroffen. Das niederwertige Byte des ersten Wortes enthält weiterhin den Bytezählwert.

Der zuletzt ausgeführte CALL 23 bestimmt die Byteaustauschoption für alle zuvor ausgeführten aktiven CALL-23- und CALL-22-Befehle.

Das Ausgangsargument ist der Status des Aufrufs, wobei die folgenden Werte Gültigkeit haben:

- 0 = erfolgreich

- 1 = deaktiviert

- 2 = ungültiger Eingangsparameter

- 3 = PRT2 wurde gewählt, ist jedoch bereits für das DF1-Protokoll aktiviert. Der Aufruf wird nicht ausgeführt.

- 4 = Zeichenkette ist zu klein

- 5 = Zeichenkette ist nicht dimensioniert

Der Datentransfer beginnt erst, wenn der SLC-Prozessor (je nach Zielfile) Bit 6 oder Bit 7 in Wort 0 des Ausgangsabbildfiles setzt. Das BASIC-Modul setzt Bit 6 oder Bit 7 in Wort 0 des Eingangsfiles, um eine erfolgreiche Übertragung an den Port zu signalisieren.

Wenn als Ziel die interne Zeichenkette gewählt wurde, wird die Übertragung mit Bit 6 in Wort 0 des Ausgangsabbildfiles eingeleitet. Das BASIC-Modul setzt Bit 6 in Wort 0 des Eingangsfiles, um die erfolgreiche Übertragung an die Zeichenkette zu signalisieren.

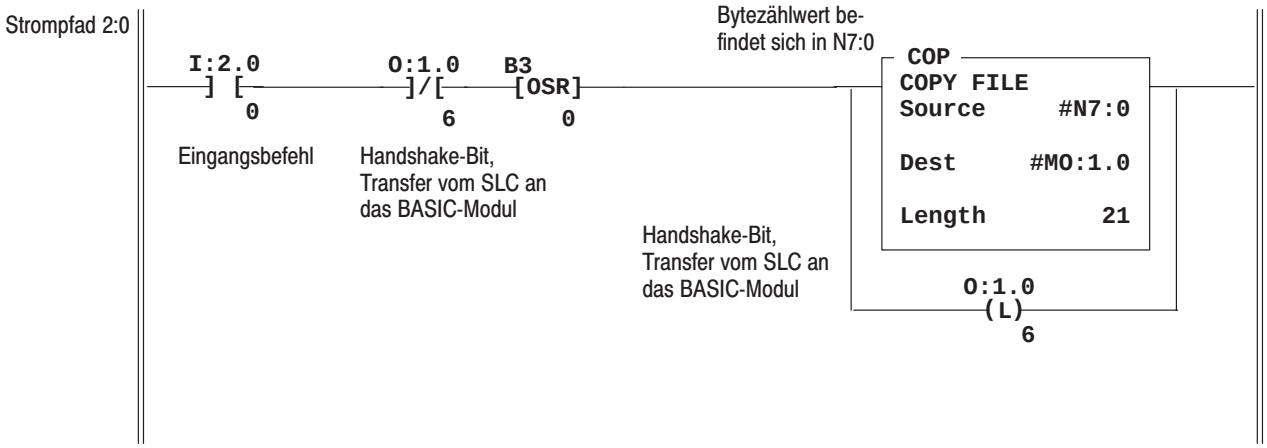
Syntax:

PUSH [Nummer des Zielports und/oder interne Zeichenkette]
PUSH [Wahl des Quellfiles]
PUSH [Wortversatz im Quellfile]
PUSH [Nummer der Zeichenkette]
PUSH [Wahl des Byteaustausches]
CALL 23
POP [Status von CALL 23]

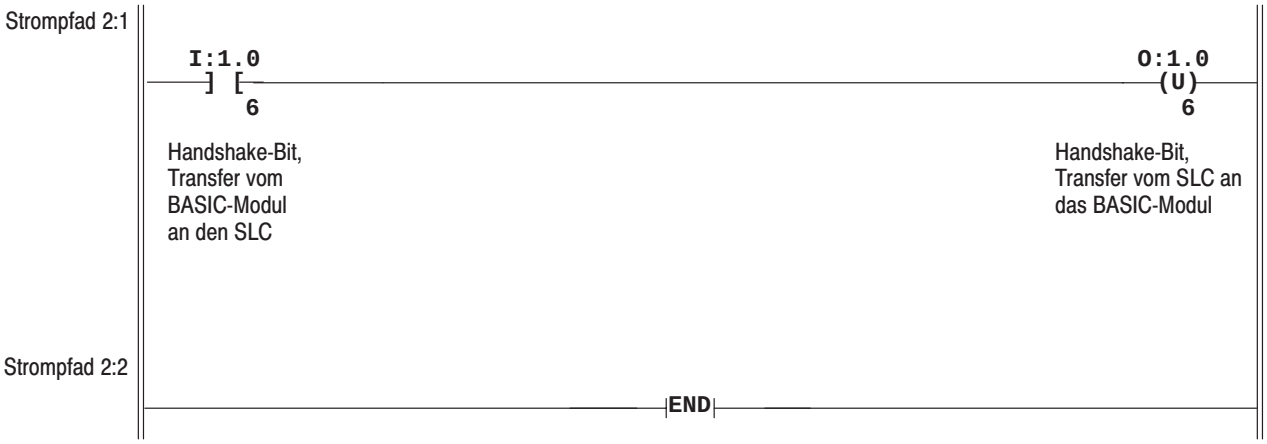
Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE CALL 23 INTERRUPTS
>20 PUSH 2 : REM SEND DATA TO PRT1
>30 PUSH 1 : REM GET DATA FROM M0 FILE
>40 PUSH 0 : REM WORD OFFSET INTO M0 FILE
>50 PUSH 0 : REM STRING NUMBER/NOT USED HERE
>60 PUSH 1 : REM ENABLE BYTE SWAPPING
>70 CALL 23
>80 POP S : REM STATUS OF CALL SETUP
>90 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 23 SETUP”
```

Im folgenden ist ein Strompfadlogikprogramm für CALL 23 abgebildet. Das BASIC-Modul befindet sich in Steckplatz 1 des SLC-Chassis. In Strompfad 2:0 werden Daten in den M0-File kopiert und das Handshake-Bit (O:1.0/6) gesetzt, um dem BASIC-Modul zu signalisieren, daß neue Daten für PRT1 verfügbar sind. Weitere Daten dürfen erst gesendet werden, wenn die zuvor enthaltenen Daten vom BASIC-Modul empfangen und bestätigt wurden. Der Datenfile für das BASIC-Modul beginnt mit dem Bytezählwert (in diesem Beispiel 40) an der Adresse N7:0. Die Daten selbst sind in N7:1–N7:20 enthalten. Das Wort, das den Datenbytezählwert enthält, ist nicht im Datenbytezählwert enthalten. Die Wortlänge des Kopierbefehls beträgt 21. In diesem Beispiel beträgt der maximale Bytezählwert 40 Bytes (20 Worte).



In Strompfad 2:1 wird das Handshake-Bit des SLCs (O:1.0/6) für Port 1 entriegelt, wenn das Handshake-Bit des BASIC-Moduls gesetzt wird, um zu signalisieren, daß das Modul die Daten empfangen hat.



CALL 28 – Schreibtransfer vom SLC-Prozessor an einen dezentralen DH-485-Datenfile

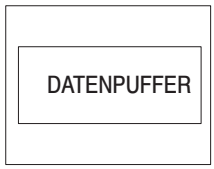
Funktion:

Mit CALL 28 werden bis zu 40 Datenworte vom CPU-Ausgangsabbildfile, CPU-M0-File und/oder einer Zeichenkette im BASIC-Modul an den dezentralen DH-485-File der spezifizierten Netzknotenadresse geschrieben.

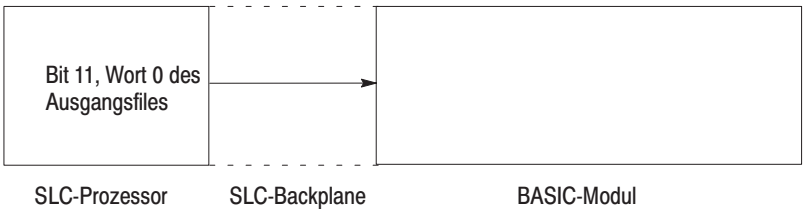
Bei Wahl einer internen Zeichenkette wird das erste Zeichen (die Transaktionsnummer) bei einem erfolgreichen Abschluß des Schreibtransfers inkrementiert, um dem BASIC-Modul zu signalisieren, daß die Datenübertragung stattgefunden hat. Der Wert der Transaktionsnummer geht nach 255 wieder auf 0 über.

Führen Sie CALL 28 einmal aus, um die Datenübertragungsparameter zu spezifizieren. Die Transfereinleitung und Transferabschlußmeldung wird mit Eingangs- und Ausgangsabbildbits (Wort 0, Bit 11) für den Steckplatz des BASIC-Moduls verwendet. Dieses Verfahren ist im folgenden beschrieben:

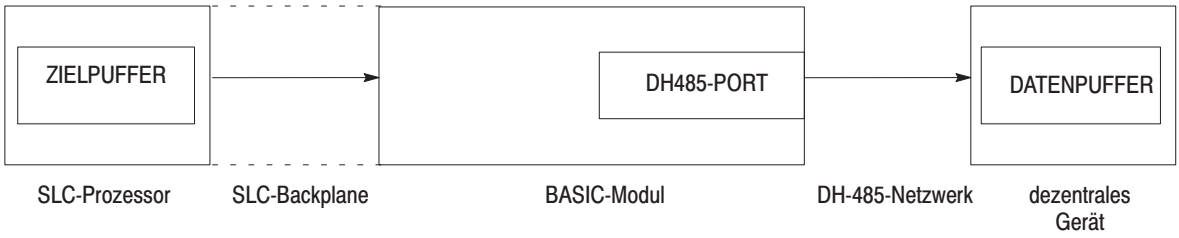
1. Der zentrale SLC-Prozessor erstellt den Datenpuffer.



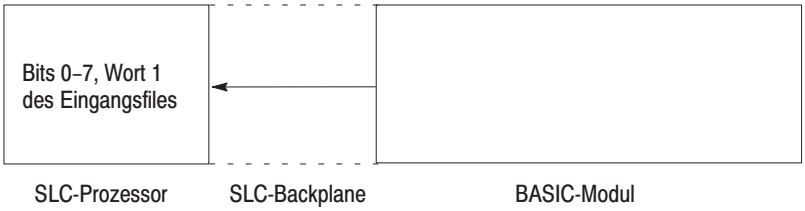
2. Der SLC-Prozessor setzt Bit 11 des Wortes 0 im Ausgangsfile, um dem BASIC-Modul zu signalisieren, daß Daten übertragen werden können.



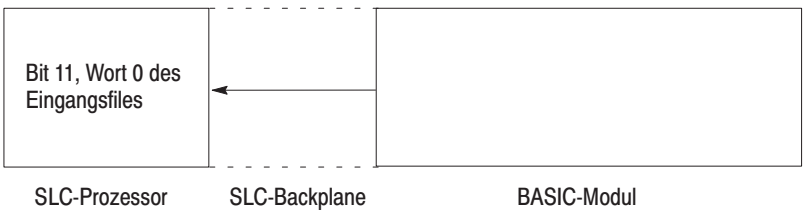
3. Das BASIC-Modul überträgt die Daten in den dezentralen Datenpuffer.



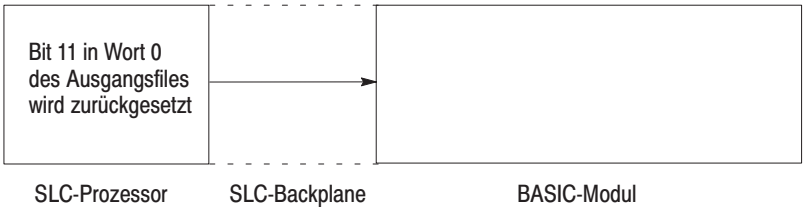
4. Das BASIC-Modul überträgt den Transferstatus in die Bits 0–7 des Wortes 1 im Eingangsfile.



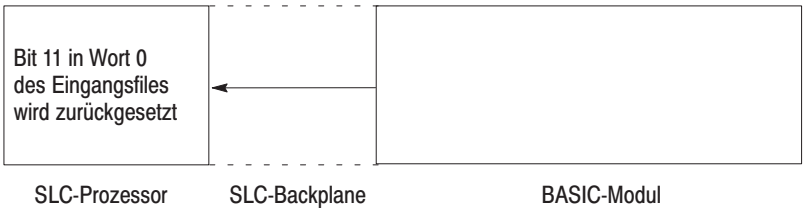
5. Das BASIC-Modul setzt Bit 11 in Wort 0 des Eingangsfiles, um dem zentralen SLC-Prozessor zu signalisieren, daß gültige Daten übertragen wurden und daß der Transferstatus verfügbar ist.



6. Der SLC-Prozessor ruft den Status ab und setzt Bit 11 in Wort 0 des Ausgangsfiles zurück. Der Datenpuffer kann nun durch den SLC-Prozessor neu belegt werden.



7. Das BASIC-Modul setzt Bit 11 in Wort 0 des Eingangsfiles an demselben Ende des Abfragezyklus zurück, an dem Bit 11 in Wort 0 des Ausgangsfiles zurückgesetzt wurde.



Nachdem dieser Aufruf aktiviert wurde, bleibt er aktiv und sendet Daten an den dezentralen Netzknoten, wenn der SLC-Prozessor Bit 11 in Wort 0 des Ausgangsfiles setzt.

Dieser Aufruf besitzt zehn Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist der Typ des erteilten SLC-WRITE-Befehls:

- 0 = Deaktivierung des zuvor ausgeführten CALLs 28
- 1 = gemeinsamer Schnittstellenfile
- 2 = SLC-Typ

Das zweite Eingangsargument ist die Netzknotenadresse des dezentralen SLC-Gerätes im DH-485-Netzwerk (0 bis 31). Liegt die Nummer nicht innerhalb dieses Bereichs, wird als Status der Wert 2 ausgegeben, und der Schreibtransfer findet nicht statt.

Das dritte Eingangsargument ist die Filenummer des dezentralen SLC-Gerätes (0 bis 255). Liegt die Nummer nicht innerhalb dieses Bereichs, wird als Status der Wert 2 ausgegeben, und der Schreibtransfer findet nicht statt. Der Parameter wird ignoriert, wenn im ersten Argument der gemeinsame Schnittstellenfile (CIF) gewählt wurde. Dieser ist immer 9.

Das vierte Eingangsargument ist der Filetyp, der an das dezentrale Gerät geschrieben werden soll. Diese Nummer wird ignoriert, wenn im ersten Argument der gemeinsame Schnittstellenfile gewählt wurde. (d.h. Ganzzahlfile wird vorausgesetzt). Wenn der Filetyp nicht in der folgenden Auflistung enthalten ist, wird als Status der Wert 2 ausgegeben, und der Schreibtransfer findet nicht statt. Geben Sie den jeweils zutreffenden Filecode ein:

Tabelle 12.B
Filetyp

Filetyp	Filetyp-Code	Worte/Element
Ganzzahlfile	ASC(N)	1 Wort/Element
Zählerfile	ASC(C)	3 Worte/Element
Zeitwerkfile	ASC(T)	3 Worte/Element
Bitfile	ASC(B)	1 Wort/Element
Steuerfile	ASC(R)	3 Worte/Element

Das fünfte Eingangsargument ist der beginnende Wortversatz im File des dezentralen SLC-Gerätes (0 bis 32766). Liegt der Wert nicht innerhalb dieses Bereichs, wird als Status der Wert 2 ausgegeben, und der Schreibtransfer findet nicht statt.

Das sechste Eingangsargument ist die Anzahl der zu übertragenden Worte. Liegt die Zahl nicht innerhalb des unten aufgeführten Bereichs, wird als Status der Wert 2 ausgegeben, und der Transfer findet nicht statt.

Tabelle 12.C
Bereich der gültigen Längen

Filetyp-Code	Bereich der gültigen Längen
ASC(N)	1 bis 40
ASC(C)	1 bis 13
ASC(T)	1 bis 13
ASC(B)	1 bis 40
ASC(R)	1 bis 13
gemeinsamer Schnittstellenfile	1 bis 40

Das siebte Eingangsargument ist der Zeitablaufwert der Nachricht. Dieser Wert (1 bis 255) ist die Zeit (x 100 ms), die für den Empfang der Schreibantwort (0,1 bis 25,5 Sekunden) zulässig sind. Die Nachrichtenübertragung wird abgebrochen, wenn die Schreibantwort nicht innerhalb dieses Zeitraums empfangen wird, und als Status wird in die Bits 0–7 in
Wort 1 des Eingangsfiles der Wert 55 geschrieben. Liegt der Zeitablaufwert nicht im gültigen Bereich (1 bis 255), entspricht der mit der POP-Anweisung übertragene Status dem Wert 2, und die Übertragung findet nicht statt.

Das achte Eingangsargument ist die Wahl des CPU-Ausgangs-abbildfiles, des CPU-M0-Files oder der internen Zeichenkette als Quelle:

- 0 = CPU-Ausgangsabbildfile
- 1 = CPU-M0-File
- 2 = interne Zeichenkette

Wenn als Quelle die interne Zeichenkette (2) gewählt wird, kann CALL 29 ausgeführt werden, um die einzelnen Datenübertragungen einzuleiten, ohne daß ein Dialog mit dem SLC-Prozessor erforderlich ist. Mit Bit 11 in Wort 0 des Ausgangsfiles wird ebenfalls eine Zeichenkettenübertragung veranlaßt.

Das neunte Eingangsargument ist der Wortversatz im CPU-Quellfile. Dieser verweist auf die Adresse der ersten Daten. Wenn Sie den CPU-Ausgangsabbildfile wählen, sollte der Versatz nicht 0 sein, da dieses Wort für die Handshake-Bits des Datentransfers reserviert ist. Der Versatz für die interne Zeichenkette ist immer 1. Das Zeichenkettenzeichen (Transaktionsnummer) in Adresse 0 wird nach jedem Datentransfer inkrementiert.

Das zehnte Eingangsargument ist die Zeichenkettennummer. Wenn mit dem achten Eingangsargument nicht die interne Zeichenkette gewählt wurde, wird der Wert dieses Eingangsarguments ignoriert, muß aber trotzdem in der PUSH-Anweisung enthalten sein.

Das Ausgangsargument ist die Validierung der Aufrufparameter. Diese kann folgende Werte haben:

- 0 = erfolgreich
- 1 = deaktiviert
- 2 = ungültiger Eingangsparameter
- 3 = DH485-Port nicht aktiviert (DF1 aktiviert)
- 4 = Zeichenkette ist zu klein
- 5 = Zeichenkette ist nicht dimensioniert
- 6 = Datenpaket ist zu lang

Zur Deaktivierung dieses Aufrufs muß in die PUSH-Anweisung eine Null eingegeben werden. Alle weiteren Parameter werden ignoriert, müssen aber trotzdem in der PUSH-Anweisung enthalten sein.

Wenn versucht wird, einen Schreibtransfer an einen dezentralen Netzknoten durchzuführen, legt das BASIC-Modul den Status des Schreibtransfers in Bits 0–7 in Wort 1 des SLC-Eingangsfiles ab. Die Definition der Statuswerte entspricht den Werten, die in CALL 93 in der POP-Anweisung programmiert wurden.

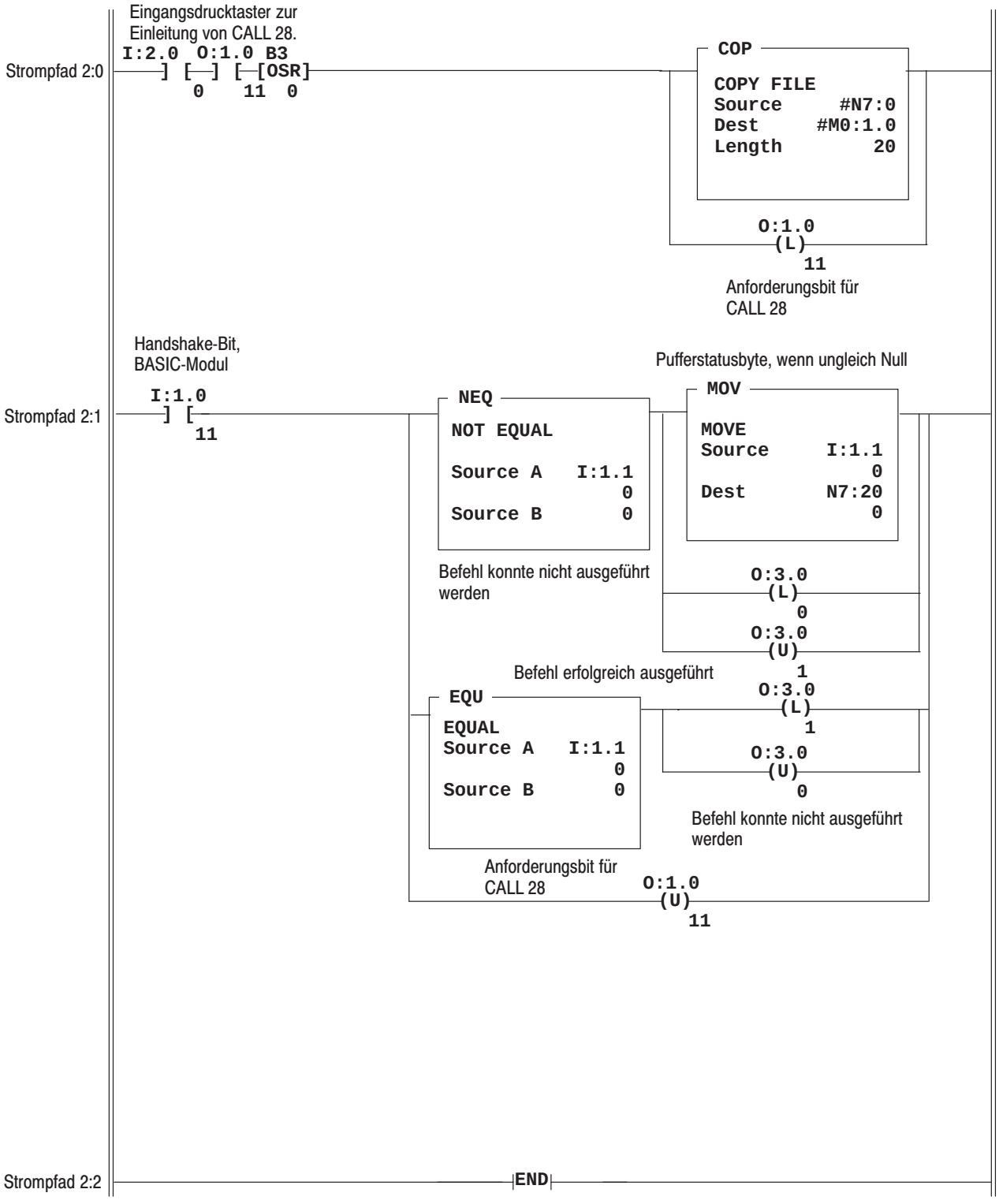
Syntax:

PUSH [Typ des WRITE-Befehls]
PUSH [dezentrale Netzknotenadresse]
PUSH [dezentrale Filenummer]
PUSH [dezentraler Filetyp]
PUSH [Versatz des beginnenden Wortes an der dezentralen Adresse]
PUSH [Anzahl der zu übertragenden Worte]
PUSH [Zeitablaufwert]
PUSH [Wahl des Quellfiles]
PUSH [Wortversatz im Quellfile]
PUSH [Zeichenkettennummer]
CALL 28
POP [Status von CALL 28]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE INTERRUPTS FOR READ COMMANDS TO A
     REMOTE DH-485 NODE
>20  PUSH 2 : REM SLC TYPED WRITE
>30  PUSH 1 : REM REMOTE SLC NODE NUMBER
>40  PUSH 7 : REM REMOTE SLC FILE NUMBER
>50  PUSH ASC(N) : REM REMOTE SLC FILE TYPE
>60  PUSH 0 : REM ELEMENT OFFSET INTO DESTINATION FILE
>70  PUSH 20 : REM NUMBER OF ELEMENTS TO BE TRANSFERRED
>80  PUSH 10 : REM MESSAGE TIME-OUT VALUE(X100MS)
>90  PUSH 1 : REM LOCAL SLC FILE TYPE (M0)
>100 PUSH 0 : REM WORD OFFSET INTO LOCAL SLC FILE
>110 PUSH 0 : REM INTERNAL STRING NUMBER - NOT USED FOR THIS
     EXAMPLE
>120 CALL 28
>130 POP S
>140 IF (S=1) THEN PRINT "CALL 28 DISABLED"
>150 IF (S=2) THEN PRINT "CALL 28 BAD INPUT PARAMETER"
>160 IF (S=3) THEN PRINT "PORT DH485 NOT ENABLED"
```

Kopieren Sie die Daten, die an den dezentralen DH-485-Netzknoten gesendet werden sollen, in den zentralen File (M0-File, der sich auf die Steckplatznummer des BASIC-Moduls bezieht). Verriegeln Sie das Anforderungsbit für CALL 28 (Wort 0, Bit 11).


Strompfad 2:0
Eingangsdrucktaster zur Einleitung von CALL 28.
I:2.0 O:1.0 B3
0 11 0
OSR
COP
COPY FILE
Source #N7:0
Dest #M0:1.0
Length 20
O:1.0
(L)
11
Anforderungsbit für CALL 28
Strompfad 2:1
Handshake-Bit, BASIC-Modul
I:1.0
11
NEQ
NOT EQUAL
Source A I:1.1
0
Source B 0
Befehl konnte nicht ausgeführt werden
Pufferstatusbyte, wenn ungleich Null
MOV
MOVE
Source I:1.1
0
Dest N7:20
0
O:3.0
(L)
0
O:3.0
(U)
1
Befehl erfolgreich ausgeführt
EQU
EQUAL
Source A I:1.1
0
Source B 0
O:3.0
(L)
1
O:3.0
(U)
0
Befehl konnte nicht ausgeführt werden
Anforderungsbit für CALL 28
O:1.0
(U)
11
Strompfad 2:2
END

CALL 29 – Lese-/Schreibtransfer an PLC/SLC von der internen Zeichenkette des BASIC-Moduls aus

Funktion:

In Verbindung mit CALL 122 bzw. 123 wird mit CALL 29 die Kommunikation zwischen dezentralen PLC-Prozessoren und der internen Zeichenkette des BASIC-Moduls ermöglicht, ohne daß ein Dialog über den zentralen SLC-Prozessor erforderlich ist. Ferner kann CALL 29 mit CALL 27 bzw. 28 verwendet werden, um die Kommunikation zwischen dezentralen SLC-Prozessoren und der internen Zeichenkette des BASIC-Moduls zu ermöglichen, ohne daß ein Dialog über den zentralen SLC-Prozessor erforderlich ist. Die Aufrufe 27, 28, 122 bzw. 123 müssen vor CALL 29 im BASIC-Programm ausgeführt werden.

CALL 29 wird aktiviert, wenn die interne Zeichenkette in den Aufrufen 27, 28, 122 bzw. 123 die einzige Wahl ist. In diesem Fall wäre es nicht sinnvoll, zur Einleitung des Datentransfers und zur Übertragung der Statusdaten die SLC-Eingangs- und -Ausgangsabbildfiles zu verwenden. Der SLC-Prozessor ist zur Ausführung dieses Vorgangs nicht erforderlich. Wenn stattdessen ein SLC-File gewählt wird, steuert der zentrale SLC-Prozessor den Datentransfer mit den Eingangs- und Ausgangsabbildbits. Wenn in diesem Fall versucht wird, CALL 29 auszuführen, wird als Status der Wert 255 angezeigt.

Ein Argument wird in eine PUSH-Anweisung und ein anderes in eine POP-Anweisung eingegeben. Das Eingangsargument ist der zu aktivierende Aufruf (CALL 27, 28, 122 bzw. 123).

Bei der Ausführung von CALL 29 wird ein Übertragungsversuch durchgeführt. Wenn der gewählte Aufruf (CALL 27, 28, 122 bzw. 123) nicht vor CALL 29 ausgeführt wird, enthält der mit der POP-Anweisung ausgegebene Status den Wert 1. Nachdem CALL 29 erfolgreich ausgeführt wurde, wird der Wert des ersten Zeichens der Zeichenkette (Transaktionsnummer) inkrementiert, um zu signalisieren, daß die Übertragung stattgefunden hat. Der Bereich dieses Zeichens liegt zwischen 0 und 255.

Nach der Ausführung von CALL 29 wird in der POP-Anweisung ein Wort (Transaktionsstatus) übertragen:

- 0 = erfolgreiche Ausführung
- 1 = der gewählte Aufruf (CALL 27, 28, 122 bzw. 123) ist nicht aktiv
- 255 = für CALL 27, 28, 122 bzw. 123 wurde der SLC-Puffer gewählt und CALL 29 wird ignoriert
- alle weiteren Codes sind identisch mit CALL 90/92

Syntax:

PUSH [27, 28, 122 bzw. 123 (der zu aktivierende Aufruf)]
CALL 29
POP [Status der Transaktion]

Beispiel:

In diesem Beispiel muß CALL 122 vor der Ausführung von CALL 29 mit der internen Zeichenkette aktiviert werden. Nach der Ausführung von CALL 29 wird versucht, ein Element vom Ganzzahlfile 10, beginnend mit Element 0 des PLC-5®-Prozessors an Netzknoten 3, an die interne Zeichenkette $(1) des BASIC-Moduls zu übertragen.

```
>1   REM EXAMPLE PROGRAM
>10  REM EXECUTE DF1 PLC REMOTE READ FROM INTERNAL
>20  REM STRING WITH NO SLC INTERVENTION
>21  REM SET UP CALL 122
>25  PUSH 5, 3, 10, ASC(N), 0, 10, 10, 1, 1, 1: CALL 122: POP
     STATUS
>30  PUSH 122
>40  CALL 29
>50  POP S
>60  IF (S=1) THEN PRINT “CALL 122 NOT ACTIVE”
>70  IF (S=255) THEN PRINT “SLC FILE CHOSEN FOR CALL 122”
>80  IF (S=0) THEN PRINT “SUCCESSFUL TRANSFER”
>90  IF (S<>0) THEN PRINT “UNSUCCESSFUL TRANSFER”
>100 END
```

Im Gegensatz zu den Aufrufen 27, 28, 122 und 123 ist für CALL 29 bei Verwendung eines SLC-Files als Quelle bzw. Ziel die SLC-Bit-Handshake-Quittierung am Ende des Befehls nicht erforderlich.

CALL 31 – Anzeige der aktuellen PRT2-Portkonfiguration

Funktion:

Mit CALL 31 wird die aktuelle Konfiguration des PRT2-Ports auf dem an den Programmierport angeschlossenen Terminalbildschirm angezeigt. Die Übertragung von Argumenten in PUSH- und POP-Anweisungen ist nicht erforderlich.

Syntax:

CALL 31

Beispiel:

```
>CALL 31

1200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff

READY
>
```

CALL 37 – Löschen der PRT2-Eingangs-/Ausgangspuffer

Funktion:

Mit CALL 37 wird der Eingangs- und/oder Ausgangspuffer des Peripherieports gelöscht. Mit Hilfe der folgenden PUSH-Anweisungen können Sie den betreffenden Puffer löschen:

- PUSH 0 – Ausgangspuffer löschen
- PUSH 1 – Eingangspuffer löschen
- PUSH 2 – beide Puffer löschen

Syntax:

PUSH [gewählter Puffer]
CALL 37

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM CLEARS THE OUTPUT BUFFER
>20 CALL 37
>30 END

READY
>RUN

READY
>
```

CALL 54 – Übertragung des BASIC-Ausgangspuffers an das CPU-Eingangsabbild

Funktion:

Mit CALL 54 können Sie die Worte 200 bis 207 des Ausgangspuffers des BASIC-Moduls in die Worte 0 bis 7 der CPU-Eingangsdatentafel übertragen. Diese Routine verfügt über keine Eingangsargumente und ein Ausgangsargument. Das Ausgangsargument ist der Status des Logikprozessors.

Wort 200 des BASIC-Ausgangspuffers ist reserviert und darf nicht geändert werden. Über dieses Wort werden Statusinformationen des Moduls an den SLC-Prozessor gesendet. Bit 15 ist das Modulmodus-Bit, das einen der folgenden Werte besitzen kann:

- 0 = SLC-Prozessor befindet sich im Run-Modus
- 1 = SLC-Prozessor befindet sich nicht im Run-Modus

Bit 14 des Wortes 200 ist das EEPROM-Prüfsummenbit, das einen der folgenden Werte besitzen kann:

- 0 = EEPROM-Prüfsumme ist korrekt
- 1 = EEPROM-Prüfsumme ist falsch

Bit 13 des Wortes 200 ist das Batteriestatus-Bit, das einen der folgenden Werte besitzen kann:

- 0 = Batteriespannung ist ausreichend
- 1 = Batteriespannung ist schwach

Während der Übertragung ist nur die Wortintegrität und nicht die Fileintegrität gewährleistet. Um die Fileintegrität sicherzustellen, können Sie im Anwendungsprogramm Handshake-Bits verwenden.

Syntax:

CALL 54
POP [Prozessormodus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>30 CALL 54 : REM XFER BASIC OUTPUT BUFFER TO CPU INPUT
    IMAGE TABLE
>40 POP X : REM LOGIC PROCESSOR STATUS IS IN X
>50 IF X<>0 THEN PRINT “PROCESSOR NOT IN RUN MODE”

READY
>RUN

READY
>
```

CALL 57 – Übertragung des BASIC-Ausgangspuffers in den CPU-M1-File

Funktion:

Mit CALL 57 können Sie bis zu 64 Worte, beginnend mit Wort 100, aus dem Ausgangspuffer des BASIC-Moduls in den M1-File der CPU-Einheit, beginnend mit Wort 0, übertragen. Diese Routine verfügt über ein Eingangs- und ein Ausgangsargument. Das Eingangsargument spezifiziert die Anzahl der zu übertragenden Worte (1 bis 64). Liegt der Wert nicht innerhalb dieses Bereichs, findet keine Übertragung statt, und das Ausgangsargument erhält den Wert 10.

Wenn das Eingangsargument einen gültigen Wert aufweist, spezifiziert das Ausgangsargument den Status des Logikprozessors. Dieser entspricht einem der folgenden Werte:

- 0 = Übertragung erfolgreich, SLC-Prozessor im Run-Modus
- 1 = Übertragung erfolgreich, SLC-Prozessor im Program-Modus
- 2 = Übertragung erfolgreich, SLC-Prozessor im Test-Modus
- 10 = Spezifikation einer ungültigen Länge
- 11 = SLC-Prozessor unterstützt diese Funktion nicht

Während der Übertragung ist nur die Wortintegrität und nicht die Fileintegrität gewährleistet. Um die Fileintegrität sicherzustellen, können Sie im Anwendungsprogramm Handshake-Bits verwenden.

Syntax:

PUSH [Anzahl der zu übertragenden Worte]
CALL 57
POP [Prozessormodus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>50 PUSH 64 : REM TRANSFER LENGTH IS 64 WORDS
>60 CALL 57 : REM TRANSFER BASIC OUTPUT BUFFER TO M1
>70 POP X : REM LOGIC PROCESSOR STATUS IS IN X
>80 PRINT X

READY
>RUN 0

READY
>
```

CALL 85 – Übertragung vom BASIC-Ausgangspuffer in den gemeinsamen DH-485-Schnittstellenfile

Funktion:

Mit CALL 85 können Sie bis zu 40 Worte in den gemeinsamen DH-485-Schnittstellenfile übertragen.

Diese Routine verfügt über zwei Eingangsargumente und ein Ausgangsargument. Das erste Eingangsargument ist der beginnende Wortversatz des gemeinsamen DH-485-Schnittstellenfiles. Liegt der Wortversatzwert nicht innerhalb des Bereichs 0 bis 127, wird als Ausgangsargument der Wert 1 ausgegeben. Das zweite Eingangsargument ist die Anzahl der vom Ausgangspuffer des BASIC-Moduls in den gemeinsamen DH-485-Schnittstellenfile zu übertragenden Worte. Liegt das zweite Eingangsargument nicht im gültigen Bereich (1 bis 40), wird als Ausgangsargument der Wert 2 ausgegeben.

Das Ausgangsargument spezifiziert den Übertragungsstatus, der einen der folgenden Werte annehmen kann:

- 0 = Übertragung erfolgreich
- 1 = ungültiger beginnender Versatz
- 2 = ungültige Länge

Während dieser Übertragung ist nur die Wortintegrität und nicht die Fileintegrität gewährleistet.

Syntax:

PUSH [beginnender Wortversatz im DH-485-Schnittstellenfile]
PUSH [Anzahl der zu übertragenden Worte]
CALL 85
POP [Übertragungsstatus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 31 : REM OFFSET ADDRESS = 31
>50 PUSH 3 : REM WORD LENGTH = 3
>60 CALL 85 : REM TRANSFER DATA TO DH-485 COMMON INTERFACE
    FILE
>70 POP R
>80 IF R<>0 PRINT “TRANSFER ERROR CODE = ”,R : REM PRINT
    ERROR

READY
>RUN

READY
>
```

CALL 91 – Schreibtransfer vom BASIC-Ausgangspuffer in den dezentralen DH-485-Datenfile

Funktion:

Mit CALL 91 können Sie bis zu 40 Worte, beginnend mit Wort 0, vom Ausgangspuffer des BASIC-Moduls in den dezentralen DH-485-Datenfile schreiben. Dabei müssen Sie die Netzknotenadresse, die Filenummer, den Filetyp und den Elementversatz spezifizieren. Diese Routine verfügt über sechs Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist die Netzknotenadresse des dezentralen Gerätes (0 bis 31). Liegt die Adresse nicht in diesem Bereich, wird dem Ausgangsargument der Wert 10 zugewiesen, und der Schreibtransfer findet nicht statt.

Das zweite Eingangsargument ist die Filenummer des dezentralen Gerätes (0 bis 255). Wenn die Nummer nicht in diesem Bereich liegt, entspricht das Ausgangsargument dem Wert 11, und der Schreibtransfer wird nicht ausgeführt.

Das dritte Eingangsargument ist der an das dezentrale Gerät übertragene Filetyp. Folgende Typencodes sind zulässig: ASC(N), ASC(S), ASC(C), ASC(T), ASC(B) und ASC(R). Wird ein anderer Filetyp als die hier genannten ausgewählt, wird dem Ausgangsargument der Wert 241 zugewiesen, und der Schreibtransfer findet nicht statt.

Tabelle 12.D
Filetyp des Schreibtransfers an das dezentrale Gerät

Filetyp	Filetyp-Code	Worte/Element
Ganzzahlfile	ASC(N)	1 Wort/Element
Statusfile	ASC(S)	1 Wort/Element
Zählerfile	ASC(C)	3 Worte/Element
Zeitwerkfile	ASC(T)	3 Worte/Element
Bitfile	ASC(B)	1 Wort/Element
Steuerfile	ASC(R)	3 Worte/Element

Das vierte Eingangsargument ist der Versatz des beginnenden Elementes im File des dezentralen Gerätes (0 bis 32767). Wenn der Wert außerhalb dieses Bereichs liegt, wird dem Ausgangsargument der Wert 12 zugewiesen, und der Transfer findet nicht statt.

Wichtig: Der Versatz ist zweimal so groß wie erwartet. Beispiel: Wenn der Versatz 3 mit der PUSH-Anweisung in den Stapelspeicher eingespeichert wird, werden die Daten, beginnend mit Element 6, an den dezentralen DH-485-Datenfile übertragen.

Das fünfte Eingangsargument ist die Anzahl der zu übertragenden Elemente. Wenn die Anzahl nicht im gültigen Bereich (siehe folgende Tabelle) liegt, wird dem Ausgangsargument der Wert 13 zugewiesen, und der Transfer findet nicht statt.

Tabelle 12.E
Gültiger Längenbereich

Filetyp	Gültiger Längenbereich
ASC(N)	1 bis 40
ASC(S)	1 bis 40
ASC(C)	1 bis 13
ASC(T)	1 bis 13
ASC(B)	1 bis 40
ASC(R)	1 bis 13

Das sechste Eingangsargument ist der Zeitablaufwert der Nachricht. Dieser Wert ist die Zeit (x 100 ms), die bis zum Erhalt der Schreibantwort verstreichen kann (1 bis 50 = 0,1 bis 5,0 Sekunden). Geht die Schreibantwort nicht innerhalb dieser Zeit ein, wird die Nachricht abgebrochen, und als Ausgangsargument wird der Wert 55 ausgegeben. Liegt der Zeitablaufwert nicht zwischen 1 und 50, wird dem Ausgangsargument der Wert 14 zugewiesen, und die Übertragung findet nicht statt.

Die Schreibdaten des BASIC-Ausgangspuffers werden, beginnend mit Wort 0, an das dezentrale Gerät übertragen, wobei die Anzahl der übertragenen Worte von der Elementlänge der Nachricht abhängt.

Das Ausgangsargument spezifiziert den Status des Nachrichtenbefehls. Nach der Rückkehr aus der Aufrufroutine wird das Ausgangsargument wie folgt definiert:

Tabelle 12.F
Ausgangsargument

Dezimalwert	Hexadezimal-wert	Beschreibung
0	00	Übertragung erfolgreich abgeschlossen
2	02	Zielnetzknoten kann Nachricht zum derzeitigen Zeitpunkt nicht empfangen
3	03	Zielnetzknoten kann nicht antworten, da Nachricht zu lang ist
4	04	Zielnetzknoten kann nicht antworten, weil er die Befehlsparameter nicht verarbeiten kann
5	05	BASIC-Modul ist offline (nicht an den Verbund angeschlossen)
6	06	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion nicht verfügbar ist
7	07	Zielnetzknoten antwortet nicht
10	0A	BASIC-Modul stellt eine ungültige Zielnetzknotenadresse fest
11	0B	BASIC-Modul stellt eine ungültige Filenummer fest
12	0C	BASIC-Modul stellt einen ungültigen Elementversatz im Zielfile fest
13	0D	BASIC-Modul stellt eine ungültige Zielfilelänge fest
14	0E	BASIC-Modul stellt einen ungültigen Zeitablaufwert fest
16	10	Zielnetzknoten kann aufgrund falscher Befehlsparameter oder eines nicht unterstützten Befehls nicht antworten
55	37	Zeitablauf der Nachricht ist eingetreten (Zeitablaufwert wurde überschritten)
80	50	kein verfügbarer Speicherbereich im Zielnetzknoten

Dezimalwert	Hexadezimal-wert	Beschreibung
96	60	Zielnetzknoten kann nicht antworten, weil File geschützt ist
231	E7	Zielnetzknoten kann nicht antworten, weil die angeforderte Länge zu groß ist
235	EB	Zielnetzknoten kann nicht antworten, weil er den Zugriff verweigert
236	EC	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion derzeit nicht verfügbar ist
241	F1	BASIC-Modul stellt einen ungültigen Zielfiletyp fest
250	FA	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige File-Zugriffsberechtigung besitzt
251	FB	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige Programm-Zugriffsberechtigung besitzt

Dieser Aufruf wird als geschützter logischer Schreibbefehl mit zwei Adreßbereichen implementiert.

Syntax:

PUSH [Netzknotenadresse des dezentralen Gerätes]
PUSH [Filenummer des dezentralen Gerätes]
PUSH [Filetyp des dezentralen Gerätes]
PUSH [Versatz des beginnenden Elements (x2) im File des dezentralen Gerätes]
PUSH [Anzahl der zu übertragenden Elemente]
PUSH [Zeitablaufwert der Nachricht]
CALL 91
POP [Status des Nachrichtenbefehls]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>20 PUSH 7 : REM REMOTE FILE 7
>30 PUSH ASC(N) : REM FILE TYPE = INTEGER
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10: REM WORD LENGTH = 10
>60 PUSH 5 : REM THE TIME-OUT VALUE = 0.5 SECOND
>70 CALL 91 : REM WRITE DATA FROM OUTPUT BUFFER
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF R<>0 PRINT "READ ERROR CODE =",R

READY
>RUN

READ ERROR CODE =  5

READY
>
```

CALL 93 – Schreibtransfer vom Ausgangspuffer in den dezentralen gemeinsamen DH-485-Schnittstellenfile

Funktion:

Mit CALL 93 können Sie bis zu 40 Worte, beginnend mit Wort 0, des BASIC-Ausgangspuffers in den dezentralen gemeinsamen DH-485-Schnittstellenfile der spezifizierten Netzknotenadresse, beginnend am spezifizierten Wortversatz, übertragen.

Das erste Eingangsargument ist die Netzknotenadresse des dezentralen Gerätes (1 bis 31). Liegt die Adresse nicht im Bereich zwischen 1 und 31, ist das Ausgangsargument 10, und der Schreibtransfer wird nicht ausgeführt.

Das zweite Eingangsargument ist der Versatz des beginnenden Wortes im File des dezentralen Gerätes (0 bis 32767). Wenn dieser Wert außerhalb dieses gültigen Bereichs liegt, wird dem Ausgangsargument der Wert 12 zugewiesen, und die Übertragung findet nicht statt.

Wichtig: Der Versatz ist zweimal so groß wie erwartet. Beispiel: Wenn der Versatz 3 mit der PUSH-Anweisung in den Stapelspeicher eingespeichert wird, werden die Daten beginnend bei Element 6 an den dezentralen DH-485-Datenfile übertragen.

Das dritte Eingangsargument ist die Anzahl der zu übertragenden Worte. Wenn der Wert nicht innerhalb des im folgenden spezifizierten Bereichs liegt, wird als Ausgangsargument der Wert 13 angezeigt, und die Übertragung wird nicht ausgeführt.

Das vierte Eingangsargument ist der Zeitablaufwert der Nachricht. Dieser Wert ist die Zeit (x 100 ms), die bis zum Erhalt der Schreibantwort verstreichen kann (1 bis 50 = 0,1 bis 5,0 Sekunden). Geht die Schreibantwort nicht innerhalb dieser Zeit ein, wird die Nachricht abgebrochen, und als Ausgangsargument wird der Wert 55 angezeigt. Wenn der Zeitwert nicht zwischen 1 und 50 liegt, wird dem Ausgangsargument der Wert 14 zugewiesen, und die Übertragung findet nicht statt.

Die Daten vom Ausgangspuffer des BASIC-Moduls werden, beginnend mit Wort 0, in den dezentralen gemeinsamen Schnittstellenfile, beginnend mit dem spezifizierten Wort, geschrieben, wobei die Anzahl der übertragenen Worte von der Wortlänge der Nachricht abhängt.

Das Ausgangsargument spezifiziert den Status des Nachrichtenbefehls. Nach der Rückkehr aus der Aufrufroutine wird das Ausgangsargument wie folgt definiert:

Tabelle 12.G
Ausgangsargument

Dezimalwert	Hexadezimal-wert	Beschreibung
0	00	Übertragung erfolgreich abgeschlossen
2	02	Zielnetzknoten kann Nachricht zum derzeitigen Zeitpunkt nicht empfangen
3	03	Zielnetzknoten kann nicht antworten, da Nachricht zu lang ist
4	04	Zielnetzknoten kann nicht antworten, weil er die Befehlsparameter nicht verarbeiten kann
5	05	BASIC-Modul ist offline (nicht an den Verbund angeschlossen)
6	06	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion nicht verfügbar ist
7	07	Zielnetzknoten antwortet nicht
10	0A	BASIC-Modul stellt eine ungültige Zielnetzknotenadresse fest
11	0B	BASIC-Modul stellt eine ungültige Filenummer fest
12	0C	BASIC-Modul stellt einen ungültigen Elementversatz im Zielfile fest
13	0D	BASIC-Modul stellt eine ungültige Zielfilelänge fest
14	0E	BASIC-Modul stellt einen ungültigen Zeitablaufwert fest
16	10	Zielnetzknoten kann aufgrund falscher Befehlsparameter oder eines nicht unterstützten Befehls nicht antworten
55	37	Zeitablauf der Nachricht ist eingetreten (Zeitablaufwert wurde überschritten)
80	50	kein verfügbarer Speicherbereich im Zielnetzknoten
96	60	Zielnetzknoten kann nicht antworten, weil File geschützt ist
231	E7	Zielnetzknoten kann nicht antworten, weil die angeforderte Länge zu groß ist
235	EB	Zielnetzknoten kann nicht antworten, weil er den Zugriff verweigert
236	EC	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion derzeit nicht verfügbar ist
241	F1	BASIC-Modul stellt einen ungültigen Zielfiletyp fest
250	FA	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige File-Zugriffsberechtigung besitzt
251	FB	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige Programm-Zugriffs-berechtigung besitzt

Syntax:

PUSH [Netzknotenaderesse des dezentralen Gerätes]
PUSH [Versatz des beginnenden Elementes (x2) im File des dezentralen Gerätes]
PUSH [Anzahl der zu übertragenden Worte]
PUSH [Zeitablaufwert der Nachricht]
CALL 93
POP [Status des Nachrichtenbefehls]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM WORDLENGTH = 10
>60 PUSH 5 : REM THE TIME-OUT VALUE = 0.5 SECONDS
>70 CALL 93 : REM WRITE DATA FROM BASIC OUTPUT BUFFER
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF R<>0 THEN PRINT  READ ERROR CODE = ",R


READY
>RUN

READ ERROR CODE =  5

READY
>
```

CALL 94 – Anzeige der aktuellen Konfiguration des PRT1-Ports

Funktion:

Mit CALL 94 wird die aktuelle Konfiguration des PRT1-Ports auf dem Bildschirm des Programmierterminals angezeigt. Die Übertragung von Argumenten mit PUSH- und POP-Anweisungen ist nicht erforderlich.

Syntax:

CALL 94

Beispiel:

```
>CALL 94

19200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
```

CALL 96 – Löschen der PRT1-Eingangs-/Ausgangspuffers

Funktion:

Mit CALL 96 können Sie den Eingangs- und den Ausgangspuffer des PRT1-Ports löschen, wobei die folgenden PUSH-Anweisungen verwendet werden:

- PUSH 0 – Löschen des Ausgangspuffers
- PUSH 1 – Löschen des Eingangspuffers
- PUSH 2 – Löschen beider Puffer

Wichtig: Wenn PRT1 für das DH-485-Protokoll konfiguriert ist, ist dieser Aufruf nicht funktionsfähig.

Syntax:

PUSH [Wahl des Puffers]
CALL 96

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : CALL 96 : REM CLEAR PRT1 OUTPUT BUFFER

READY
>
```

CALL 112 – Steuerung der Anwender-LEDs

Funktion:

Mit CALL 112 werden die Anwender-LEDs (LED1 und LED2) aktiviert bzw. deaktiviert. Für diesen Aufruf sind zwei Eingangs- und keine Ausgangsargumente erforderlich. Das erste Eingangsargument aktiviert bzw. deaktiviert LED1, und das zweite Eingangsargument aktiviert bzw. deaktiviert LED2. Das Eingangsargument 1 aktiviert die LED, während das Eingangsargument 0 die LED deaktiviert. Andere Werte beeinflussen die betreffende LED nicht.

Syntax:

PUSH [Zustand LED1]
PUSH [Zustand LED2]
CALL 112

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>100 PUSH 1 : REM TURN ON LED1
>110 PUSH 0 : REM TURN OFF LED2
>120 CALL 112 : REM SET THE LEDS

READY
>RUN

READY
>
```

CALL 114 – Übertragung eines DF1-Datenpakets

Funktion:

Mit CALL 114 wird das DF1-Datenpaket übertragen. Diese Routine verfügt über keine Eingangs- und keine Ausgangsargumente. Bei der Ausführung dieses Aufrufs werden die DF1-Daten für den Abruf durch den DF1-Treiber für die Übertragung als einzelnes Nachrichtenpaket bereitgestellt. Im Halbduplex-Slave-Betrieb wird das Nachrichtenpaket bei Erhalt des nächsten ENQ-Signals vom DF1-Master übertragen. Im Vollduplexbetrieb wird das Nachrichtenpaket sofort übertragen.

Zur Konfiguration der gewünschten Daten im Sendepuffer des PRT2-Ports können eine oder mehrere PRINT#-, PH0.#- oder PH1.#-Anweisungen verwendet werden. Nach der Konfiguration der Daten im Sendepuffer wird mit CALL 114 die Übertragung in Form eines DF1-Nachrichtenpakets eingeleitet.

Lassen Sie bei der Konfiguration der DF1-Datenpakete Vorsicht walten. Wenn versucht wird, fünf oder weniger Datenbytes (das Minimum ist sechs Bytes) zu übertragen, wird die Fehlermeldung **`ERROR: DF1 DATA PACKET TO TRANSMIT IS TOO SMALL`** an den Programmierport gesendet, und das BASIC-Modul wird in den Befehlsmodus geschaltet.

Wenn versucht wird, mehr als 256 Datenbytes in den Sendepuffer zu übertragen, wird die Fehlermeldung **`ERROR: BUFFER OVERFLOW`** an den Programmierport gesendet, und das BASIC-Modul wird in den Befehlsmodus geschaltet.

Das Programm muß warten, bis eine Übertragung vollständig ausgeführt wurde, bevor es ein weiteres Datenpaket konfiguriert. Prüfen Sie mit CALL 115 den DF1-Übertragungsstatus, um festzustellen, ob die Übertragung abgeschlossen ist.

Syntax:

CALL 114

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 114
>20 END
```

CALL 115 – Überprüfung des DF1-Sendestatus

Funktion:

Mit CALL 115 können Sie den DF1-Sendestatus überprüfen. Diese Routine erfordert kein Eingangs- und ein Ausgangsargument. Der Wert des Ausgangsarguments kennzeichnet den DF1-Sendestatus. Die für diesen Status verwendeten Werte sind nachfolgend aufgeführt:

- 0 = kein Übertragungsergebnis anstehend
- 1 = Übertragungsergebnis anstehend
- 2 = Übertragung erfolgreich ausgeführt
- 3 = Übertragung nicht erfolgreich ausgeführt
- 4 = Aufforderungszeitablauf, keine Übertragung – dieser Status sollte niemals im Vollduplexmodus angezeigt werden
- 5 = Bei Modem-Handshake-Quittierung trat entweder ein Verlust des CTS-Signals während der Übertragung oder eine nicht behebbare Senderstörung ein. Wenn die Modem-Handshake-Quittierung nicht gewählt wurde, trat eine nicht behebbare Senderstörung ein.
- 6 = Bei Modem-Handshake-Quittierung mit Dauerträger im Halbduplex- oder Vollduplexmodus signalisiert dieser Fehler eine Übertragungsstörung, die auf eine Unterbrechung des Modems (d.h. einen DCD-Signalverlust von mehr als 10 Sekunden) zurückzuführen ist.
- 7 = DF1-Treiber ist nicht aktiviert

Wichtig: Im Vollduplex-Modus darf der Sendestatus 4 niemals angezeigt werden.

Syntax:

CALL 113
POP [DF1-Sendestatus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 113
>20 POP X
>30 END
```

CALL 123 – Schreibtransfer an einen dezentralen DF1-Datenfile

Funktion:

Mit CALL 123 können bis zu 64 Datenworte aus dem CPU-Ausgangsabbildfile, dem CPU-M0-File und/oder aus einer Zeichenkette im BASIC-Modul an einen dezentralen DF1-Netzknoten (PLC-2®-, -3®- oder -5-Prozessor) übertragen werden.

Die folgende Tabelle enthält eine Auflistung bestimmter Anmerkungen, die bei der Ausführung des Aufrufs 123 mit einem PLC-3- bzw. PLC-5-Prozessor gelten.

Tabelle 12.H
Anmerkungen zu PLC-Anwendungen

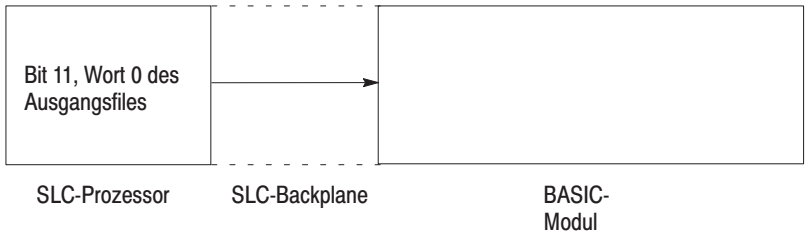
PLC	Anmerkungen
–3	Bei Zeitwerken und Zählern ist die mit einer PUSH-Anweisung in den Stapel eingespeicherte Filenummer (dritter Parameter) die Strukturnummer. Diese ist auf maximal 255 Worte begrenzt.
	Eingangs- und Ausgangsfiles können mit diesem Aufruf nicht aufgerufen werden. Bei Wahl dieser Filetypen wird vom Stapelspeicher der Wert 2 (ungültiger Eingangsparameter) ausgegeben.
–5	Bei Zeitwerkdaten besteht ein Element aus drei 16-Bit-Worten, die in der folgenden Reihenfolge im Quellfile gespeichert sind: Steuerung, Sollwert und Istwert.

Bei Wahl einer internen Zeichenkette wird das erste Zeichen (Transaktionsnummer) nach dem erfolgreich ausgeführten Schreibtransfer inkrementiert, um dem BASIC-Modul zu signalisieren, daß die Daten der Zeichenkette an den PLC-Prozessor übertragen wurden. Der Wert der Transaktionsnummer geht von 255 automatisch wieder auf 0 über.

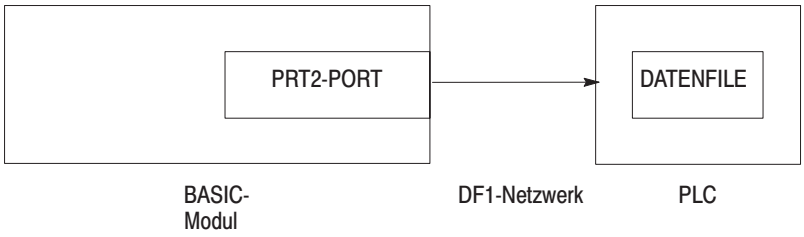
Die Parameter des DF1-Ports werden mit CALL 108 eingestellt. Der DF1-Port ist beim Vollduplex- und Halbduplex-Slave-Protokoll funktionsfähig.

Führen Sie CALL 123 einmal aus, um die Datenübertragungsparameter zu konfigurieren. Die Abbildbits der Eingangs- und Ausgangsdatentafel (Wort 0, Bit 11) des Steckplatzes, der das BASIC-Modul enthält, leiten die Datenübertragung und signalisieren den Abschluß der Übertragung ein. Dieser Vorgang ist im folgenden näher beschrieben:

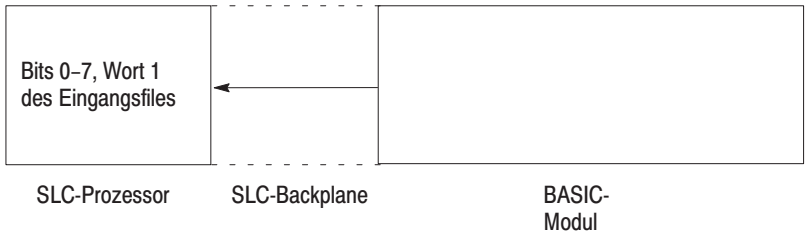
1. Der SLC-Prozessor erstellt die Pufferdaten und setzt Bit 11 in Wort 0 des Ausgangsfiles, um dem BASIC-Modul zu signalisieren, daß gültige Daten verfügbar sind.



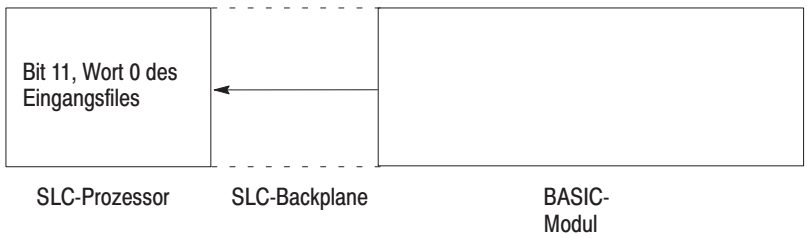
2. Das BASIC-Modul überträgt die Daten in den PLC-Datenzielfile.



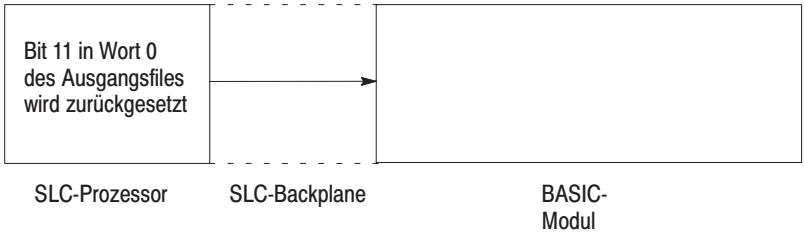
3. Das BASIC-Modul überträgt den Transaktionsstatus an die Bits 0–7 in Wort 1 des Eingangsfiles.



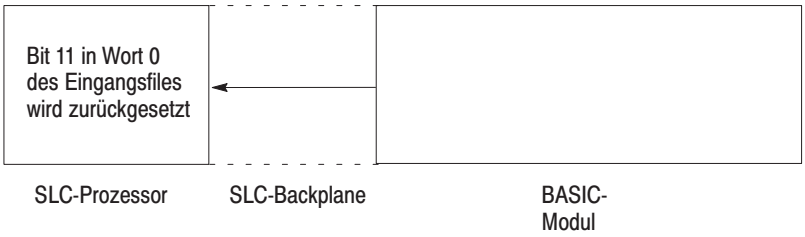
4. Das BASIC-Modul setzt Bit 11 in Wort 0 des Eingangsfiles, um dem SLC-Prozessor zu signalisieren, daß die Daten übertragen wurden und daß der Übertragungsstatus gültig ist.



5. Der SLC-Prozessor ruft den Status ab und setzt Bit 11 in Wort 0 des Ausgangsfiles zurück. Der Datenpuffer kann vom SLC-Prozessor neu belegt werden.



6. Das BASIC-Modul setzt Bit 11 in Wort 0 des Eingangsfiles an demselben Ende des Abfragezyklus, an dem Bit 11 in Wort 0 des Ausgangsfiles zurückgesetzt wurde, zurück.



Dieser Aufruf ist so lange aktiv, bis er mit anderen Eingangsparametern erneut ausgeführt wird.

Dieser Aufruf verfügt über zehn Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist der Typ des erteilten PLC-WRITE-Befehls:

- 0 = Deaktivierung des zuvor ausgeführten Aufrufs
- 2 = gemeinsamer Schnittstellenfile – ungeschützter WRITE-Befehl eines PLC-2-Prozessors
- 3 = PLC-3-File – Wortbereichs-WRITE-Befehl
- 5 = PLC-5-File – typischer WRITE-Befehl

Das zweite Eingangsargument ist die Netzknotenadresse (im Dezimalformat) des dezentralen PLCs (0 bis 254). Wenn die Adresse nicht in diesem Bereich liegt, ist der Status gleich 2, und die Schreibnachricht wird nicht übertragen.

Das dritte Eingangsargument ist die an den dezentralen PLC übertragene Filenummer (0 bis 255). Wenn die Nummer nicht in diesem Bereich liegt, ist der Status gleich 2, und die Schreibnachricht wird nicht übertragen. Der Parameter wird ignoriert, wenn im ersten Eingangsparameter der gemeinsame Schnittstellenfile gewählt wurde, muß aber trotzdem mit einer PUSH-Anweisung in den Stapel eingespeichert werden.

Das vierte Eingangsargument ist der Typ des Zielfiles im dezentralen Gerät. Dieser Wert wird ignoriert, wenn im ersten Parameter der gemeinsame Schnittstellenfile gewählt wurde (Ganzzahlfile wird vorausgesetzt). Wenn ein anderer, nicht in der folgenden Tabelle enthaltener Filetyp eingegeben wird, ist der Status gleich 2, und die Schreibnachricht wird nicht übertragen.

Tabelle 12.I
Typ des an das dezentrale Gerät übertragenen Files

Filetyp	Filetype-Code	Worte/Element (1 Wort = 16 Bits)
Ganzzahlfile	ASC(N)	1 Wort/Element
Statusfile	ASC(S)	1 Wort/Element
Zählerfile	ASC(C)	3 Worte/Element
Zeitwerkfile	ASC(T)	3 Worte/Element
Bitfile	ASC(B)	1 Wort/Element
Steuerfile	ASC(R)	3 Worte/Element
Eingangsfile	ASC(I)	1 Wort/Element
Ausgangsfile	ASC(O)	1 Wort/Element

Das fünfte Eingangsargument ist der Versatz des beginnenden Wortes im File des dezentralen PLC-2-Prozessors (0 bis 32766). Bei PLC-3-Ganzzahl-, -Binär- und -Statusfiles ist der Wert 0–9999 (dezimal). Bei PLC-3-E/A-Files ist der Wert 0–4095 (dezimal). Bei PLC-3-Zeitwerk- und -Zählerfiles ist der Wert 0. Wenn der Wert nicht im gültigen Bereich liegt, ist der Status gleich 2, und die Übertragung findet nicht statt.

Das sechste Eingangsargument ist die Anzahl der zu übertragenden Elemente. Wenn die Anzahl nicht im jeweiligen Bereich (siehe Tabelle) liegt, ist der Status gleich 2, und die Übertragung wird nicht ausgeführt.

Tabelle 12.J
Bereich der gültigen Elementlängen

Filetyp-Code	Bereich der gültigen Elementlänge
ASC(N)	1 bis 64
ASC(S)	1 bis 64
ASC(C)	1 bis 21
ASC(T)	1 bis 21
ASC(B)	1 bis 64
ASC(R)	1 bis 21
ASC(I)	1 bis 64
ASC(O)	1 bis 64
gemeinsamer Schnittstellenfile	1 bis 64

Das siebte Eingangsargument ist der Zeitablaufwert der Nachricht. Dieser Wert (1 bis 255) entspricht der Zeit (x 100 ms), die verstreichen kann, bis die Schreibantwort erhalten wird (0,1 bis 25,5 Sekunden). Geht die Schreibantwort nicht innerhalb dieser Zeit ein, wird die Nachricht abgebrochen und als Status der Wert 55 in Wort 1 des Eingangsfiles übertragen. Wenn der Zeitablaufwert nicht innerhalb des gültigen Bereichs (1 bis 255) liegt, wird in der POP-Anweisung als Statusausgangsargument der Wert 2 angezeigt, und die Übertragung wird nicht ausgeführt.

Das achte Eingangsargument ist die Wahl des CPU-Ausgangsabbildquellfiles, des CPU-M0-Files oder der internen Zeichenkette:

- 0 = CPU-Ausgangsabbildfile
- 1 = CPU-M0-File
- 2 = interne Zeichenkette

Wenn Sie die interne Zeichenkette (2) wählen, kann CALL 29 ausgeführt werden, um die einzelnen Datentransfers ohne die Interaktion eines SLC-Prozessors einzuleiten. Zeichenkettentransaktionen werden auch mit Bit 11 in Wort 0 des Ausgangsfiles initiiert.

Das neunte Ausgangsargument ist der Wortversatz im CPU-File. Dieser Versatz zeigt auf das erste Datenwort. Wenn Sie den CPU-Ausgangsabbildfile (0) wählen, kann der Versatz nicht 0 sein, da dieses Wort für die Handshake-Bits der Datenübertragung reserviert ist. Der Versatz der internen Zeichenkette ist immer 1. Das erste Zeichen der Zeichenkette (Transaktionsnummer der Adresse 0) wird bei jedem erfolgreichen Transfer inkrementiert, um dem BASIC-Modul zu signalisieren, daß die Daten übertragen wurden. Der Wert der Transaktionsnummer geht automatisch von 255 auf 0 über.

Das zehnte Eingangsargument ist die Zeichenkettennummer. Wenn im achten Eingangsargument nicht die interne Zeichenkette gewählt wurde, wird der Wert dieses Eingangsarguments ignoriert, muß aber dennoch in der PUSH-Anweisung enthalten sein.

Das Ausgangsargument ist die Bestätigung des Aufrufs, wobei die folgenden Werte gelten:

- 0 = erfolgreich
- 1 = deaktiviert
- 2 = ungültiger Eingangsparameter
- 3 = DF1 nicht aktiviert
- 4 = Zeichenkette zu klein
- 5 = Zeichenkette nicht dimensioniert

Zur Deaktivierung dieses Aufrufs muß in die PUSH-Anweisung des ersten Eingangsparameters eine Null eingegeben werden. Alle weiteren Parameter werden ignoriert, müssen aber dennoch mit einer PUSH-Anweisung programmiert werden.

Bei jedem Versuch, einen Schreibtransfer an ein dezentrales Gerät auszuführen, überträgt das BASIC-Modul den Status des Schreibtransfers in die Bits 0–7 des Eingangswortes 0. Die möglichen Statuscodes sind in der folgenden Tabelle aufgeführt. Der jeweilige Status ist gültig, wenn das BASIC-Modul Bit 11 in Wort 0 des Eingangsfiles setzt.

Tabelle 12.K
Transaktionsstatuscodes

Code	Bedeutung
0	Übertragung erfolgreich
1	Übertragung nicht erfolgreich
2	Aufforderungszeitablauf
3	Handshake-Modus gewählt – entweder ist ein Verlust des CTS-Signals während der Übertragung oder eine nicht behebbare Senderstörung eingetreten
	Handshake-Modus abgewählt – eine nicht behebbare Senderstörung ist eingetreten
4	Bei Modem-Handshake-Quittierung mit Dauerträger eine Übertragungsstörung, die auf eine Unterbrechung des Modems (d.h. einen DCD-Signalverlust von mehr als 10 Sekunden) zurückzuführen ist
5	DF1-Treiber nicht aktiviert
6	Zeitablauf der Nachricht
81	unzulässiger Befehl bzw. unzulässiges Format

Code	Bedeutung
82	eine am Host vorliegende Störung verhindert die Kommunikation
83	Station des dezentralen Hosts nicht anwesend, nicht angeschlossen oder ausgeschaltet
84	Host kann Funktion aufgrund einer Hardwarestörung nicht durchführen
85	Adressierungsproblem oder speichergeschützte Strompfade
86	Funktion kann nicht ausgeführt werden, weil der Befehlsschutz gewählt wurde
87	Prozessor ist in den Programmiermodus geschaltet
88	Kompatibilitätsfile fehlt oder es liegt ein Problem bezüglich der Kommunikationszone vor
89	dezentrale Station kann Befehl nicht in den Puffer einspeichern
8B	Herunterladevorgang verursachte Störung an der dezentralen Station
8C	zentrale Station kann Befehl aufgrund aktivierter IPBs nicht ausführen
C1	unzulässiges Adreßformat – Datenfeld enthält einen unzulässigen Wert
C2	unzulässiges Adreßformat – unzureichende Anzahl von Datenfeldern spezifiziert
C3	unzulässiges Adreßformat – zu hohe Anzahl von Datenfeldern spezifiziert
C4	unzulässiges Adreßformat – Symbol kann nicht gefunden werden
C5	unzulässiges Adreßformat – Symbol ist 0 oder größer als die von diesem Gerät unterstützte maximale Zeichenanzahl
C6	unzulässige Adresse – Adresse nicht vorhanden oder Adresse zeigt nicht auf einen nutzbaren Wert in diesem Befehl
C7	unzulässige Größe – Filegröße ist falsch; Adresse überschreitet Fileende
C8	Aufforderung kann nicht vollständig ausgeführt werden
C9	Daten bzw. File zu groß
CA	Aufforderung zu groß; Transaktionsgröße plus Wortadresse zu groß
CB	Zugriff verweigert, Privilegverletzung
CC	Ressource nicht verfügbar; Zustand kann nicht erzeugt werden
CD	Ressource bereits verfügbar; Zustand bereits vorhanden
CE	Befehl kann nicht ausgeführt werden
CF	Überlauf, Histogrammüberlauf
D0	kein Zugriff
D1	unzulässiger Datentyp
D2	ungültiger Parameter; ungültige Daten im Such- oder Befehlsblock
D3	Adresse bezieht sich auf gelöschten Bereich
D4	Befehlsausführung aus unbekanntem Grund nicht erfolgreich; PLC-3-Histogrammüberlauf
D5	Datenumwandlungsfehler

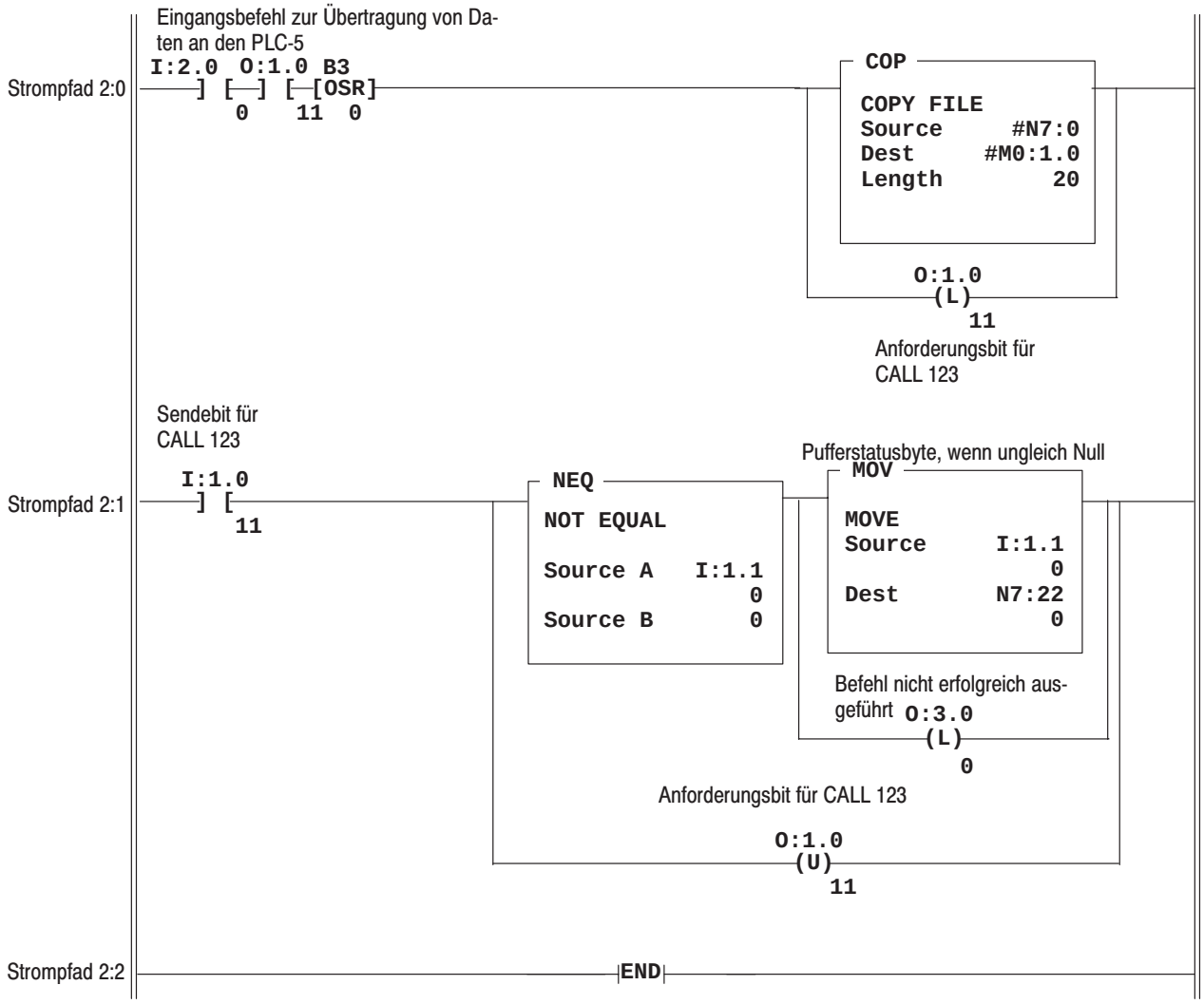
Code	Bedeutung
D6	Scanner kann nicht mit einem 1771-Chassisadapter Daten austauschen
D7	Adapter kann nicht mit dem Modul kommunizieren
D8	Antwort des 1771-Moduls war ungültig
D9	duplizierte Label-Kennzeichnung
DA	File ist geöffnet – eine andere Station hat die Zugriffsberechtigung
DB	eine andere Station hat die Zugriffsberechtigung auf das Programm

Syntax:

PUSH [Typ des PLC-WRITE-Befehls]
PUSH [Netzknotenadresse des dezentralen PLCs]
PUSH [Filenummer des dezentralen PLCs]
PUSH [Filetyp des dezentralen PLCs]
PUSH [Versatz des beginnenden Wortes im dezentralen PLC]
PUSH [Anzahl der zu übertragenden Elemente]
PUSH [Nachrichten-Zeitablaufwert]
PUSH [Wahl des Quellfiles]
PUSH [Wortversatz im Quellfile]
PUSH [Zeichenkettennummer]
CALL 123
POP [Status von CALL 123]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PLC REMOTE WRITE COMMAND
>20  PUSH 5 : REM PLC-5 FILE
>30  PUSH 0 : REM PLC-5 NODE ADDRESS
>40  PUSH 7 : REM PLC-5 FILE NUMBER
>50  PUSH ASC(N) : REM PLC-5 FILE TYPE
>60  PUSH 0 : REM STARTING WORD OFFSET FOR PLC-5
>70  PUSH 20 : REM NUMBER OF WORDS TO TRANSFER
>80  PUSH 10 : REM COMMAND TIME-OUT VALUE (X100MS)
>90  PUSH 1 : REM USE M0 FILE
>100 PUSH 0 : REM OFFSET WITHIN M0 FILE
>110 PUSH 0 : REM STRING NUMBER - NOT APPLICABLE FOR THIS
     EXAMPLE
>120 CALL 123
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 123 SETUP"
```



PRINT

Funktion:

Mit der PRINT-Anweisung wird das BASIC-Modul veranlaßt, einen Wert an das Terminal zu übertragen. Sie können den Wert von Ausdrücken, Zeichenketten, Direktwerten, Variablen und Textzeichenketten ausgeben. Die verschiedenen Größen können in der Ausgabeliste zusammengefaßt werden, indem sie durch Kommas getrennt werden. Wenn die Liste mit einem Komma beendet wird, wird der Wagenrücklauf/Zeilenvorschub unterdrückt. P. ist ein Kürzel für PRINT.

Die Werte werden nebeneinander gedruckt und jeweils durch zwei Leerstellen getrennt. Eine PRINT-Anweisung ohne Argumente sendet eine Wagenrücklauf-/Zeilenvorschubfolge an das Terminal.

Wichtig: Der Interpreter des BASIC-Moduls beendet die Ausgabe einer Zeichenkette, wenn er eine Null (0) oder ein CR-Zeichen (13) feststellt. Wenn Zeichenketten, die diese Werte enthalten, ausgegeben werden sollen, müssen die Zeichen einzeln in einer Schleifenkonfiguration gedruckt werden. Das Zeichen CR LF kann in einer PRINT-Anweisung mit einem nachstehenden Komma unterdrückt werden. Beispiel: PRINT A,

Syntax:

PRINT

Beispiel:

```
>PRINT 10*10,3*3
 100  9

>PRINT ”1746-BAS”
1746-BAS

>PRINT 5,1E3
 5  1000
```

Wichtig: Bei jeder Ausgabe von Daten über den seriellen Port unter Verwendung der Hardware- oder Software-Handshake-Quittierung (XON/XOFF) muß sichergestellt werden, daß genügend Pufferspeicher vorhanden ist. Anderenfalls wird die Ausführung des BASIC-Programms gestoppt, um auf Speicherplatz im Puffer zu warten. Wenn im Puffer Speicherplatz vorhanden ist, nimmt das BASIC-Modul die Programmausführung an der zuletzt abgearbeiteten Stelle wieder auf. Der Übertragungspuffer jedes Ports kann 256 Zeichen aufnehmen. Siehe auch die Beschreibung der Aufrufe 36, 37, 95 und 96.

Mit den Symbolen @ und # kann die Druckausgabe jeweils über PRT1 und PRT2 erfolgen.

Der Ausdruck PRINT CR wird zur Ausgabe eines Wagenrücklaufs ohne anschließenden Zeilenvorschub verwendet.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, CR,
>20 PRINT “B”

READY
>RUN

B

READY
>
```

“A” wurde ausgegeben und anschließend von “B” überschrieben.

Der Ausdruck PRINT SPC( ) ermöglicht die Ausgabe einer spezifizierten Anzahl von Leerzeichen.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, SPC(10), “B”

READY
>RUN

A          B
```

Der Ausdruck PRINT TAB( ) wird zur Ausgabe einer spezifizierten Anzahl von Tabulatorzeichen verwendet.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, TAB(1), “B”

READY
>RUN

A    B
```

Der Ausdruck PRINT USING(Fx) ermöglicht die Ausgabe aller numerischen Werte in Exponenten-Schreibweise. Das “x” kennzeichnet die Gesamtanzahl der angezeigten Mantisse-Ziffern. Vor dem Dezimalpunkt steht eine Ziffer. Der Wert “x” muß zwischen 3 und 8 liegen. Der angezeigte Wert wird an diese Grenzwerte angepaßt.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT USING(F4), 123.45678

READY
>RUN

1.234 E+2
```

Der Ausdruck PRINT USING(#.#) ermöglicht die Ausgabe aller numerischen Werte in der Dezimaldarstellung und in dem durch den Befehl spezifizierten Format.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT USING(###.##),
>20 PRINT 4.67890, 123.456
>30 PRINT .0123, .234
>40 PRINT 123.456, 2.1

READY
>RUN

4.67      123.45
0.01      0.23
123.45    2.10

READY
>
```

Der Ausdruck PRINT USING(0) ermöglicht die Wiederherstellung des vorgegebenen Ausgabemodus, sofern dieser durch den Ausdruck PRINT USING(Fx) oder PRINT USING(#.#) geändert wurde.

PH0., PH1.

Funktion:

Mit den Anweisungen PH0. und PH1. wird das BASIC-Modul zur Ausgabe eines Hexadezimalwertes über das Terminal veranlaßt. Die Funktionsweise dieser Anweisung entspricht der der PRINT-Anweisung, mit der Ausnahme, daß die Werte im Hexadezimalformat angezeigt werden. Ist die ausgegebene Zahl kleiner als 255 (0FFH), werden die beiden führenden Nullen von der PH0.-Anweisung unterdrückt. Über die PH1-Anweisung werden immer vier Hexadezimalwerte ausgegeben.

Wenn die Ausgabe mit PH0. bzw. PH1 spezifiziert wird, erscheint hinter der Zahl immer der Buchstabe H. Die Werte sind abgeschnittene Ganzzahlen. Wenn die Zahl nicht im Bereich der gültigen Ganzzahlen (d.h. zwischen 0 und 65535 [0FFFFH]) liegt, schaltet das BASIC-Modul automatisch in den normalen Druckmodus um. In diesem Fall wird nach dem Wert kein H gedruckt. Da Ganzzahlen entweder im Dezimal- oder Hexadezimalformat eingegeben werden, kann mit den Anweisungen PRINT, PH0. und PH1. eine Umwandlung von dezimal in hexadezimal und umgekehrt durchgeführt werden. Alle Kommentare, die für die PRINT-Anweisung gelten, treffen auch auf die Anweisungen PH0. und PH1. zu.

Wichtig: Bei jeder Ausgabe von Daten über den seriellen Port unter Verwendung der Hardware- oder Software-Handshake-Quittierung (XON/XOFF) muß sichergestellt werden, daß genügend Pufferspeicher vorhanden ist. Anderenfalls wird die Ausführung des BASIC-Programms gestoppt, um auf Speicherplatz im Puffer zu warten. Wenn im Puffer Speicherplatz vorhanden ist, nimmt das BASIC-Modul die Programmausführung an der zuletzt abgearbeiteten Stelle wieder auf. Der Übertragungspuffer jedes Ports kann 256 Zeichen aufnehmen. Siehe auch die Beschreibung der Aufrufe 36, 37, 95 und 96.

Syntax:

PH0., PH1.

Beispiel:

```
>PH0.2*2
 04H

>PH1.2*2
 0004H

>PH0. 100
 64H

>PH0. 1000
 3E8H

>PH1. 1000
 03E8H

>PH1. 3E8
 3.0 E+8

>PH0. PI
 03H

>
```

ST@

Funktion:

Mit der Anweisung ST@ können Fließkommazahlen des BASIC-Moduls in einer spezifizierten Adresse gespeichert werden. Der Ausdruck [Ausdr] hinter der ST@-Anweisung spezifiziert die RAM-Adresse, in der die Zahl gespeichert werden soll. Diese Anweisung liest den Wert von dem oberen Bereich des Argumentstapels und speichert ihn im RAM-Speicher in der durch [Ausdr] spezifizierten Adresse.

Zusammen mit CALL 77 ermöglicht diese Anweisung die Sicherung von Variablen in einem geschützten Speicherbereich. Dieser wird beim Einschalten oder bei der Erteilung des RUN-Befehls nicht auf Null gesetzt.

Syntax:

ST@ [Ausdr]

Beispiel:

```
>P. MTOP
 24515

P. MTOP-10*6
 24455

>PUSH 24455 : CALL 77
```

```
>P. MTOP
 24455
>1   REM EXAMPLE PROGRAM
>5   DIM A(10),B(10)
>10  REM *** ARRAY SAVE ***
>20  FOR I = 0 TO 9
>30  A(I) = I+20
>40  PUSH A(I) : REM PUT NUMBER ON STACK
>50  ST@ 5FFFH-I*6
>60  NEXT I
>70  REM *** GET ARRAY ***
>80  FOR I = 0 TO 9
>90  LD@ 5FFFH-I*6
>100 POP B(I) : REM GET NUMBER FROM STACK
>110 PRINT B(I)
>120 NEXT I

READY
>RUN

 20
 21
 22
 23
 24
 25
 26
 27
 28
 29

READY
>PUSH 5FFFH : CALL 77

>P. MTOP
 24575
```

Kapitel 13

Eingangsfunktionen

Dieses Kapitel enthält eine Beschreibung und Darstellung der Befehle, mit denen das BASIC-Modul im BASIC-Programm oder von der Befehlszeile aus Eingangsdaten von seinen externen Ports ablesen kann. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 13.A enthalten.

Tabelle 13.A
Kapitelinhalt

Funktion	Mnemonik	Seite
Ablesen einer Zeichenzeile aus dem Puffer des PRT2-Ports	INPL#	13-48
Ablesen einer Zeichenzeile aus dem Puffer des PRT1-Ports	INPL@	13-48
Ablesen einer Zeichenkette aus dem Puffer des Programmierports	INPS	13-48
Ablesen einer Zeichenkette aus dem Puffer des PRT2-Ports	INPS#	13-48
Ablesen einer Zeichenkette aus dem Puffer des PRT1-Ports	INPS@	13-48
Eingabe einer Zeichenkette oder Variablen	INPUT	13-49
Eingabe einer Zeichenkette oder Variablen über den PRT2-Port	INPUT#	13-49
Eingabe einer Zeichenkette oder Variablen über den PRT1-Port	INPUT@	13-49
Laden einer Variablen	LD@	13-51
Ablesen der Daten in der Datenanweisung	READ	13-53

CALL 22 – Datenübertragung von PRT1 bzw. PRT2 an die CPU-Files

Funktion:

Mit CALL 22 werden Daten von den seriellen Ports des BASIC-Moduls direkt an den CPU-Eingangsdatenfile, den CPU-M1-File und/oder an eine modulinterne Zeichenkette übertragen. Während des Transfers werden die Daten automatisch in 8-Bit-großen Blöcken vom Eingangspuffer des gewählten Ports an den Puffer des gewählten SLC-Prozessors und/oder an eine im BASIC-Modul enthaltene Zeichenkette übertragen und dort gespeichert. Der Transfer findet dann statt, wenn im Eingangspuffer des jeweiligen Ports die spezifizierte Zeichenanzahl festgestellt oder über den Port der anwenderdefinierte Begrenzer empfangen wird. Bei der Datenspeicherung wird im 16-Bit-Wort der Zieladresse entweder zuerst das niederwertige und anschließend das hochwertige Byte oder zuerst das hochwertige und anschließend das niederwertige Byte abgelegt. Die Daten werden entsprechend der Wortgrenzen übertragen. Bei der Übertragung einer ungeraden Byteanzahl enthält das unbelegte Byte eine Null.

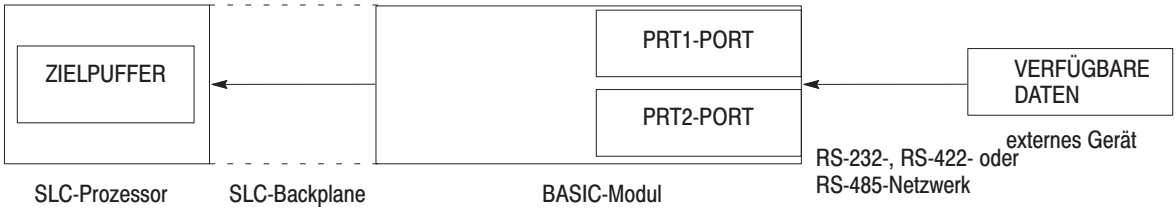
Das Datenpackformat für alle durch CALL 22 aktivierten Ports wird durch die Byteübertragungsfolge (zuerst das niederwertige und anschließend das hochwertige Byte oder zuerst das hochwertige und dann das niederwertige Byte) des zuletzt ausgeführten Aufrufs 22 bzw. 23 bestimmt.

Das niederwertige Byte des ersten Wortes im Zielfile enthält die Zeichenanzahl (Byteanzahl) der übertragenen Daten. Bei Empfang eines Begrenzers wird die Byteanzahl erweitert, um das erste Vorkommen des Begrenzers einzuschließen. Das zweite Wort des Zielfiles enthält die ersten zwei Datenzeichen.

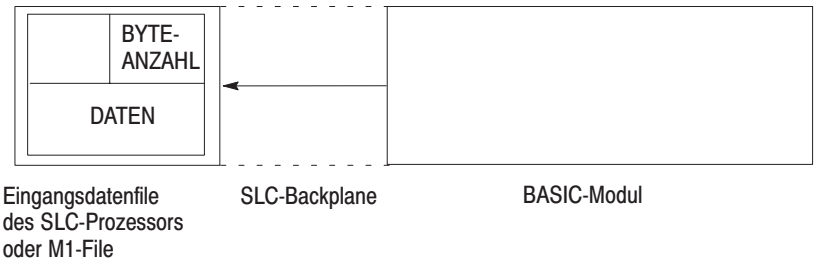
Wenn eine interne Zeichenkette als Ziel gewählt wird, enthält das erste Zeichen der Zeichenkette die Byteanzahl. Das zweite Zeichen der internen Zeichenkette enthält den Transaktionswert und wird inkrementiert, um dem BASIC-Modul zu signalisieren, das in der Zeichenkette neue Daten enthalten sind. Der Wert dieses Zeichens geht automatisch von 255 auf 0 über. Das dritte Zeichen der Kette enthält das erste Datenzeichen.

Führen Sie CALL 22 aus, um die Datentransferparameter zu konfigurieren. Nach der Ausführung dieses Aufrufs erhält das BASIC-Modul Daten über den Port und überträgt sie in den Zielfile. Mit den Bits des Eingangs- und Ausgangsabbildfiles (Wort 0, Bits 8 und 9) des Steckplatzes, in dem sich das BASIC-Modul befindet, wird die Übertragung eingeleitet und die Beendigung der Übertragung signalisiert. Dieser Vorgang ist im folgenden näher beschrieben:

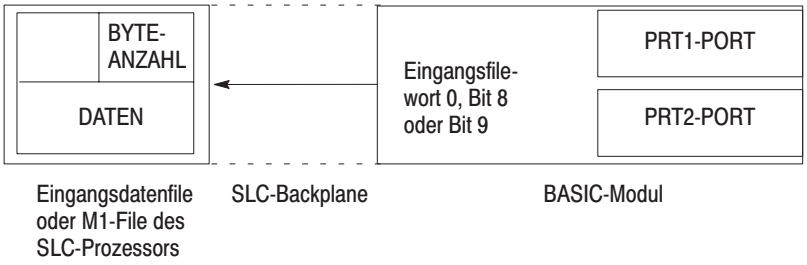
1. Wenn Daten über den Port verfügbar sind, übrträgt das BASIC-Modul diese Daten automatisch in den Zielpuffer. Am Ende jeder BASIC-Zeile wird derselbe Port auf Daten überprüft.



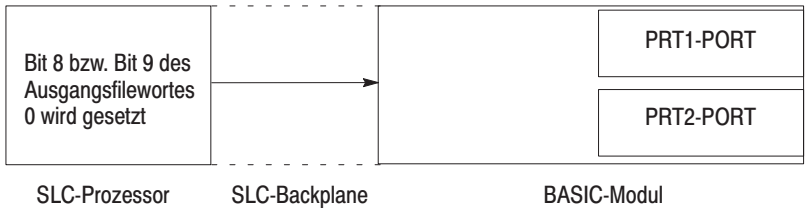
2. Das BASIC-Modul legt die Byteanzahl der gültigen Daten im unteren Byte des ersten verfügbaren Wortes des Zielpuffers ab. Das obere Byte des ersten verfügbaren Wortes wird nicht belegt.



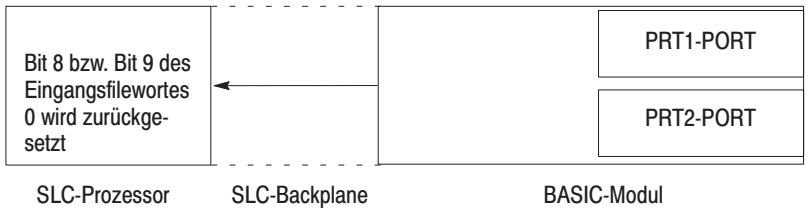
3. Das BASIC-Modul setzt Bit 8 oder Bit 9 des Eingangsfilewortes 0, um dem SLC-Prozessor zu signalisieren, daß gültige Daten verfügbar sind. Bit 8 signalisiert, daß die Daten über PRT1 verfügbar sind, während Bit 9 anzeigt, daß die Daten über PRT2 verfügbar sind.



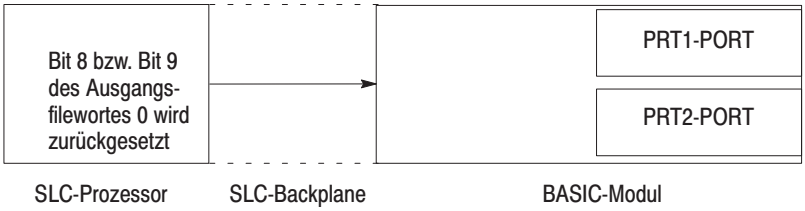
4. Das Strompfadlogikprogramm des SLC-Prozessors ruft die Daten aus dem File ab und setzt Bit 8 bzw. Bit 9 des Ausgangsfilewortes 0, um dem BASIC-Modul zu signalisieren, daß die Daten empfangen wurden.



5. Das BASIC-Modul setzt Bit 8 bzw. Bit 9 des Eingangsfilewortes 0 an demselben Ende des Abfragezyklus zurück, an dem Bit 8 bzw. Bit 9 des Ausgangsfilewortes 0 gesetzt wurde.



6. Das Strompfadlogikprogramm des SLC-Prozessors setzt Bit 8 bzw. Bit 9 des Ausgangsfilewortes 0 zurück. Das BASIC-Modul kann nun das nächste Datenpaket nach Empfang über den Port in den Zielpuffer laden.



Die Datenübertragung wird fortgesetzt, bis der Aufruf für den Port mit anderen Eingangsparametern erneut ausgeführt wird. In diesem Fall wird der vorherige CALL 22 dieses Ports automatisch deaktiviert, und der neue CALL 22 wird wirksam. Mehrfache Ausführungen dieses Aufrufs für denselben Port erfolgen nicht gleichzeitig. Port 1 und Port 2 können jedoch gleichzeitig aktiviert werden, indem CALL 22 für jeden Port separat erteilt wird.

Dieser Aufruf verfügt über sieben Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist die Nummer des Quellports (1 oder 2) des BASIC-Moduls. Durch Eingabe von Null werden alle zuvor aktiven CALL-Befehle 22 deaktiviert.

- 0 = Deaktivierung von CALL 22 für alle aktiven Ports, die durch vorhergehende Ausführungen von CALL 22 aktiviert wurden
- 1 = Port 1 als Quelle
- 2 = Port 2 als Quelle

Das zweite Eingangsargument ist die maximale Anzahl von 8-Bit-Zeichen, die über den seriellen Port des BASIC-Moduls in den Zielfile kopiert werden sollen. Die maximale Zeichenanzahl wird mit dem vierten Eingangsargument gewählt:

- CPU-Eingangsabbildfile: maximale Zeichenanzahl = 10 (5 Worte)
- CPU-M1-File: maximale Zeichenanzahl = 126 (63 Worte)
- interne Zeichenkette: maximale Zeichenanzahl = Größe der Zeichenkette –3. (Das erste Zeichen ist der Bytezählwert, das zweite der inkrementierte Transaktionswert und das letzte das Abschlußzeichen. Die maximale Zeichenanzahl einer internen Zeichenkette beträgt 254.)

Wenn bei Empfang eines Begrenzers weniger als die maximale Anzahl eingeht, wird das Datenpaket, einschließlich des Begrenzers, an den CPU-File übertragen. Mit Hilfe der in das niederwertige Byte des ersten Filewortes eingegebenen Byteanzahl bestimmt der SLC-Prozessor die Größe der gültigen Daten, die in den Zielfile übertragen werden.

Wenn die empfangenen Daten die Länge der Zeichenkette bzw. die Größe des CPU-Files überschreiten, werden die restlichen Daten abgeschnitten.

Das dritte Eingangsargument ist der Dezimalwert des ASCII-Zeichen-Begrenzers. Hierfür kann jedes gültige ASCII-Zeichen gewählt werden. Wenn kein Begrenzer verwendet werden soll, geben Sie einen NULLWERT (Dezimalwert 0) ein. Die Daten werden unabhängig von der empfangenen Zeichenanzahl in den Zielpuffer übertragen, wenn der Begrenzer über den gewählten Port empfangen wird.

Das vierte Eingangsargument ist die Wahl des CPU-Eingangsabbild-Zielfiles und des CPU-M1-Files mit oder ohne der internen Zeichenkette oder nur der internen Zeichenkette:

- 0 = CPU-Eingangsabbildfile
- 1 = CPU-M1-File
- 2 = CPU-Eingangsabbildfile und interne Zeichenkette
- 3 = CPU-M1-File und interne Zeichenkette
- 4 = nur interne Zeichenkette

Beim Datentransfer an die interne Zeichenkette des BASIC-Moduls müssen Sie die Transaktionsnummer im BASIC-Programm überprüfen, um festzustellen, ob Daten in die interne Zeichenkette eingefügt und aktualisiert wurden.

Das fünfte Eingangsargument ist der Wortversatz im CPU-Zielfile. Wenn der CPU-Eingangsdatenfile gewählt wird, darf der Versatz nicht 0 oder 1 sein, da diese Worte reserviert sind. Wort 0 ist für die Handshake-Bits des Datentransfers reserviert und Wort 1 für den Transaktionsstatus. Durch die Eingabe einer 0 oder 1 wird der CALL-Status 2 mit einer POP-Anweisung aus dem Stapelspeicher abgerufen. Dieser Versatz weist auf die Pufferadresse der Byteanzahl. Bei Wahl des M1-Files kann der Versatz Null sein. Bei Wahl der internen Zeichenkette beginnt die Datenanordnung immer mit dem dritten Zeichen der Zeichenkette. (Das erste Zeichen enthält die Byteanzahl und das zweite die Transaktionsnummer.) Deshalb hat der Versatzwert keine Auswirkung auf die Anordnung der Zeichenkettendaten, sondern nur auf die Datenanordung des Eingangsabbildfiles und des M1-Files.

Das sechste Eingangsargument ist die Zeichenkettennummer. Wenn die Belegung der internen Zeichenkette mit dem vierten Eingangsargument abgewählt wird, bleibt der Wert dieses Eingangsarguments unbeachtet, muß aber trotzdem mit einer PUSH-Anweisung in den Stapel eingespeichert werden.

Mit dem siebten Eingangsargument wird die Bytefolge gewählt. Es kann einen der folgenden Werte annehmen:

- 0 = Die vom Port des BASIC-Moduls übertragenen Datenbytes werden bei der Weiterleitung an das Ziel nicht vertauscht. Die Datenfolge je Wort lautet: zuerst das niederwertige Byte und dann das hochwertige Byte. Das niederwertige Byte des ersten Wortes im Zielfile enthält die Byteanzahl.
- 1 = Die vom Port des BASIC-Moduls übertragenen Datenbytes werden bei der Weiterleitung an das Ziel vertauscht, d.h. die Datenfolge je Wort lautet: zuerst das hochwertige Byte und dann das niederwertige Byte. Das erste Wort wird von der vertauschten Bytefolge nicht beeinflußt. Das niederwertige Byte des ersten Wortes enthält weiterhin die Byteanzahl.

Der zuletzt ausgeführte CALL 22 bzw. CALL 23 bestimmt die Bytefolge aller aktiven, zuvor ausgeführten Aufrufe 22.

Das Ausgangsargument ist der Status des Aufrufs, wobei die folgenden Werte gelten:

- 0 = erfolgreiche Konfiguration
- 1 = deaktiviert
- 2 = ungültiger Eingangsparameter
- 3 = PRT2 wurde gewählt, ist jedoch bereits für das DF1-Protokoll aktiviert. Die Übertragung wird nicht ausgeführt.
- 4 = die Zeichenkette ist zu klein
- 5 = die Zeichenkette ist nicht dimensioniert

Wenn die Daten vom seriellen Port schneller aufgenommen werden als sie vom PLC-Prozessor abgerufen werden, wird der Eingangspuffer des Ports voll. Um zu verhindern, daß Daten verlorengehen, sollte zwischen dem Port und dem externen Gerät die Handshake-Quittierung verwendet werden.

Syntax:

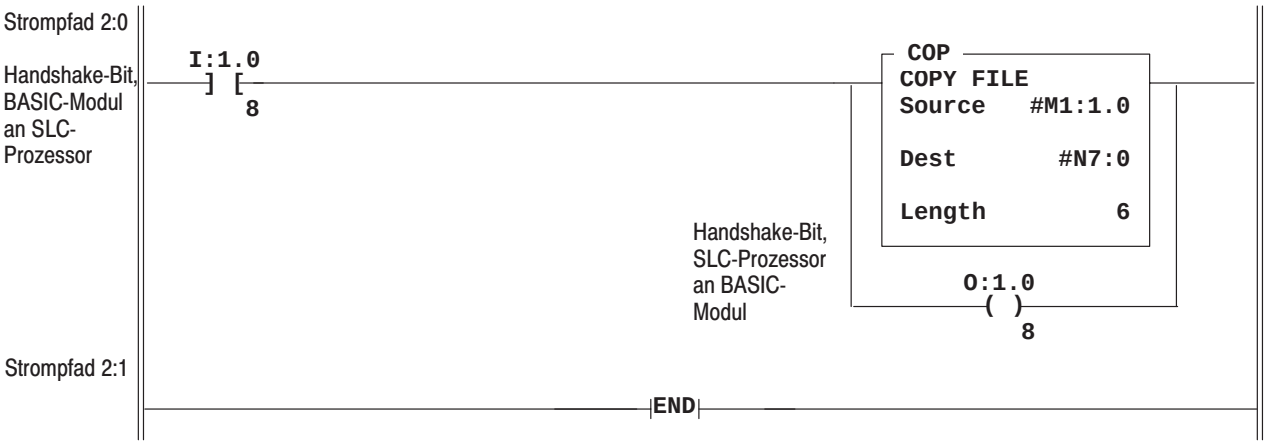
PUSH [Nummer des Quellports]
PUSH [maximale zu übertragende Zeichenanzahl]
PUSH [Dezimalwert des Zeichenbegrenzers]
PUSH [Wahl des Zielfiles und/oder der Zeichenkette]

PUSH [Wortversatz im Zielfile]
PUSH [Zeichenkettennummer]
PUSH [Wahl der Bytefolge]
CALL 22
POP [Status von CALL 22]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE CALL 22 INTERRUPTS
>20  PUSH 1 : REM PRT1 ACTIVE FOR CALL 22
>30  PUSH 10 : REM RECEIVING 10 BYTES OF DATA MAXIMUM
>40  PUSH 13 : REM <CR> USED AS TERMINATION CHARACTER
     (13 DECIMAL)
>50  PUSH 1 : REM SEND DATA TO M1 FILE
>60  PUSH 0 : REM OFFSET INTO M1 FILE
>70  PUSH 0 : REM STRING NUMBER – NOT USED
>80  PUSH 1 : REM BYTE SWAPPING ENABLED
>90  CALL 22
>100 POP S : REM STATUS OF CALL 22 SETUP
>110 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 22 SETUP”
>120 END
```

Im folgenden ist ein Strompfadlogikbeispiel für CALL 22 abgebildet. Das BASIC-Modul befindet sich in Steckplatz 1 des SLC-Racks. In Strompfad 2:0 werden die Daten aus dem M1-File kopiert, wenn das BASIC-Modul das Handshake-Bit (I:1.0/8) setzt. Der SLC-Prozessor setzt das Handshake-Bit (O:1.0/8), sobald die Daten aus dem M1-File kopiert worden sind. Dadurch wird dem BASIC-Modul signalisiert, daß es Bit (I:1.0/8) zurücksetzen muß. Das erste Wort des M1-Files enthält die Byteanzahl. Dieses Wort ist nicht in der Datenbyteanzahl enthalten. Im folgenden Beispiel werden maximal zehn Bytes erwartet.



CALL 27 – Lesen eines dezentralen DH-485-Datenfiles (SLC)

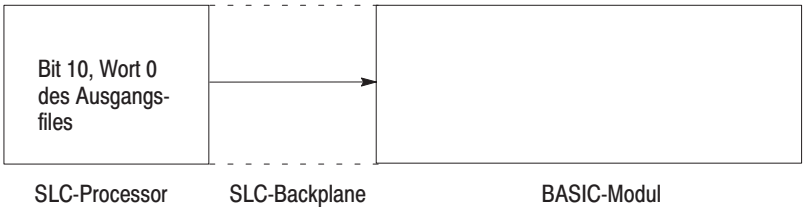
Funktion:

Mit CALL 27 können bis zu 64 Datenworte von einem dezentralen DH-485-Netzknoten abgelesen und im zentralen CPU-Eingangsabbildfile, im CPU-M1-File und/oder in einer Zeichenkette im BASIC-Modul gespeichert werden.

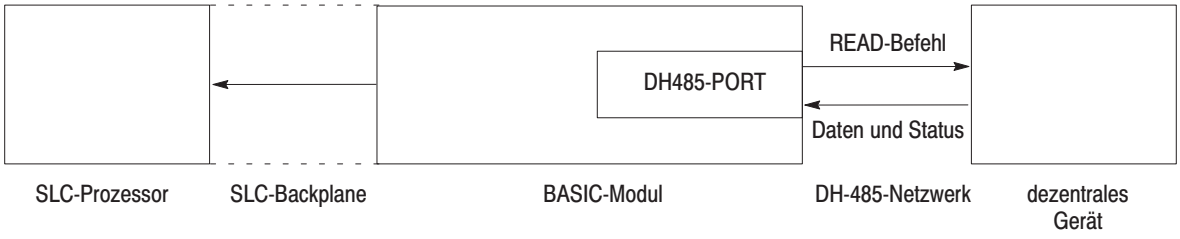
Wenn als Ziel eine interne Zeichenkette gewählt wird, wird das erste Zeichen inkrementiert, um dem BASIC-Modul zu signalisieren, daß in der Zeichenkette neue Daten enthalten sind. Der Wert dieses Zeichens geht automatisch von 255 auf 0 über.

Führen Sie CALL 27 einmal aus, um die Datenübertragungsparameter zu definieren. Zur Einleitung und Beendigung der Übertragung werden die Eingangs- und Ausgangsabbildbits (Wort 0, Bit 10) des Steckplatzes verwendet, in dem sich das BASIC-Modul befindet. Das BASIC-Modul sendet den in diesem Aufruf konfigurierten DH-485-READ-Befehl an das im Netzwerk spezifizierte DH-485-Gerät. Dieser Vorgang ist im folgenden beschrieben:

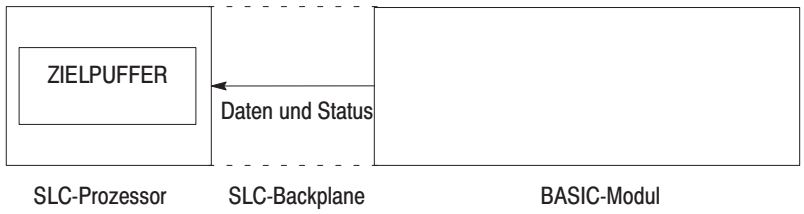
1. Der zentrale SLC-Prozessor setzt Bit 10 des Ausgangsfilewortes 0, um die Ausführung des in CALL 27 konfigurierten READ-Befehls zu veranlassen.



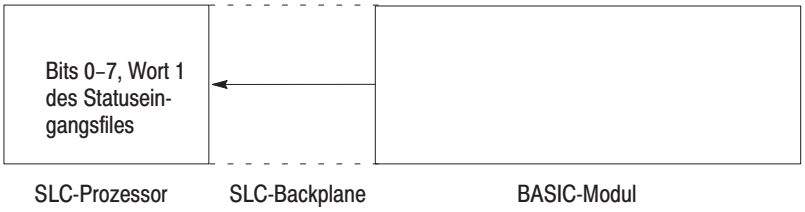
2. Das BASIC-Modul erteilt automatisch den entsprechenden READ-Befehl an das dezentrale Gerät im DH-485-Netzwerk. Die Daten und der Status werden an das BASIC-Modul zurückgesendet.



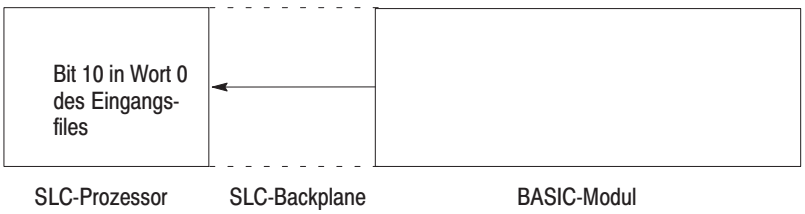
3. Wenn die Daten verfügbar sind, werden sie vom BASIC-Modul in den zentralen SLC-Zielpuffer übertragen.



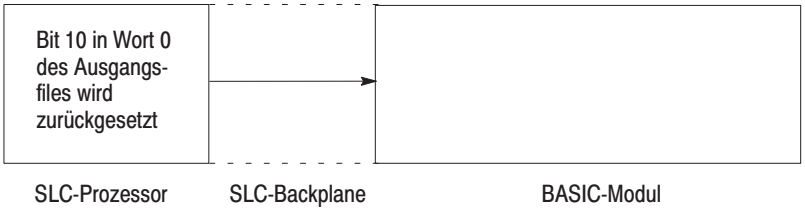
4. Das BASIC-Modul legt den Status in den Bits 0–7 des Eingangsfilewortes ab.



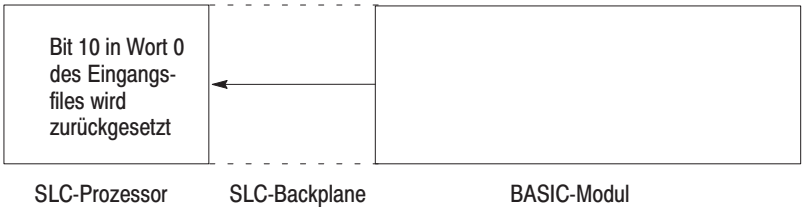
5. Das BASIC-Modul setzt Bit 10 in Wort 0 des Eingangsfiles, um dem SLC-Prozessor zu signalisieren, daß gültige Daten verfügbar sind.



6. Der zentrale SLC-Prozessor ruft die Daten und den Status aus dem Puffer ab und setzt Bit 10 in Wort 0 des Ausgangsfiles zurück, um dem BASIC-Modul zu signalisieren, daß die Daten empfangen wurden.



7. Das BASIC-Modul setzt Bit 10 in Wort 0 des Eingangsfiles an demselben Ende des Abfragezyklus zurück, an dem Bit 10 in Wort 0 des Ausgangsfiles zurückgesetzt wurde.



Der SLC-Prozessor darf Bit 10 in Wort 0 des Ausgangsfiles nicht in derselben Abfrage der Strompfadlogik setzen und zurücksetzen. Anderenfalls empfängt das BASIC-Modul möglicherweise das übertragene Bit nicht.

Dieser Aufruf bleibt aktiv, bis er mit anderen Eingangsparametern erneut ausgeführt wird.

Dieser Aufruf verfügt über zehn Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist der Typ des erteilten SLC-READ-Befehls:

- 0 = Deaktivierung des zuvor ausgeführten CALLs 27
- 1 = Lesetransfer aus dem gemeinsamen Schnittstellenfile
- 2 = eingegebener SLC-Lesebefehl

Das zweite Eingangsargument ist die Netzknotenadresse des dezentralen SLC-Gerätes (0 bis 31). Liegt die Nummer nicht in diesem Bereich, ist der Statuswert 2, und die Lesenachricht wird nicht übertragen.

Das dritte Eingangsargument ist die Filenummer des dezentralen SLC-Gerätes (0 bis 255). Liegt die Nummer außerhalb dieses Bereichs, wird der Statuswert 2 angezeigt, und die Lesenachricht wird nicht übertragen. Der Parameter wird ignoriert, wenn im ersten Parameter der gemeinsame Schnittstellenfile (CIF) gewählt wurde. Dieser ist immer File 9.

Das vierte Eingangsargument ist der vom dezentralen Gerät abzulesende Filetyp. Dieser Wert wird ignoriert, wenn im ersten Parameter der CIF-File gewählt wurde (ein Ganzzahlfile wird vorausgesetzt). Wenn der Filetyp nicht in der folgenden Auflistung enthalten ist, wird als Status der Wert 2 angezeigt, und die Lesenachricht wird nicht übertragen. Geben Sie den jeweils in der folgenden Tabelle angegebenen Filetypcode ein, wenn Sie den vierten Eingangsparameter mit einer PUSH-Anweisung in den Stapelspeicher übertragen.

Tabelle 13.B
Typ des vom dezentralen Gerät abzulesenden Files

Filetyp	Filetyp-Code	Worte/Element
Ganzzahlfile	ASC(N)	1 Wort/Element
Zählerfile	ASC(C)	3 Worte/Element
Zeitwerkfile	ASC(T)	3 Worte/Element
Bitfile	ASC(B)	1 Wort/Element
Steuerfile	ASC(R)	3 Worte/Element

Das fünfte Eingangsargument ist der Versatz des beginnenden Wortes im File des dezentralen Gerätes (0 bis 32766). Wenn der Wert nicht in diesem Bereich liegt, wird als Status der Wert 2 angezeigt, und die Übertragung findet nicht statt. (Der SLC-500-Prozessor unterstützt nur den Bereich von 0 bis 255 Worte je File.)

Das sechste Eingangsargument ist die Anzahl der zu übertragenden Worte. Wenn die Anzahl nicht im jeweiligen Bereich liegt (siehe Tabelle), wird als Status der Wert 2 angezeigt, und die Übertragung findet nicht statt. Die Prozessoren SLC 5/01 und SLC 5/02 unterstützen Übertragungen von maximal 41 Worten.

Tabelle 13.C
Bereich der gültigen Wortlängen

Filetyp-Code	Bereich der gültigen Wortlänge
ASC(N)	1 bis 64
ASC(C)	1 bis 21
ASC(T)	1 bis 21
ASC(B)	1 bis 64
ASC(R)	1 bis 21
Gemeinsamer Schnittstellenfile	1 bis 64

Das siebte Eingangsargument ist der Zeitablaufwert der Nachricht. Dieser Wert (1 bis 255) entspricht der Zeit (x 100 ms), die bis zum Erhalt der Leseantwort verstreichen kann (0,1 bis 25,5 Sekunden). Geht die Leseantwort nicht innerhalb dieser Zeit ein, wird die Nachricht abgebrochen und als Status der Wert 55 in die Bits 0–7 des Eingangsfilewortes 1 übertragen. Wenn der Zeitablaufwert nicht innerhalb des gültigen Bereichs (1 bis 255) liegt, wird in der POP-Anweisung der Statuswert 2 angezeigt, und die Übertragung wird nicht ausgeführt.

Das achte Eingangsargument ist die Wahl des Zielfiles und/oder der Zeichenkette:

- 0 = CPU-Eingangsabbildfile
- 1 = CPU-M1-File
- 2 = interne Zeichenkette
- 3 = CPU-Eingangsabbildfile und interne Zeichenkette
- 4 = CPU-M1-File und interne Zeichenkette

Wenn Sie die interne Zeichenkette (2) wählen, kann CALL 29 ausgeführt werden, um die einzelnen Datentransfers ohne die Interaktion eines SLC-Prozessors einzuleiten. Zeichenkettentransaktionen werden auch mit Bit 10 in Wort 0 des Ausgangsfiles eingeleitet.

Das neunte Eingangsargument ist der Wortversatz im CPU-Zielfile. Dieser Versatz zeigt auf das erste übertragene Datenwort. Der Versatz der internen Zeichenkette ist immer 1 (Transaktionsnummer der Adresse 0), wobei die Daten folgen. Der Wert der Transaktionsnummer wird nach jedem Datentransfer inkrementiert und geht von 255 automatisch wieder auf 0 über.

Wenn der CPU-Eingangsabbildfile gewählt wird, darf der Versatz nicht 0 oder 1 sein. Wort 0 ist für die Handshake-Bits des Datentransfers und Wort 1 für den Transaktionsstatus reserviert. Durch die Eingabe einer 0 oder 1 wird in der POP-Anweisung ein Fehler ausgegeben (2 = ungültiger Eingangsparameter), und der Aufruf wird nicht ausgeführt.

Das zehnte Eingangsargument ist die Zeichenkettennummer. Wenn im achten Eingangsargument nicht die interne Zeichenkette gewählt wurde, wird der Wert dieses Eingangsargumentes ignoriert, muß aber dennoch in der PUSH-Anweisung enthalten sein.

Das Ausgangsargument gibt den Status des Aufrufs an, wobei die folgenden Werte gelten:

- 0 = erfolgreich
- 1 = deaktiviert
- 2 = ungültiger Eingangsparameter
- 3 = DH485-Port nicht aktiviert (DF1 aktiviert)
- 4 = Zeichenkette zu klein
- 5 = Zeichenkette nicht dimensioniert

Zur Deaktivierung dieses Aufrufs muß in die PUSH-Anweisung des ersten Eingangsparameters eine Null eingegeben werden. Alle weiteren Parameter werden ignoriert, müssen aber dennoch mit einer PUSH-Anweisung programmiert werden.

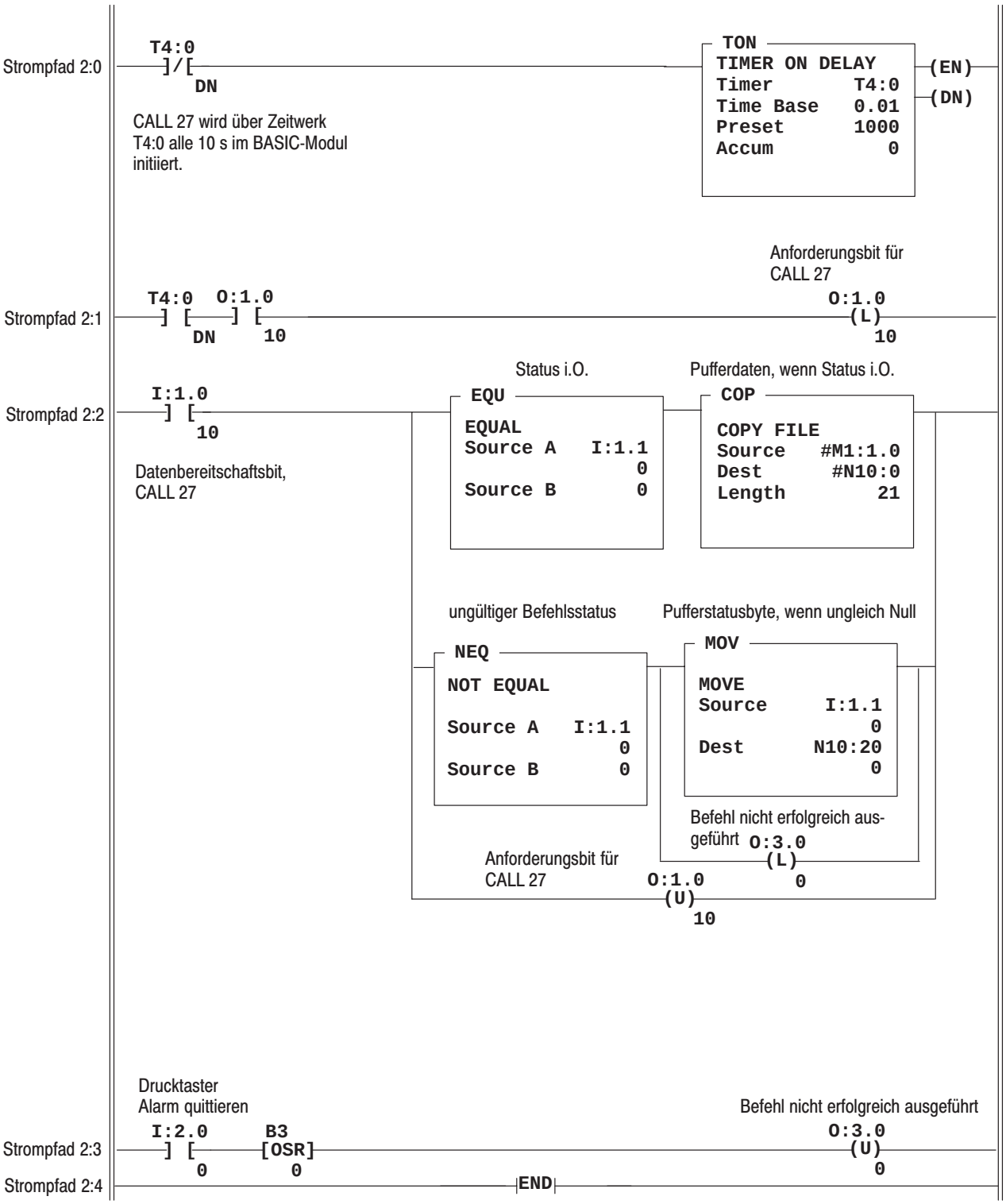
Bei jedem Versuch, einen Lesetransfer von einem dezentralen Gerät auszuführen, wird der Status des Lesetransfers in die Bits 0–7 des Eingangswortes 1 übertragen. Diese Werte sind ebenso definiert wie die mit CALL 92 über eine POP-Anweisung ausgegebenen Werte. Der Status ist gültig, wenn das BASIC-Modul Bit 10 in Wort 0 des Eingangsfiles setzt.

Syntax:

PUSH [Typ des READ-Befehls]
PUSH [dezentrale Netzknotenadresse]
PUSH [dezentrale Filenummer]
PUSH [dezentraler Filetyp]
PUSH [Versatz des beginnenden Wortes im dezentralen File]
PUSH [Anzahl der zu übertragenden Worte]
PUSH [Zeitablaufwert der Nachricht]
PUSH [Wahl des Zielfiles]
PUSH [Wortversatz im Zielfile]
PUSH [Zeichenkettennummer]
CALL 27
POP [Status von CALL 27]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE REMOTE DH-485 READ COMMAND INTERRUPT
>20  PUSH 2 : REM SLC TYPED READ COMMAND
>30  PUSH 2 : REM NODE ADDRESS OF REMOTE SLC
>40  PUSH 7 : REM FILE NUMBER OF REMOTE SLC
>50  PUSH ASC(N) : REM FILE TYPE OF REMOTE SLC
>60  PUSH 100 : REM REMOTE ELEMENT OFFSET INTO REMOTE SLC
     FILE
>70  PUSH 20 : REM NUMBER OF ELEMENTS TO BE TRANSFERRED
>80  PUSH 5 : REM MESSAGE TIMEOUT (X100MS)
>90  PUSH 1 : REM DESTINATION FILE TO PUT DATA (M1 FILE)
>100 PUSH 0 : REM WORD OFFSET INTO DESTINATION FILE
>110 PUSH 0 : REM STRING NUMBER – NOT AVAILABLE FOR THIS
     EXAMPLE
>120 CALL 27
>130 POP S
>140 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 27 SETUP”
```


Strompfad 2:0
T4:0
]/[
DN
TON
TIMER ON DELAY
Timer T4:0
Time Base 0.01
Preset 1000
Accum 0
(EN)
(DN)
CALL 27 wird über Zeitwerk T4:0 alle 10 s im BASIC-Modul initiiert.
Anforderungsbit für CALL 27
Strompfad 2:1
T4:0 O:1.0
] [ ] [
DN 10
O:1.0
(L)
10
Status i.O.
Pufferdaten, wenn Status i.O.
Strompfad 2:2
I:1.0
] [
10
EQU
EQUAL
Source A I:1.1
0
Source B 0
COP
COPY FILE
Source #M1:1.0
Dest #N10:0
Length 21
Datenbereitschaftsbit, CALL 27
ungültiger Befehlsstatus
Pufferstatusbyte, wenn ungleich Null
NEQ
NOT EQUAL
Source A I:1.1
0
Source B 0
MOV
MOVE
Source I:1.1
0
Dest N10:20
0
Befehl nicht erfolgreich ausgeführt
O:3.0
(L)
0
Anforderungsbit für CALL 27
O:1.0
(U)
10
Drucktaster Alarm quittieren
Befehl nicht erfolgreich ausgeführt
Strompfad 2:3
I:2.0
] [
0
B3
[OSR]
0
O:3.0
(U)
0
Strompfad 2:4
END

CALL 29 – Lese-/Schreibtransfer an einen PLC/SLC von der internen Zeichenkette des BASIC-Moduls

Funktion:

In Verbindung mit CALL 122 oder CALL 123 wird mit CALL 29 die Kommunikation zwischen dezentralen PLC-Prozessoren und der internen Zeichenkette des BASIC-Moduls ermöglicht, ohne daß ein Dialog über den zentralen SLC-Prozessor erforderlich ist. Ferner kann zur Kommunikation zwischen dezentralen SLC-Prozessoren und der internen Zeichenkette des BASIC-Moduls ohne den Dialog über den zentralen SLC-Prozessor CALL 29 in Verbindung mit CALL 27 oder CALL 28 eingesetzt werden. Die Aufrufe 27, 28, 122 bzw. 123 müssen vor CALL 29 im BASIC-Modul ausgeführt werden.

CALL 29 wird aktiviert, wenn die interne Zeichenkette in den Aufrufen 27, 28, 122 bzw. 123 die einzige Wahl ist. In diesem Fall ist es nicht sinnvoll, zur Einleitung des Datentransfers und zur Übertragung der Statusdaten die SLC-Eingangs- und -Ausgangsabbildfiles zu belegen. Der SLC-Prozessor ist zur Ausführung dieses Vorgangs nicht erforderlich. Wenn stattdessen ein SLC-File gewählt wird, steuert der zentrale SLC-Prozessor die Übertragung mit Hilfe der E/A-Abbildbits. Wenn in diesem Fall versucht wird, CALL 29 auszuführen, wird als Status der Wert 255 angezeigt.

Die PUSH- und die POP-Anweisung enthalten jeweils ein Argument. Das Eingangsargument ist der zu aktivierende Aufruf (CALL 27, 28, 122 oder 123).

Bei der Ausführung von CALL 29 wird ein Übertragungsversuch durchgeführt. Wenn der gewählte Aufruf (27, 28, 122 bzw. 123) nicht vor Ausführung von CALL 29 ausgeführt wurde, zeigt die POP-Anweisung den Statuswert 1 an. Wenn CALL 29 erfolgreich ausgeführt ist, wird der Wert des ersten Zeichens der Zeichenkette (Transaktionsnummer) erhöht, um zu signalisieren, daß der Transfer stattgefunden hat. Der Bereich dieses Zeichens liegt zwischen 0 und 255.

Nach der Ausführung von CALL 29 wird in der POP-Anweisung ein Wort ausgegeben, das den Transaktionsstatus kennzeichnet.

- 0 = erfolgreiche Durchführung
- 1 = der gewählte Aufruf (CALL 27, 28, 122 oder 123) ist nicht aktiviert
- 255 = für CALL 27, 28, 122 oder 123 wurde der SLC-Puffer gewählt und CALL 29 wird ignoriert
- Alle weiteren Codes sind identisch mit CALL 90/92

Syntax:

PUSH [27, 28, 122 oder 123 (der zu aktivierende Aufruf]
CALL 29
POP [Transaktionsstatus]

Beispiel:

In diesem Beispiel muß CALL 122 vor der Ausführung von CALL 29 mit der internen Zeichenkette aktiviert werden. Nach der Ausführung von CALL 29 wird versucht, ein Element vom Ganzzahlfile 10, beginnend mit Element 0 des PLC-5-Prozessors an Netzknoten 3, an die interne Zeichenkette $(1) des BASIC-Moduls zu übertragen.

```
>1   REM EXAMPLE PROGRAM
>10  REM EXECUTE DF1 PLC REMOTE READ FROM INTERNAL
>20  REM STRING WITH NO SLC INTERVENTION
>21  REM SET UP CALL 122
>25  PUSH 5, 3, 10, ASC(N), 0, 10, 10, 1, 1, 1: CALL 122: POP
     STATUS
>30  PUSH 122
>40  CALL 29
>50  POP S
>60  IF (S=1) THEN PRINT “CALL 122 NOT ACTIVE”
>70  IF (S=255) THEN PRINT “SLC FILE CHOSEN FOR CALL 122”
>80  IF (S=0) THEN PRINT “SUCCESSFUL TRANSFER”
>90  IF (S<>0) THEN PRINT “UNSUCCESSFUL TRANSFER”
>100 END
```

Bei Verwendung eines SLC-Files oder einer Zeichenkette des BASIC-Moduls ersetzt CALL 29 die Funktion der Handshake-Bits in den Aufrufen 27, 28, 122 und 123.

CALL 35 – Abruf des numerischen Eingangszeichens von PRT2

Funktion:

Mit CALL 35 wird das aktuelle Zeichen im 256 Zeichen langen Eingangspuffer des PRT2-Ports abgerufen. Die empfangenen Zeichen werden im Ausgangsargument dieses Aufrufs als Dezimalwerte ausgegeben. Der PRT2-Port empfängt die vom Gerät übertragenen Daten und speichert sie in diesem Puffer. Wenn kein Zeichen vorhanden ist, enthält das Ausgangsargument den Wert 0 (Null). Ist ein Zeichen vorhanden, entspricht das Ausgangsargument dem ASCII-Wert des Zeichens. Diese Routine verfügt über kein Ausgangsargument.

Syntax:

CALL 35
POP [ASCII-Wert des Zeichens]

Beispiele:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 35
>20 POP X
>30 IF X=0 THEN GOTO 10
>40 PRINT CHR(X)

READY
>RUN

A
READY
>
```

Im oben aufgeführten Programmbeispiel wird vorausgesetzt, daß der PRT2-Eingangspuffer das ASCII-Zeichen A enthält.

Wichtig: Eine 0 (Null) ist für einige Kommunikationsprotokolle ein gültiges Zeichen. Mit CALL 36 läßt sich die tatsächliche Anzahl der im Puffer enthaltenen Zeichen bestimmen.

Wichtig: Löschen Sie den Pufferspeicher vor der Datenspeicherung, um die Gültigkeit der Daten sicherzustellen.

```
>1   REM EXAMPLE PROGRAM
>10  REM PERIPHERAL PORT INPUT USING CALL 35
>20  STRING 200,20
>30  DIM D(254)
>40  CALL 35 : POP X
>50  IF X <>2 GOTO 40
>55  REM WAIT FOR DEVICE TO SEND START OF TEXT
>60  REM
>70  DO
>80  I=I+1
>90  CALL 35 : POP D(I): REM STORE DATA IN ARRAY
>100 UNTIL D(I)=3 : REM WAIT FOR DEVICE TO SEND END OF TEXT
>120 REM
>130 REM FORMAT AND PRINT DATA TYPES
>140 PRINT “RAW DATA=”
>150 FOR J=1 TO I : PRINT D(J),: NEXT J
>155 REM PRINT RAW DECIMAL DATA
>160 PRINT: PRINT: PRINT
>170 PRINT “ASCII DATA=”
>180 FOR J=1 TO I : PRINT CHR(D(J)),:NEXT J
>185 REM PRINT ASCII DATA
>190 PRINT: PRINT: PRINT
>200 PRINT “$(1)=”
>210 FOR J=1 TO I: ASC($(1),J)=D(J): NEXT J
>215 REM STORE DATA IN STRING
>220 PRINT $(1)
>230 PRINT: PRINT: PRINT
>240 I=0
>250 REM
>260 GOTO 40

READY
>RUN

RAW DATA=
 65 66 67 68 69 70 71 49 50 51 52 53 54 55 56 57 3

ASCII DATA=
 ABCDEFG123456789

$(1)=
 ABCDEFG123456789
```

CALL 53 – Übertragung des CPU-Ausgangsabbilds in den BASIC-Eingangspuffer

Funktion:

Mit CALL 53 werden die Worte 0 bis 7 der CPU-Ausgangsdatentafel in die Worte 200 bis 207 des BASIC-Eingangspuffers übertragen. Diese Routine verfügt über kein Eingangsargument und ein Ausgangsargument. Das Ausgangsargument kennzeichnet den Status des Logikprozessors, der einem der folgenden Werte entspricht:

- 0 = Logikprozessor im Run-Modus
- 1 = Logikprozessor nicht im Run-Modus

Während der Übertragung ist die Wortintegrität, jedoch nicht die Fileintegrität gewährleistet. Die Fileintegrität kann im Anwendungsprogramm mit Hilfe der Handshake-Bits sichergestellt werden.

Alle vom SLC-500-Prozessor an das BASIC-Modul übertragenen Daten müssen durch den Eingangspuffer des BASIC-Moduls geleitet werden. Die Adressen des BASIC-Eingangspuffers sind in Tabelle 13.D aufgeführt.

Tabelle 13.D
Adressen des BASIC-Eingangspuffers

Adresse	Definition
0 bis 39	Datenübertragung vom gemeinsamen DH-485-Schnittstellenfile
40 bis 99	reserviert
100 bis 163	Datenübertragung vom M0-File des CPU-Moduls SLC 500
164 bis 199	reserviert
200 bis 207	Datenübertragung von der CPU-Ausgangsdatentafel des SLC 500

Syntax:

CALL 53
POP [Prozessorstatus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>30 CALL 53 : REM XFER CPU OUTPUT IMAGE TO BASIC INPUT
    BUFFER
>40 POP X : REM LOGIC PROCESSOR STATUS
>50 IF (X<>0) THEN PRINT “PROCESSOR NOT IN RUN MODE”

READY
>RUN
```

CALL 56 – Übertragung des CPU-M0-Files in den BASIC-Eingangspuffer

Funktion:

Mit CALL 56 können bis zu 64 Worte, beginnend mit Wort 0, aus dem CPU-M0-File in den Eingangspuffer des BASIC-Moduls, beginnend mit Wort 100, übertragen werden. Diese Routine verfügt über ein Eingangs- und ein Ausgangsargument. Das Eingangsargument ist die Anzahl der zu übertragenden Worte (0 bis 64). Liegt der Wert nicht im Bereich zwischen 0 und 64, findet keine Übertragung statt, und als Ausgangsargument wird der Wert 10 angezeigt. Bei einem gültigen Eingangsargument kennzeichnet das Ausgangsargument den Status des Logikprozessors, der einem der folgenden Werte entspricht:

- 0 = erfolgreiche Übertragung, Logikprozessor im Run-Modus
- 1 = erfolgreiche Übertragung, Logikprozessor im Program-Modus
- 2 = erfolgreiche Übertragung, Logikprozessor im Test-Modus
- 10 = Angabe einer ungültigen Länge
- 11 = Logikprozessor unterstützt diese Funktion nicht

Während der Übertragung ist die Wortintegrität, jedoch nicht die Fileintegrität gewährleistet. Die Fileintegrität kann im Anwendungsprogramm mit Hilfe der Handshake-Bits sichergestellt werden.

Syntax:

PUSH [Anzahl der zu übertragenden Worte]
CALL 56
POP [Prozessorstatus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 64 :  REM TRANSFER 64 WORDS
>40 CALL 56 :  REM TRANSFER M0 TO BASIC INPUT BUFFER
>50 POP X :  REM LOGIC PROCESSOR STATUS IS IN X
>60 IF (X=10) PRINT “ILLEGAL INPUT ARGUMENT”
>70 IF (X<>0).AND.(X<>10) THEN  PRINT “PROCESSOR NOT IN RUN
    MODE”

READY
>RUN
```

CALL 84 – Übertragung des DH-485-Schnittstellenfiles in den BASIC-Eingangspuffer

Funktion:

Mit CALL 84 können ab dem spezifizierten Versatz des DH485-Schnittstellenfiles bis zu 40 Worte in den Eingangspuffer des BASIC-Moduls, beginnend am selben spezifizierten Versatz von Wort 0, übertragen werden. Diese Routine verfügt über zwei Eingangsargumente und ein Ausgangsargument. Das erste Eingangsargument ist der beginnende Versatz im DH-485-Schnittstellenfile und im Eingangspuffer des BASIC-Moduls (0 bis 39). Liegt der Wert nicht im Bereich von 0 bis 39, wird als Ausgangsargument der Wert 1 angezeigt, und die Übertragung findet nicht statt. Das zweite Eingangsargument ist die Anzahl der zu übertragenden Worte (1 bis 40). Liegt die Wortanzahl nicht im Bereich von 1 bis 40, wird als Ausgangsargument der Wert 2 ausgegeben, und die Übertragung findet nicht statt.

- 0 = Übertragung erfolgreich
- 1 = ungültiger beginnender Versatz
- 2 = ungültige Länge

Während der Übertragung ist die Wortintegrität, jedoch nicht die Fileintegrität gewährleistet. Die Fileintegrität kann im Anwendungsprogramm mit Hilfe der Handshake-Bits sichergestellt werden.

Syntax:

PUSH [Versatz des beginnenden Wortes im DH-485-Schnittstellenfile]
PUSH [Anzahl der zu übertragenden Worte]
CALL 84
POP [Übertragungsstatus]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 0 : REM OFFSET ADDRESS = 0
>50 PUSH 32 : REM WORD OFFSET = 32
>60 CALL 84 : REM TRANSFER THE DATA TO THE BASIC INPUT
    BUFFER
>70 POP R : REM GET THE OUTPUT ARGUMENT
>80 IF (R<>0) THEN PRINT “TRANSFER ERROR CODE = ”,R : REM
    PRINT ERROR

READY
>RUN

READY
>
```

CALL 90 – Lesetransfer des dezentralen DH-485-Datenfiles in den BASIC-Eingangspuffer

Funktion:

Mit CALL 90 können von einer definierten Netzknotenadresse, einer Filenummer, einem definierten Filetyp und einem Elementversatz bis zu 40 Worte eines dezentralen DH-485-Datenfiles abgelesen und beginnend bei Wort 0 in den Eingangspuffer des BASIC-Moduls übertragen werden. Diese Routine verfügt über sechs Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist die Netzknotenadresse des dezentralen Gerätes (0 bis 31). Liegt die Adresse nicht im Bereich zwischen 0 und 31, ist das Ausgangsargument 10, und der Lesetransfer findet nicht statt.

Das zweite Eingangsargument ist die Filenummer des dezentralen Gerätes (0 bis 255). Liegt die Nummer nicht zwischen 0 und 255, wird als Ausgangsargument der Wert 11 ausgegeben, und der Lesetransfer findet nicht statt.

Das dritte Eingangsargument ist der vom dezentralen Gerät abzulesende Filetyp. Die gültigen Filetyp-Codes lauten: ASC(N), ASC(S), ASC(C), ASC(T), ASC(B) und ASC(R). Wenn der Filetyp nicht einem dieser gültigen Typen entspricht, wird als Ausgangsargument der Wert 241 angegeben, und der Lesetransfer findet nicht statt.

Tabelle 13.E
Typ des vom dezentralen Gerät abzulesenden Files

Filetyp	Filetyp-Code	Worte/Element
Ganzzahlfile	ASC(N)	1 Wort/Element
Statusfile	ASC(S)	1 Wort/Element
Zäherfile	ASC(C)	3 Worte/Element
Zeitwerkfile	ASC(T)	3 Worte/Element
Bitfile	ASC(B)	1 Worte/Element
Steuerfile	ASC(R)	3 Worte/Element

Das vierte Eingangsargument ist der Versatz des beginnenden Elementes im File des dezentralen Gerätes (0 bis 32767). Wenn der Wert außerhalb des gültigen Bereichs liegt, ist das Ausgangsargument 12, und die Übertragung wird nicht ausgeführt.

Wichtig: Der Versatz ist zweimal so groß wie erwartet. Beispiel: Wenn der Versatz 3 mit der PUSH-Anweisung in den Stapelspeicher übertragen wird, werden die Daten beginnend mit Element 6 an den dezentralen DH-485-Datenfile übertragen.

Das fünfte Eingangsargument ist die Anzahl der zu übertragenden Elemente. Wenn die Anzahl nicht im gültigen Bereich (siehe Tabelle 13.F) liegt, wird als Ausgangsargument der Wert 13 angezeigt, und die Übertragung findet nicht statt.

Tabelle 13.F
Gültiger Längenbereich

Filetyp-Code	Gültiger Längenbereich
ASC(N)	1 bis 40
ASC(S)	1 bis 40
ASC(C)	1 bis 13
ASC(T)	1 bis 13
ASC(B)	1 bis 40
ASC(R)	1 bis 13

Das sechste Eingangsargument ist der Zeitablaufwert der Nachricht. Dieser Wert ist die Zeit (x 100 ms), die bis zum Erhalt der Leseantwort verstreichen kann (1 bis 50 = 0,1 bis 5,0 Sekunden). Geht die Leseantwort innerhalb dieser Zeit nicht ein, wird die Nachricht abgebrochen, und als Ausgangsargument wird der Wert 55 angezeigt. Wenn der Wert nicht zwischen 1 und 50 liegt, wird als Ausgangsargument der Wert 14 angezeigt, und die Übertragung findet nicht statt.

Die Lesedaten vom dezentralen Gerät werden in den Eingangspuffer des BASIC-Moduls, beginnend mit Wort 0, übertragen, wobei die Anzahl der übertragenen Worte von der Elementlänge der Nachricht abhängt.

Das Ausgangsargument spezifiziert den Status des Nachrichtenbefehls. Nach Ausführung des Aufrufs wird das Ausgangsargument wie folgt definiert:

Tabelle 13.G
Ausgangsargument

Dezimale Ausgabe	Hexadezimle Ausgabe	Beschreibung
0	00	Übertragung erfolgreich abgeschlossen
2	02	Zielnetzknoten kann Nachricht zum derzeitigen Zeitpunkt nicht annehmen
3	03	Zielnetzknoten kann nicht antworten, da Nachricht zu lang ist
4	04	Zielnetzknoten kann nicht antworten, weil er die Befehlsparameter nicht verarbeiten kann
5	05	BASIC-Modul ist offline (nicht an den Verbund angeschlossen)
6	06	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion nicht verfügbar ist

Dezimale Ausgabe	Hexadezimle Ausgabe	Beschreibung
7	07	Zielnetzknoten antwortet nicht
10	0A	BASIC-Modul stellt eine ungültige Zielnetzknotenadresse fest
11	0B	BASIC-Modul stellt eine ungültige Filenummer fest
12	0C	BASIC-Modul stellt einen ungültigen Elementversatz im Zielfile fest
13	0D	BASIC-Modul stellt eine ungültige Zielfilelänge fest
14	0E	BASIC-Modul stellt einen ungültigen Zeitablaufwert fest
16	10	Zielnetzknoten kann aufgrund falscher Befehlsparameter oder eines nicht unterstützten Befehls nicht antworten
55	37	Zeitablauf der Nachricht ist eingetreten (Zeitablaufwert wurde überschritten)
80	50	kein verfügbarer Speicherbereich im Zielnetzknoten
96	60	Zielnetzknoten kann nicht antworten, weil File geschützt ist
231	E7	Zielnetzknoten kann nicht antworten, weil die angeforderte Länge zu groß ist
235	EB	Zielnetzknoten kann nicht antworten, weil er den Zugriff verweigert
236	EC	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion derzeit nicht verfügbar ist
241	F1	BASIC-Modul stellt einen ungültigen Zielfiletyp fest
250	FA	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige File-Zugriffsberechtigung besitzt
251	FB	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige Programm-Zugriffsberechtigung besitzt

Syntax:

PUSH [Netzknotenadresse des dezentralen Gerätes]
PUSH [Filenummer des dezentralen Gerätes]
PUSH [Filetyp des dezentralen Gerätes]
PUSH [Versatz des beginnenden Elementes (x2) im File des dezentralen Gerätes]
PUSH [Anzahl der zu übertragenden Elemente]
PUSH [Zeitablaufwert der Nachricht]
CALL 90
POP [Status des Nachrichtenbefehls]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>20 PUSH 5 : REM REMOTE FILE   5
>30 PUSH ASC(C) : REM FILE TYPE = COUNTER
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM ELEMENT LENGTH = 10 = 30 WORDS
>60 PUSH 5 : REM TIMEOUT = 0.5 SECONDS
>70 CALL 90
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF (R<>0) THEN PRINT “READ ERROR CODE =”,R

READY
>RUN

READ ERROR CODE = 5
```

CALL 92 – Lesetransfer vom dezentralen gemeinsamen DH-485-Schnittstellenfiles in den BASIC-Eingangspuffer

Funktion:

Mit CALL 92 können, beginnend am spezifizierten Wortversatz, bis zu 40 Worte des dezentralen gemeinsamen DH-485-Schnittstellenfiles der spezifizierten Netzknotenadresse abgelesen und beginnend mit Wort 0 in den Eingangspuffer des BASIC-Moduls übertragen werden. Diese Routine verfügt über vier Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist die Netzknotenadresse des dezentralen Gerätes (1 bis 31). Liegt die Adresse nicht zwischen 1 und 31, ist das Ausgangsargument 10, und der Lesetransfer findet nicht statt.

Das zweite Eingangsargument ist der Versatz des beginnenden Wortes im File des dezentralen Gerätes (0 bis 32767). Wenn dieser Wert außerhalb des gültigen Bereichs liegt, wird als Ausgangsargument der Wert 12 ausgegeben, und die Übertragung wird nicht ausgeführt.

Wichtig: Der Versatz ist zweimal so groß wie erwartet. Beispiel: Wenn der Versatz 3 mit der PUSH-Anweisung in den Stapelspeicher eingespeichert wird, werden die Daten beginnend bei Element 6 in den dezentralen DH-485-Datenfile übertragen.

Das dritte Eingangsargument ist die Anzahl der zu übertragenden Worte. Wenn der Wert nicht innerhalb des gültigen Bereichs von 1 bis 40 liegt, wird als Ausgangsargument der Wert 13 angezeigt, und die Übertragung wird nicht ausgeführt.

Das vierte Eingangsargument ist der Zeitablaufwert der Nachricht. Dieser Wert ist die Zeit (x 100 ms), die bis zum Erhalt der Leseantwort verstreichen kann (1 bis 50 = 0,1 bis 5,0 Sekunden). Geht die Leseantwort nicht innerhalb dieser Zeit ein, wird die Nachricht abgebrochen, und als Ausgangsargument wird der Wert 55 angezeigt. Wenn der Zeitwert nicht zwischen 1 und 50 liegt, wird als Ausgangsargument der Wert 14 angezeigt, und die Übertragung findet nicht statt.

Die Lesedaten vom dezentralen Gerät werden beginnend mit Wort 0 in den Eingangspuffer des BASIC-Moduls eingelesen, wobei die Anzahl der übertragenen Worte von der Wortlänge der Nachricht abhängt.

Das Ausgangsargument spezifiziert den Status des Nachrichtenbefehls. Nach Ausführung des Aufrufs wird das Ausgangsargument wie folgt definiert:

Tabelle 13.H
Ausgangsargument

Dezimale Ausgabe	Hexadezimale Ausgabe	Beschreibung
0	00	Übertragung erfolgreich abgeschlossen
2	02	Zielnetzknoten kann Nachricht zum derzeitigen Zeitpunkt nicht annehmen
3	03	Zielnetzknoten kann nicht antworten, da Nachricht zu lang ist
4	04	Zielnetzknoten kann nicht antworten, weil er die Befehlsparameter nicht verarbeiten kann
5	05	BASIC-Modul ist offline (nicht an den Verbund angeschlossen)
6	06	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion nicht verfügbar ist
7	07	Zielnetzknoten antwortet nicht
10	0A	BASIC-Modul stellt eine ungültige Zielnetzknotenadresse fest
11	0B	BASIC-Modul stellt eine ungültige Filenummer fest
12	0C	BASIC-Modul stellt einen ungültigen Elementversatz im Zielfile fest
13	0D	BASIC-Modul stellt eine ungültige Zielfilelänge fest
14	0E	BASIC-Modul stellt einen ungültigen Zeitablaufwert fest
16	10	Zielnetzknoten kann aufgrund falscher Befehlsparameter oder eines nicht unterstützten Befehls nicht antworten
55	37	Zeitablauf der Nachricht ist eingetreten (Zeitablaufwert wurde überschritten)
80	50	kein verfügbarer Speicherbereich im Zielnetzknoten
96	60	Zielnetzknoten kann nicht antworten, weil File geschützt ist
231	E7	Zielnetzknoten kann nicht antworten, weil die angeforderte Länge zu groß ist
235	EB	Zielnetzknoten kann nicht antworten, weil er den Zugriff verweigert

Dezimale Ausgabe	Hexadezimale Ausgabe	Beschreibung
236	EC	Zielnetzknoten kann nicht antworten, weil die angeforderte Funktion derzeit nicht verfügbar ist
241	F1	BASIC-Modul stellt einen ungültigen Zielfiletyp fest
250	FA	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige File-Zugriffsberechtigung besitzt
251	FB	Zielnetzknoten kann nicht antworten, weil ein anderer Netzknoten die alleinige Programm-Zugriffsberechtigung besitzt

Syntax:

PUSH [Netzknotenadresse des dezentralen Gerätes]
PUSH [Versatz des beginnenden Elementes (x2) im File des dezentralen Gerätes]
PUSH [Anzahl der zu übertragenden Worte]
PUSH [Zeitablaufwert der Nachricht]
CALL 92
POP [Status des Nachrichtenbefehls]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM WORD LENGTH = 10
>60 PUSH 5   REM TIME-OUT VALUE = 0.5 SECONDS
>70 CALL 92
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF (R<>0) THEN PRINT "READ ERROR CODE IS",R : REM PRINT
    ERROR

READY
>RUN

READ ERROR CODE IS 5
```

CALL 117 – Abruf der DF1-Datenpaketlänge

Funktion:

Mit CALL 117 wird die Länge des DF1-Datenpakets abgerufen. Diese Routine verfügt über kein Eingangs- und ein Ausgangsargument. Das Ausgangsargument gibt die Länge des ältesten Datenpaketes an, das in der Warteschlange des DF1-Empfangspuffers enthalten ist.

Wichtig: Wenn der Empfangspuffer leer ist, wird der Wert 0000 in den Argument-Stapel übertragen.

Werden mit CALL 117 Daten in ein Programm eingelesen, überprüft die BASIC-Baugruppe, ob die DF1-Kommunikation mit CALL 108 aktiviert wurde. Ist dies nicht der Fall, wird auf dem Terminal eine Fehlermeldung ausgegeben, und das BASIC-Modul schaltet in den Befehlsmodus um.

Nach Erhalt der Länge des DF1-Datenpakets muß die GET-Anweisung programmiert werden, um die Daten des empfangenen DF1-Datenpakets einzulesen.

Syntax:

CALL 117
POP [Länge des DF1-Datenpakets]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 117
>20 POP X
>30 END
```

CALL 118 – Freilaufende PLC/SLC-Schreibtransfers

Funktion:

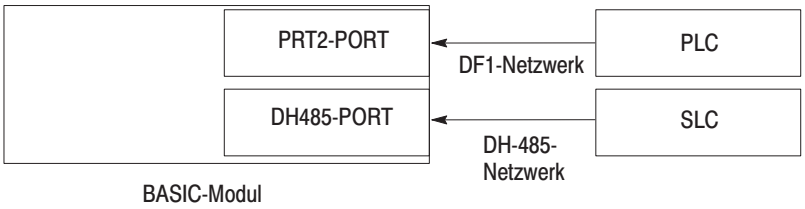
Mit CALL 118 kann das BASIC-Modul Datenpakete empfangen, die mit PLC-2-, PLC-3- oder PLC-5-Nachrichtenbefehlen über das DF1-Netzwerk gesendet wurden. Ferner ermöglicht dieser Aufruf den Empfang von Datenpaketen eines SLC-Netzknotens im DH-485-Netzwerk. Der DF1-Port (PRT2) und der DH485-Port können nicht gleichzeitig aktiviert sein. Die Portkonfiguration wird mit Brücke JW4 des BASIC-Moduls eingestellt.

Die mit Nachrichtenschreibbefehlen von diesen PLC-/SLC-Prozessoren an das BASIC-Modul gesendeten Daten werden, beginnend am spezifizierten Wortversatz, in einer internen Zeichenkette im CPU-M1-File, im CPU-Eingangsabbildfile und/oder in der Zeichenkette des BASIC-Moduls abgelegt.

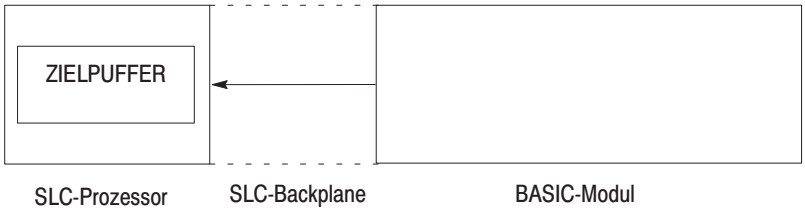
Das niederwertige Byte des ersten Wortes im Zielfile enthält die Zeichenanzahl (Byteanzahl) der übertragenen Daten. Das hochwertige Byte dieses Wortes wird nicht belegt. Wenn als Ziel eine interne Zeichenkette gewählt wird, enthält das erste Zeichen die Byteanzahl. Das zweite Zeichen (Transaktionsnummer) der internen Zeichenkette wird nach dem erfolgreichen Empfang eines Datenpakets inkrementiert, um dem BASIC-Modul zu signalisieren, daß die Zeichenkette neue Daten enthält. Der Wert dieser Transaktionsnummer geht von 255 automatisch auf 0 über.

Führen Sie CALL 118 einmal aus. Anschließend überprüft das BASIC-Modul am Ende jeder BASIC-Zeile den Port. Das BASIC-Modul empfängt neue Daten vom PLC- bzw. SLC-Prozessor und überträgt sie wie folgt:

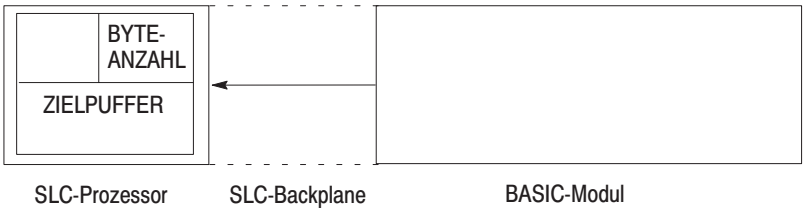
1. Das BASIC-Modul empfängt Datenpakete, die von dem im Aufruf konfigurierten PLC/SLC-Prozessor eingeleitet werden. PRT2 und DH485 können nicht gleichzeitig aktiviert sein. Diese Einstellung wird an Brücke JW4 vorgenommen.



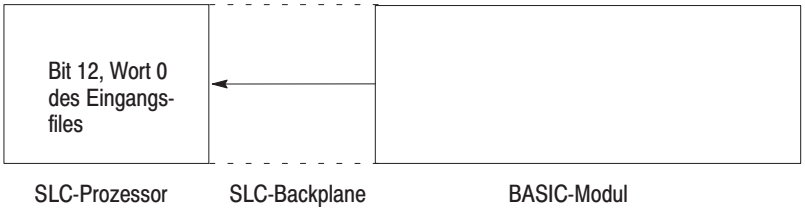
2. Das BASIC-Modul überträgt die Daten in den Zielpuffer des zentralen SLCs.



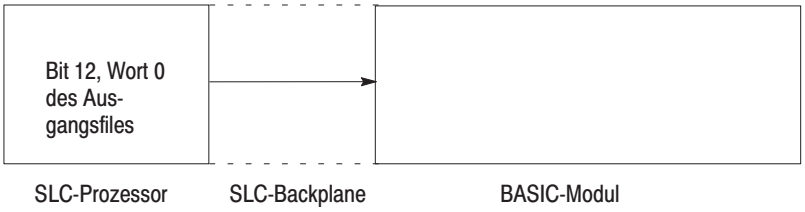
3. Das BASIC-Modul überträgt die Byteanzahl in das niederwertige Byte des ersten im Zielpuffer verfügbaren Wortes.



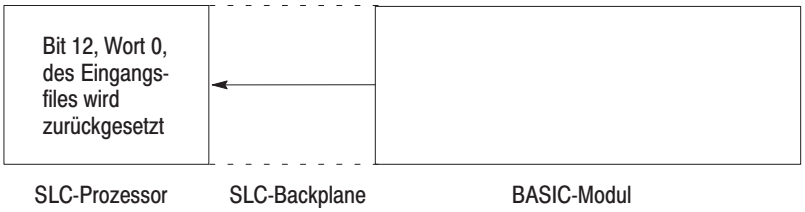
4. Das BASIC-Modul setzt Bit 12 in Wort 0 des Eingangsfiles, um dem SLC-Prozessor zu signalisieren, daß gültige Daten verfügbar sind.



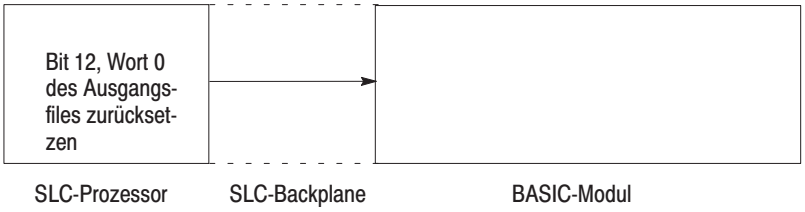
5. Der SLC-Prozessor ruft die Daten aus dem Puffer ab und setzt Bit 12 in Wort 0 des Ausgangsfiles, um dem BASIC-Modul zu signalisieren, daß die Daten empfangen wurden.



6. Das BASIC-Modul setzt Bit 12 in Wort 0 des Eingangsfiles an demselben Ende des Abfragezyklus zurück, an dem Bit 12 in Wort 0 des Ausgangsfiles zurückgesetzt wurde.



7. Der SLC-Prozessor setzt Bit 12 in Wort 0 des Ausgangsfiles zurück. Nach Empfang des nächsten Datenpaketes kann das BASIC-Modul mit dem Laden des Zielpuffers beginnen.



Der SLC-Prozessor darf Bit 12 in Wort 0 des Ausgangsfiles nicht in derselben Abfrage der Strompfadlogik setzen und zurücksetzen. Anderenfalls empfängt das BASIC-Modul möglicherweise das übertragene Bit nicht.

Dieser Aufruf bleibt aktiv, bis er mit anderen Eingangsparametern erneut ausgeführt wird. Ist dies der Fall, wird der vorhergehende CALL 118 automatisch deaktiviert, und der neue CALL 118 wird wirksam. Mehrere Aufrufe 118 werden nicht gleichzeitig ausgeführt.

Dieser Aufruf verfügt über fünf Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument aktiviert bzw. deaktiviert den Aufruf:

- 0 = Deaktivierung des zuvor ausgeführten Aufrufs 118
- 1 = Aktivierung des Aufrufs. Die folgenden Befehle sind zulässig:

  PLC (nicht geschützte Schreibtransfers)
  PLC (Wortbereich-Schreibtransfers)
  PLC (eingegebene Schreibtransfers)
  SLC 5/02 (nicht geschützte Schreibtransfers)
  SLC 5/02 (eingegebene Schreibtransfers)

Wenn die empfangenen Daten die Länge der Zeichenkette bzw. die Größe des CPU-Files überschreiten, werden die restlichen Daten abgeschnitten.

Mit dem zweiten Eingangsargument wird der CPU-Eingangsabbild-Zielfile mit oder ohne interne Zeichenkette, der CPU-M1-File mit oder ohne interne Zeichenkette oder nur die interne Zeichenkette gewählt.

- 0 = CPU-Eingangsabbildfile
- 1 = CPU-M1-File
- 2 = interne Zeichenkette
- 3 = CPU-Eingangsabbildfile und interne Zeichenkette
- 4 = CPU-M1-File und interne Zeichenkette

Wenn als Ziel die interne Zeichenkette gewählt wird (2), signalisieren die Handshake-Bits (Bit 12, Wort 0) der Eingangs-/Ausgangsdatentafel nicht, daß Daten vom BASIC-Modul empfangen wurden. Das zweite Zeichen der Kette (Transaktionsnummer) wird nach jedem erfolgreich ausgeführten Datentransfer inkrementiert und muß im BASIC-Programm überwacht werden. Anschließend müssen die Daten vor dem Empfang des nächsten Datenpakets aus der Zeichenkette entfernt werden; anderenfalls gehen sie verloren.

Das dritte Eingangsargument ist der Wortversatz im CPU-Zielfile. Dieser Versatz zeigt auf das erste Wort, daß die Byteanzahl der gültigen übertragenen Daten enthält. Der Versatz der internen Zeichenkette ist immer 2. Die Byteanzahl wird an die Zeichenadresse 0 übertragen. Adresse 1 ist eine Transaktionsnummer, die nach jeder erfolgreichen Übertragung eines Datenpakets inkrementiert wird.

Findet der Datentransfer über den DH485-Port statt, sollte der Versatz nicht größer als hexadezimal 40 (dezimal 64) sein. Freilaufende Datenschreibpakete, die größer als 64 sind, verursachen einen Schreibtransfer an den Puffer des DH485-Programmierports und verursachen somit Funktionsstörungen. Die Datenpaketgröße kann dem Maximum des gewählten Eingangsfiles entsprechen.

Wenn der CPU-Eingangsabbildfile als Ziel gewählt wird, darf dieser Versatz nicht 0 oder 1 sein. Wort 0 ist für die Handshake-Bits und Wort 1 für die mit den Aufrufen 27, 28, 122 und 123 verwendete PLC-Transaktionsnummer reserviert. Durch die Eingabe einer 0 oder 1 wird in der POP-Anweisung ein Fehler ausgegeben (2 = ungültiger Eingangsparameter), und der Aufruf wird nicht ausgeführt.

Das vierte Eingangsargument ist die Zeichenkettennummer. Wenn mit dem zweiten Eingangsargument keine interne Zeichenkette gewählt wurde, wird der Wert dieses Eingangsarguments ignoriert, muß aber dennoch in den Stapel eingespeichert (PUSH) werden.

Das fünfte Eingangsargument ist die maximal zulässige Wortlänge des Datenpakets. Wenn das BASIC-Modul längere Datenpakete empfängt, werden diese zurückgewiesen. Die Eingabe des Wertes 0 veranlaßt, daß das BASIC-Modul Datenpakete jeder Länge akzeptiert und daß alle Datenpakete empfangen werden, sofern die maximale Länge des Zielfiles nicht überschritten wird. Daten, die diese maximale Länge überschreiten, werden abgeschnitten.

Das Ausgangsargument ist der Aufrufstatus, der folgende Werte annehmen kann:

- 0 = erfolgreich
- 1 = deaktiviert
- 2 = ungültiger Eingangsparameter
- 3 = gewählter DH485-/DF1-Port nicht aktiviert
- 4 = Zeichenkette zu klein
- 5 = Zeichenkette nicht dimensioniert

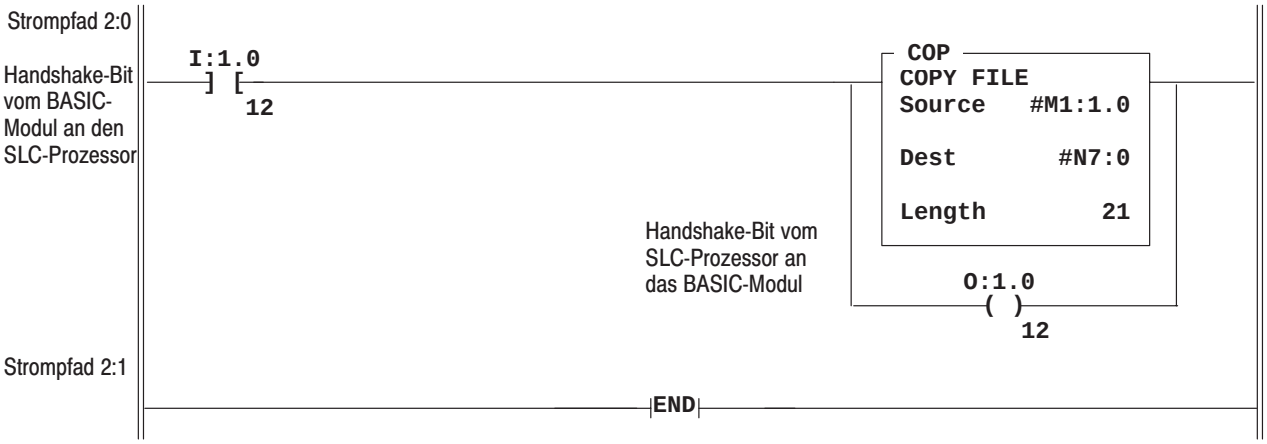
Syntax:

PUSH [Aktivierung/Deaktivierung des Aufrufs]
PUSH [Wahl des Zielfiles und/oder der Zeichenkette]
PUSH [Wortversatz im Zielfile]
PUSH [Zeichenkettennummer]
PUSH [maximale Wortlänge]
CALL 118
POP [Status von CALL 118]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE PLC/SLC UNSOLICITED WRITE INTERRUPT
>20 PUSH 1 : REM ENABLE THE CALL
>30 PUSH 1 : REM DESTINATION SLC M1 FILE
>40 PUSH 0 : REM WORD OFFSET INTO M1 FILE
>50 PUSH 0 : REM STRING NUMBER – NOT USED
>60 PUSH 20 : REM MAX ALLOWED WORD LENGTH OF DATA PACKET
>70 CALL 118
>80 POP S
>90 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 118 SETUP”
```

Im folgenden ist ein Strompfadlogikbeispiel für CALL 118 abgebildet. Das BASIC-Modul befindet sich in Steckplatz 1 des SLC-Racks. In Strompfad 2:0 werden die Daten aus dem M1-File kopiert, wenn das BASIC-Modul das Handshake-Bit (I:1.0/12) setzt. In Strompfad 2:0 wird das Handshake-Bit (O:1.0/12) gesetzt, sobald die Daten aus dem M1-File kopiert worden sind. Dadurch wird dem BASIC-Modul signalisiert, daß es Bit (I:1.0/12) zurücksetzen muß. Das erste Wort enthält die Byteanzahl. Im folgenden Beispiel werden maximal 20 Datenworte erwartet.



CALL 122 – Lesetransfer eines dezentralen DF1-Datenfiles (PLC)

Funktion:

Mit CALL 122 können bis zu 64 Datenworte eines dezentralen DF1-Netzknotens (PLC-2, -3 oder -5) in den CPU-Eingangsabbildfile, den CPU-M1-File und/oder eine Zeichenkette im BASIC-Modul eingelesen werden.

Die folgende Tabelle enthält eine Auflistung bestimmter Anmerkungen, die bei der Verwendung von CALL 122 mit einem PLC-3- bzw. PLC-5-Prozessor gelten.

Tabelle 13.I
Anmerkungen für PLC-spezifische Anwendungen

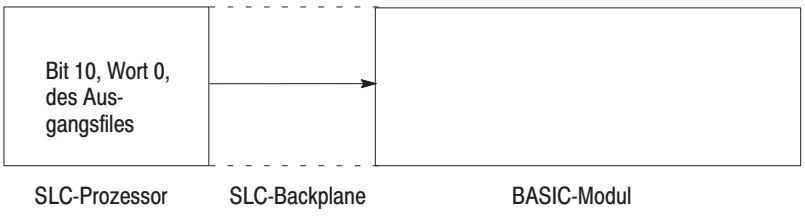
PLC	Anmerkungen
–3	Bei Zeitwerken und Zählern ist die in den Stapel eingespeicherte Filenummer (dritter Parameter) die Strukturnummer, die auf maximal 255 Worte begrenzt ist.
–5	Bei Zeitwerkdaten besteht ein Element aus drei 16-Bit-Worten, die in der folgenden Reihenfolge im Zielfile gespeichert werden: Steuerung, Sollwert und Istwert.

Bei der Wahl einer internen Zeichenkette wird das erste Zeichen (Transaktionsnummer) nach dem erfolgreich ausgeführten Lesetransfer inkrementiert, um dem BASIC-Modul zu signalisieren, daß in der Zeichenkette neue Daten enthalten sind. Der Wert der Transaktionsnummer geht von 255 automatisch wieder auf 0 über.

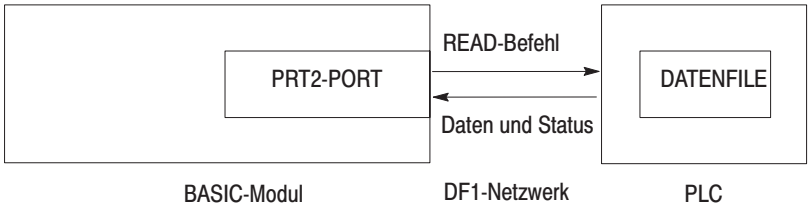
Die Parameter des DF1-Ports werden mit CALL 108 konfiguriert. Der DF1-Port ist im Vollduplex- und Halbduplex-Slaveprotokoll funktionsfähig.

Führen Sie CALL 122 einmal aus, um die Datenübertragungsparameter zu konfigurieren. Die Bits der Eingangs- und Ausgangsdatentafel (Bit 10, Wort 0) des Steckplatzes, in dem sich das BASIC-Modul befindet, leiten die Datenübertragung ein und signalisieren den Abschluß der Übertragung. Dieser Vorgang ist im folgenden näher beschrieben:

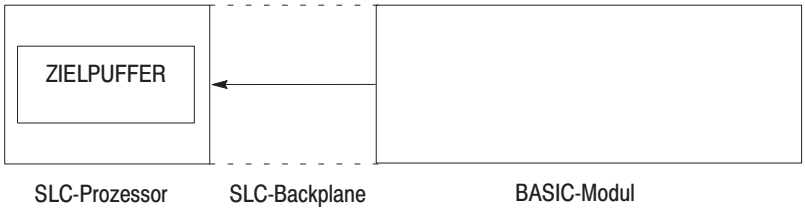
1. Der SLC-Prozessor setzt Bit 10 in Wort 0 des Ausgangsfiles, und veranlaßt damit, daß das BASIC-Modul den in diesem Aufruf konfigurierten READ-Befehl ausführt.



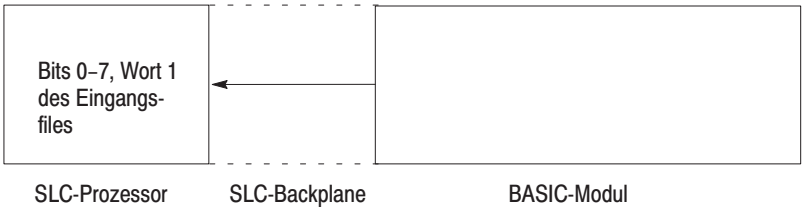
2. Das BASIC-Modul erteilt den entsprechenden READ-Befehl an den PLC-Prozessor. Die Daten und der Status werden vom PLC-Prozessor empfangen.



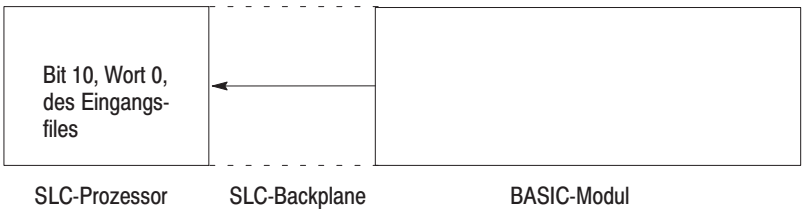
3. Wenn Daten verfügbar sind, werden sie vom BASIC-Modul in den Zielpuffer übertragen.



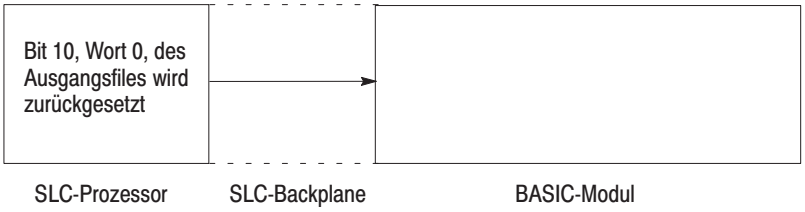
4. Das BASIC-Modul überträgt den Transaktionsstatus an die Bits 0–7 in Eingangswort 1.



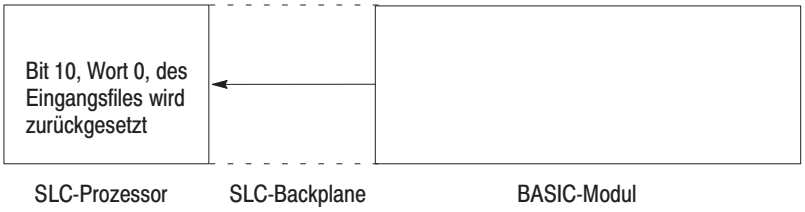
5. Das BASIC-Modul setzt Bit 10 in Wort 0 des Eingangsfiles, um dem SLC-Prozessor zu signalisieren, daß gültige Daten und der Status verfügbar sind.



6. Der SLC-Prozessor ruft die Daten und den Status aus dem Puffer ab und setzt Bit 10 in Wort 0 des Ausgangsfiles zurück, um dem BASIC-Modul zu signalisieren, daß er die Daten empfangen hat.



7. Das BASIC-Modul setzt Bit 10 in Wort 0 des Eingangsfiles an demselben Ende des Abfragezyklus zurück, in dem Bit 10 in Wort 0 des Ausgangsfiles zurückgesetzt wurde.



Der SLC-Prozessor darf Bit 10 in Wort 0 des Ausgangsfiles nicht in derselben Abfrage der Strompfadlogik setzen und zurücksetzen. Anderenfalls empfängt das BASIC-Modul möglicherweise das übertragene Bit nicht.

Dieser Aufruf bleibt aktiv, bis er mit anderen Eingangsparametern erneut ausgeführt wird.

Dieser Aufruf verfügt über zehn Eingangsargumente und ein Ausgangsargument.

Das erste Eingangsargument ist der Typ des erteilten PLC-READ-Befehls:

- 0 = Deaktivierung des zuvor ausgeführten Aufrufs
- 2 = gemeinsamer Schnittstellenfile – nicht geschützter READ-Befehl eines PLC-2-Prozessors
- 3 = PLC-3-File – Wortbereich-READ-Befehl
- 5 = PLC-5-File – eingegebener READ-Befehl

Das zweite Eingangsargument ist die Netzknotenadresse des dezentralen PLCs (0 bis 255). Wenn die Adresse nicht in diesem Bereich liegt, ist der Status gleich 2, und die Lesenachricht wird nicht übertragen.

Das dritte Eingangsargument ist die zu lesende Filenummer des dezentralen PLCs (0 bis 255). Wenn die Nummer nicht in diesem Bereich liegt, ist der Status gleich 2, und die Lesenachricht wird nicht übertragen. Der Parameter wird ignoriert, wenn im ersten Eingangsparameter der gemeinsame Schnittstellenfile gewählt wurde, muß aber dennoch mit einer PUSH-Anweisung in den Stapel übertragen werden.

Das vierte Eingangsargument ist der vom dezentralen PLC abzulesende Filetyp. Geben Sie den Filecode entsprechend der folgenden Tabelle ein. Dieses Argument wird ignoriert, wenn im ersten Parameter der gemeinsame Schnittstellenfile gewählt wurde, muß aber dennoch mit einer PUSH-Anweisung (setzt Ganzzahltyp voraus) in den Stapel eingespeichert werden. Wenn ein anderer, nicht in dieser Tabelle enthaltener Filetyp eingegeben wird, ist der Status gleich 2, und die Lesenachricht wird nicht übertragen.

Tabelle 13.J
Typ des vom dezentralen Gerät abzulesenden Files

Filetyp	Filetyp-Code	Worte/Element (1 Wort = 16 Bits)
Ganzzahlfile	ASC(N)	1 Wort/Element
Statusfile	ASC(S)	1 Wort/Element
Zählerfile	ASC(C)	3 Worte/Element
Zeitwerkfile	ASC(T)	3 Worte/Element
Bitfile	ASC(B)	1 Wort/Element
Steuerfile	ASC(R)	3 Worte/Element
Eingangsfile	ASC(I)	1 Wort/Element
Ausgangsfile	ASC(O)	1 Wort/Element

Das fünfte Eingangsargument ist der Versatz des beginnenden Wortes im File des dezentralen PLC-2-Prozessors (0 bis 32766). Bei PLC-3-Ganzzahl-, -Binär- und -Statusfiles ist der Wert 0–9999. Bei PLC-3-E/A-Files ist der Wert 0–4095. Bei PLC-3-Zeitwerk- und Zählerfiles muß der Wert 0 sein. Wenn der Wert nicht im gültigen Bereich liegt, ist der Status gleich 2, und die Übertragung findet nicht statt.

Das sechste Eingangsargument ist die Anzahl der zu übertragenden Elemente. Wenn die Anzahl nicht im jeweiligen Bereich (siehe Tabelle) liegt, ist der Status gleich 2, und die Übertragung wird nicht ausgeführt.

Tabelle 13.K
Gültige Bereiche der Elementlängen

Filetyp-Code	Gültige Bereiche der Elementlängen
ASC(N)	1 bis 64
ASC(S)	1 bis 64
ASC(C)	1 bis 21
ASC(T)	1 bis 21
ASC(B)	1 bis 64
ASC(R)	1 bis 21
ASC(I)	1 bis 21
ASC(O)	1 bis 21
gemeinsamer Schnittstellenfile	1 bis 21

Das siebte Eingangsargument ist der Nachrichten-Zeitablaufwert. Dieser Wert (1 bis 255) entspricht der Zeit (x 100 ms), die verstreichen kann, bis die Leseantwort erhalten wird (0,1 bis 25,5 Sekunden). Geht die Leseantwort nicht innerhalb dieser Zeit ein, wird die Nachricht abgebrochen, und als Status wird der Wert 55 in Wort 1 des Eingangsfiles angezeigt. Wenn der Zeitablaufwert nicht innerhalb des gültigen Bereichs (1 bis 255) liegt, wird als Statusausgangsargument der Wert 2 angezeigt, und die Übertragung findet nicht statt.

Mit dem achten Eingangsargument wird der CPU-Eingangsabildfile des Ziels mit oder ohne Zeichenkette, der CPU-M1-File mit oder ohne Zeichenkette oder nur die interne Zeichenkette gewählt:

- 0 = CPU-Eingangsabbildfile
- 1 = CPU-M1-File
- 2 = interne Zeichenkette
- 3 = CPU-Eingangsabbildfile und interne Zeichenkette
- 4 = CPU-M1-File und interne Zeichenkette

Wenn Sie die interne Zeichenkette (2) wählen, kann CALL 29 ausgeführt werden, um die einzelnen Datentransfers ohne die Interaktion eines SLC-Prozessors einzuleiten. Zeichenkettentransaktionen werden auch mit Bit 10 in Wort 0 des Ausgangsfiles eingeleitet.

Das neunte Eingangsargument ist der Wortversatz im CPU-Zielfile. Dieser Versatz zeigt auf das erste übertragene Datenwort. Der Versatz der internen Zeichenkette ist immer 1. Das erste Zeichen (Transaktionsnummer der Adresse 0) wird nach jedem erfolgreichen Datentransfer inkrementiert, um dem BASIC-Modul zu signalisieren, daß in der Zeichenkette neue Daten enthalten sind. Der Wert der Transaktionsnummer geht von 255 automatisch wieder auf 0 über.

Wenn der CPU-Eingangsabbildfile gewählt wird, darf der Versatz nicht 0 oder 1 sein, da diese beiden Worte reserviert sind. Wort 0 ist für die Handshake-Bits des Datentransfers und Wort 1 für den Transaktionsstatus reserviert. Durch die Eingabe einer 0 oder 1 wird in der POP-Anweisung ein Fehler ausgegeben (2 = ungültiger Eingangsparameter), und der Aufruf wird nicht ausgeführt.

Wenn die empfangenen Daten die Zeichenkettenlänge bzw. die Größe des CPU-Files überschreiten, werden die restlichen Daten abgeschnitten.

Das zehnte Eingangsargument ist die Zeichenkettennummer. Wenn im achten Eingangsargument nicht die interne Zeichenkette als Ziel gewählt wurde, wird der Wert dieses Eingangsarguments ignoriert, muß aber trotzdem mit einer PUSH-Anweisung in den Stapelspeicher eingespeichert werden.

Das Ausgangsargument ist der Status des Aufrufs und kann die folgenden Werte annehmen:

- 0 = erfolgreich
- 1 = deaktiviert
- 2 = ungültiger Eingangsparameter
- 3 = DF1 nicht aktiviert
- 4 = Zeichenkette zu klein
- 5 = Zeichenkette nicht dimensioniert

Zur Deaktivierung dieses Aufrufs muß mit einer PUSH-Anweisung eine Null in den ersten Eingangsparameter übertragen werden. Alle weiteren Parameter werden ignoriert, müssen aber trotzdem mit einer PUSH-Anweisung in den Stapel übertragen werden.

Bei jedem Versuch, einen dezentralen Netzknoten abzulesen, wird der Status des Lesetransfers in die Bits 0–7 des Eingangswortes 1 abgelegt. Die möglichen Statuscodes sind in Tabelle 13.L aufgeführt.

Der Status ist gültig, wenn das BASIC-Modul Bit 10 in Wort 0 des Eingangsfiles setzt.

Tabelle 13.L
Transaktionsstatuscodes

Code	Bedeutung
0	Übertragung erfolgreich
1	Übertragung nicht erfolgreich
2	Aufforderungszeitablauf
3	Handshake-Modus gewählt – entweder ist eine Unterbrechung des CTS-Signals während der Übertragung oder eine nicht behebbare Senderstörung eingetreten
	Handshake-Modus abgewählt – eine nicht behebbare Senderstörung ist eingetreten
4	Bei Modem-Handshake-Quittierung mit Dauerträger eine Übertragungsstörung, die auf eine Unterbrechung des Modems (d.h. einen DCD-Signalverlust von mehr als 10 Sekunden) zurückzuführen ist
5	DF1-Treiber ist nicht aktiviert
6	Nachrichtenzeitablauf
81	unzulässiger Befehl bzw. unzulässiges Format
82	eine am Host vorliegende Störung verhindert die Kommunikation
83	Station des dezentralen Hosts nicht anwesend, nicht angeschlossen oder ausgeschaltet
84	Host kann Funktion aufgrund einer Hardwarestörung nicht durchführen
85	Adressierungsproblem oder speichergeschützte Strompfade
86	Funktion kann nicht ausgeführt werden, weil der Befehlsschutz gewählt wurde
87	Prozessor ist in den Programmiermodus geschaltet
88	Kompatibilitätsfile fehlt oder es liegt ein Problem bezüglich der Kommunikationszone vor
89	dezentrale Station kann Befehl nicht in den Puffer einspeichern
8B	Herunterladevorgang verursachte Störung an der dezentralen Station
8C	zentrale Station kann Befehl aufgrund aktivierter IPBs nicht ausführen
C1	unzulässiges Adreßformat – Datenfeld enthält einen unzulässigen Wert
C2	unzulässiges Adreßformat – unzureichende Anzahl von Datenfeldern spezifiziert
C3	unzulässiges Adreßformat – zu hohe Anzahl von Datenfeldern spezifiziert
C4	unzulässiges Adreßformat – Symbol kann nicht gefunden werden

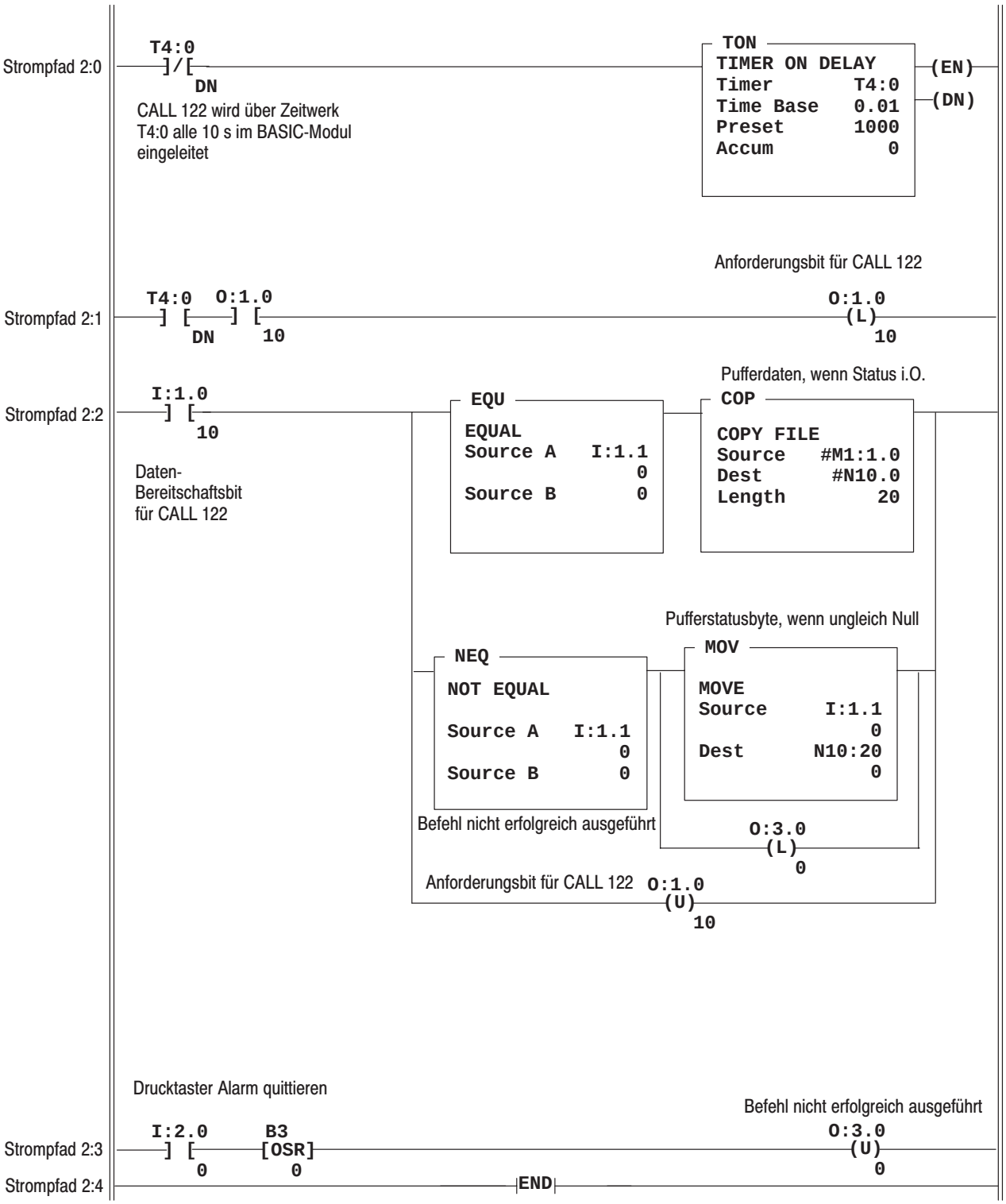
Code	Bedeutung
C5	unzulässiges Adreßformat – Symbol ist 0 oder größer als die von diesem Gerät unterstützte maximale Zeichenanzahl
C6	unzulässige Adresse – Adresse nicht vorhanden oder Adresse zeigt nicht auf einen nutzbaren Wert in diesem Befehl
C7	unzulässige Größe – Filegröße ist falsch; Adresse überschreitet Fileende
C8	Aufforderung kann nicht vollständig ausgeführt werden
C9	Daten bzw. File zu groß
CA	Aufforderung zu groß; Transaktionsgröße plus Wortadresse zu groß
CB	Zugriff verweigert, Privilegverletzung
CC	Ressourcen nicht verfügbar; Zustand kann nicht erzeugt werden
CD	Ressourcen bereits verfügbar; Zustand bereits vorhanden
CE	Befehl kann nicht ausgeführt werden
CF	Überlauf; Histogrammüberlauf
D0	kein Zugriff
D1	unzulässiger Datentyp
D2	ungültiger Parameter; ungültige Daten im Such- oder Befehlsblock
D3	Adresse bezieht sich auf gelöschten Bereich
D4	Befehlsausführung aus unbekanntem Grund nicht erfolgreich; PLC-3-Histogrammüberlauf
D5	Datenumwandlungsfehler
D6	Scanner kann nicht mit einem 1771-Chassisadapter kommunizieren
D7	Adapter kann nicht mit Modul kommunizieren
D8	Antwort des 1771-Moduls war ungültig
D9	duplizierte Label-Kennzeichnung
DA	File ist geöffnet – eine andere Station hat die Zugriffsberechtigung
DB	eine andere Station hat die Zugriffsberechtigung auf das Programm

Syntax:

PUSH [Typ des PLC-READ-Befehls]
PUSH [Netzknotenadresse des dezentralen PLCs]
PUSH [Filenummer des dezentralen PLCs]
PUSH [Filetyp des dezentralen PLCs]
PUSH [Versatz des beginnenden Elementes im dezentralen PLC]
PUSH [Anzahl der zu übertragenden Elemente]
PUSH [Zeitablaufwert der Nachricht]
PUSH [Wahl des Zielfiles]
PUSH [Wortversatz im Zielfile]
PUSH [Zeichenkettennummer]
CALL 122
POP [Status von CALL 122]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PLC REMOTE READ COMMAND
>20  PUSH 5 : REM PLC–5 FILE
>30  PUSH 0 : REM NODE ADDRESS OF PLC–5
>40  PUSH 7 : REM FILE NUMBER OF PLC–5
>50  PUSH ASC(N) : REM FILE TYPE OF PLC–5
>60  PUSH 0 : REM STARTING WORD OFFSET OF PLC–5 FILE
>70  PUSH 20 : REM NUMBER OF DATA WORDS TO READ
>80  PUSH 10 : REM COMMAND TIME–OUT VALUE (X100MS)
>90  PUSH 1 : REM DESTINATION IS SLC M1 FILE
>100 PUSH 0 : REM WORD OFFSET WITHIN M1 FILE
>110  PUSH  0  :  REM  STRING  NUMBER  –  NOT  USED  FOR  THIS
EXAMPLE
>120 CALL 122
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 122 SETUP”
```


Strompfad 2:0
T4:0
]/[
DN
CALL 122 wird über Zeitwerk T4:0 alle 10 s im BASIC-Modul eingeleitet
TON
TIMER ON DELAY
Timer T4:0
Time Base 0.01
Preset 1000
Accum 0
(EN)
(DN)
Anforderungsbit für CALL 122
Strompfad 2:1
T4:0 O:1.0
] [ ] [
DN 10
O:1.0
(L)
10
Pufferdaten, wenn Status i.O.
Strompfad 2:2
I:1.0
] [
10
Daten-Bereitschaftsbit für CALL 122
EQU
EQUAL
Source A I:1.1
0
Source B 0
COP
COPY FILE
Source #M1:1.0
Dest #N10.0
Length 20
Pufferstatusbyte, wenn ungleich Null
NEQ
NOT EQUAL
Source A I:1.1
0
Source B 0
MOV
MOVE
Source I:1.1
0
Dest N10:20
0
Befehl nicht erfolgreich ausgeführt
O:3.0
(L)
0
Anforderungsbit für CALL 122
O:1.0
(U)
10
Drucktaster Alarm quittieren
Befehl nicht erfolgreich ausgeführt
Strompfad 2:3
I:2.0 B3
] [[OSR]
0 0
O:3.0
(U)
0
Strompfad 2:4
END

GET

Funktion:

Die GET-Anweisung wird im Run-Modus angewandt. Im Befehlsmodus wird bei dieser Anweisung das Ergebnis 0 ausgegeben. Der GET-Operator liest den Terminaleingang ab. Wenn von dort ein Zeichen verfügbar ist, wird der GET-Anweisung der Wert des Zeichens zugeordnet. Nach Lesen der GET-Anweisung im Programm wird ihr der Wert 0 zugeordnet, bis vom Terminal ein weiteres Zeichen gesendet wird.

Verwenden Sie zum Ablesen des PRT2-Ports den Operator GET# und zum Ablesen des PRT1-Ports den Operator GET@. Im folgenden Beispiel wird ein vom Terminal empfangenes Zeichen in Dezimaldarstellung ausgegeben.

Syntax:

GET

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 A = GET
>20 IF (A<>0) THEN PRINT A : REM ZERO MEANS NO ENTRY
>30 GOTO 10
>RUN

 65 [A]
 49 [1]
 24 [^X]
 50 [2]

STOP – IN LINE  30
READY
>
```

Der GET-Operator kann nur einmal gelesen werden, bevor ihm der Wert Null zugewiesen wird. Dadurch wird sichergestellt, daß immer das erste eingegebene Zeichen gelesen wird, unabhängig von der Positionierung des GET-Operators im Programm. Am Programmierport werden keine Zeichen gepuffert.

INPL

Funktion:

Mit der INPL-Anweisung wird die vollständige Zeile (bis zu 254 Zeichen) vom Puffer des Programmierports abgelesen. Die Zeile muß in einer Zeichenkettenvariablen gespeichert sein. Die INPL-Anweisung liest alle Zeichen des Programmierports ab, bis ein Wagenrücklauf oder die Zeichengrenze 254 erreicht ist. Von den über den Programmierport abgelesenen Zeichen wird kein Echo zurückgeführt.

Mit der INPL#-Anweisung wird eine vollständige Zeile vom Puffer des PRT2-Ports, und mit der INPL@-Anweisung wird eine vollständige Zeile vom Puffer des PRT1-Ports abgelesen. Die Funktionsweise dieser beiden Anweisungen entspricht der der INPL-Anweisung.

Syntax:

Zeichenkettenvariable

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 270,254 : REM ONE STRING OF < 254 BYTES
>20 INPL $(0) : REM READ LINE FROM PROGRAM PORT
>30 PRINT# $(0) : REM ECHO STRING TO PORT PRT2
```

INPS

Funktion:

Mit der INPS-Anweisung wird eine vollständige Zeichenkette aus dem Puffer des Programmierports gelesen. Es erfolgt kein Echo der Zeichen. Zu Kommunikationszwecken sollten Sie anstelle der INPUT- oder INPL-Anweisung die INPS-Anweisung verwenden, da möglicherweise alle ASCII-Zeichen signifikant sind. INPUT ist hierfür weniger geeignet, da die Eingabe bei einem Komma oder einem Wagenrücklauf gestoppt wird. INPL wird bei einem Wagenrücklauf abgebrochen.

Mit INPS# wird eine vollständige Zeichenkette vom Puffer des PRT2-Ports, und mit INPS@ wird eine vollständige Zeichenkette vom Puffer des PRT1-Ports abgelesen. Die Funktionsweise dieser beiden Anweisungen entspricht der der INPS-Anweisung.

Syntax:

INPS Zeichenkettenvariable, Zeichenanzahl

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>100 PRINT, “TYPE P TO PROCEED OR S TO STOP”
>110 REM READ SINGLE CHARACTER FROM PROGRAM PORT
>120 INPS $(0),1
>130 IF ASC($(0),1)= ASC(P) GOTO 500
>140 IF ASC($(0),1)= ASC(S) GOTO 700
>150 GOTO 100
```

INPUT

Funktion:

Mit der INPUT-Anweisung werden während der Programmausführung Daten über das Terminal eingegeben. Mit einer Eingangsanweisung können Daten einer bzw. mehreren Variablen zugeordnet werden. Die Variablen müssen durch ein Komma voneinander getrennt werden.

Mit der INPUT#-Anweisung können Sie Daten über den PRT2-Port und mit der INPUT@-Anweisung über den PRT1-Port einlesen. Die Funktionsweise beider Anweisungen entspricht der der INPUT-Anweisung.

Syntax:

INPUT

Beispiele:

```
>INPUT A,C
```

>Durch Eingabe von INPUT A,C wird auf dem Terminalbildschirm ein Fragezeichen (?) angezeigt. Sie müssen nun zwei durch ein Komma getrennte Zahlen eingeben. Wenn die eingegebenen Daten unzureichend sind, erscheint auf dem Terminalbildschirm die Meldung **TRY AGAIN**.

```
>1  REM EXAMPLE PROGRAM
>10 INPUT A,C
>20 PRINT A,C
>RUN

?1

TRY AGAIN

?1,2
l      2

READY
```

Sie können die INPUT-Anweisung so programmieren, daß eine beschreibende Aufforderung auf die erforderliche Eingabe hinweist. Die angezeigte Meldung erscheint in Anführungszeichen nach der INPUT-Anweisung. Wenn vor der ersten Variablen der Eingangsliste ein Komma steht, wird das Fragezeichen nicht angezeigt.

```
>1  REM EXAMPLE PROGRAM
>10 INPUT “ENTER A NUMBER”A
>20 PRINT SQR(A)
>30 END

READY
>RUN

ENTER A NUMBER
?4
 2

READY
>

>NEW

>1  REM EXAMPLE PROGRAM
>10 INPUT “ENTER A NUMBER – ”,A
>20 PRINT SQR(A)
>30 END

>RUN

ENTER A NUMBER – 25
 5

READY
>
```

Mit INPUT können auch Zeichenketten zugeordnet werden. Zeichenketten werden stets mit einem Wagenrücklauf (CR) beendet. Wenn mit einer INPUT-Anweisung mehr als ein Zeichenketteneingang angefordert wird, wird ein Fragezeichen angezeigt. Zeichenketten und Variablen können mit einer INPUT-Anweisung zugeordnet werden.

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 INPUT “NAME(CR),AGE – ”,$(1),A
>30 PRINT “HELLO ”,$(1), “YOU ARE ”,A,“ YEARS OLD.”
>40 END

READY
>RUN

NAME(CR),AGE – PAM
?29
HELLO PAM YOU ARE 29 YEARS OLD.

READY
>
```

Mit einer einzelnen INPUT-Anweisung können Zeichenketten und Variablen zugeordnet werden.

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,10
>20 INPUT “NAME(CR), AGE – ”,$(1),A
>30 PRINT “HELLO ”,$(1),“, YOU ARE ”, A,“ YEARS OLD”
>40 END
>RUN

NAME(CR),AGE – FRED
?15
HELLO FRED, YOU ARE 15 YEARS OLD

READY
>
```

LD@

Wichtig: Diese Anweisung bezieht sich nicht auf bestimmte Ports.

Funktion:

Mit der LD@-Anweisung werden Fließkommazahlen, die mit einer ST@-Anweisung gespeichert wurden, abgerufen. Die Adresse, in der die Zahl nach Ausführung der LD@-Anweisung gespeichert werden soll, wird durch den nach der LD@-Anweisung programmierten Ausdruck *Ausdr* bestimmt. Die LD@-Anweisung legt die Zahl im ARGUMENTSTAPEL in der mit [Ausdr] definierten Adresse ab.

Diese Anweisung kann mit CALL 77 verwendet werden, um Variablen aus einem geschützten Speicherbereich abzurufen. Dieser geschützte Bereich wird bei der Inbetriebnahme oder bei der Erteilung des RUN-Befehls nicht auf 0 gesetzt.

Wichtig: LD@ bezieht sich nicht auf bestimmte Ports.

Syntax:

LD@ [Ausdr]

Beispiel:

```
>P. MTOP
 24515

P. MTOP 10*6
 24455

>PUSH 24455 : CALL 77

>1   REM EXAMPLE PROGRAM
>5   DIM A(10),B(10)
>10  REM *** ARRAY SAVE ***
>20  FOR I = 0 TO 9
>30  A(I) = I+20
>40  PUSH A(I) : REM PUT NUMBER ON STACK
>50  ST@ 5FFFH-I*6
>60  NEXT I
>70  REM *** GET ARRAY ***
>80  FOR I = 0 TO 9
>90  LD@ 5FFFH-I*6
>100 POP B(I) : REM GET NUMBER FROM STACK
>110 PRINT B(I)
>120 NEXT I0

READY
>RUN

 20
 21
 22
 23
 24
 25
 26
 27
 28
 29

READY
>PUSH 5FFFH : CALL 77

>P. MTOP
 24575
```

READ

Funktion:

Mit der READ-Anweisung werden die in der DATA-Anweisung spezifizierten Ausdrücke abgerufen, und der Wert des Ausdrucks wird der Variablen der READ-Anweisung zugeordnet. Auf die READ-Anweisung folgen immer eine oder mehrere Variablen. Mehrere Variablen müssen durch Kommas getrennt werden.

Syntax:

READ

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 READ A,C
>30 PRINT A,C
>40 NEXT I
>50 RESTORE
>60 READ A,C
>70 PRINT A,C
>80 DATA 10,20,10/2,20/2,SIN(PI),COS(PI)

READY
>RUN

 10  20
 5  10
 0 -1
 10  20

READY
>
```

Jedesmal, wenn im Programm eine READ-Anweisung vorhanden ist, wird der unmittelbar nächste Ausdruck der DATA-Anweisung bewertet und der Variablen der READ-Anweisung zugeordnet. DATA-Anweisungen können an jeder beliebigen Stelle im Programm angeordnet werden. Sie werden nicht ausgeführt und verursachen keinen Fehler. DATA-Anweisungen sind verkettet und erscheinen als eine lange DATA-Anweisung. Wenn zu einem gegebenen Zeitpunkt alle Daten abgelesen wurden und eine weitere READ-Anweisung ausgeführt wird, wird das Programm gestoppt, und die Fehlermeldung **`ERROR: NO DATA – IN LINE XX`** wird auf dem Terminal ausgedruckt.

Kapitel 14

Konfigurationsfunktionen

Dieses Kapitel enthält eine Beschreibung und Darstellung der Befehle, die zur Konfiguration der Portparameter im BASIC-Programm und von der Befehlszeile aus verwendet werden. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 14.A enthalten.

Tabelle 14.A
Kapitelinhalt

CALL 30 – Konfiguration der Parameter des PRT2-Ports

Funktion:

Mit CALL 30 werden die Parameter des PRT2-Ports konfiguriert. Tabelle 14.B enthält eine Auflistung dieser Parameter. Außerdem sind die zulässigen Einstellungen in der Reihenfolge aufgeführt, in der sie vor der Ausführung des Aufrufs mit einer PUSH-Anweisung in den Stapel übertragen werden.

Tabelle 14.B
Parameter des PRT2-Ports

Parameter des PRT2-Ports	Einstellungen
Bits je Wort	5, 6, 7, 8
Parität	0 = keine, 1 = ungerade, 2 = gerade
Anzahl der Stoppbits	1 = 1 Stoppbit , 2 = 2 Stoppbits , 3 = 1,5 Stoppbits
Software-Handshake-Quittierung	0 = keine, 1 = XON-XOF
Hardware-Handshake-Quittierung	0 = DCD deaktiviert, 1 = DCD aktiviert

Syntax:

PUSH [Bits je Wort]
PUSH [Parität]
PUSH [Anzahl der Stoppbits]
PUSH [Software-Handshake-Quittierung aktivieren/deaktivieren]
PUSH [Hardware-Handshake-Quittierung aktivieren/deaktivieren]
CALL 30

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM CALL 30 INPUT PARAMETERS:
>20  REM FIRST PUSH : 5, 6, 7, OR 8 (BITS/CHARACTER)
>30  REM SECOND PUSH : 0, 1, OR 2 (NO PARITY, ODD, OR EVEN)
>40  REM THIRD PUSH : 1, 2, OR 3 (1, 2, OR 1.5 STOP BITS)
>50  REM FOURTH PUSH: 0 OR 1 (SOFTWARE HANDSHAKING
     DISABLE, ENABLED)
>60  REM FIFTH PUSH : 0 OR 1 (HARDWARE HANDSHAKING
     DISABLED, ENABLED)
>70  REM PRT2 DEFAULT CONFIGURATION IS:
>80  REM 1200 BAUD, 8 BITS/CHAR, NO PARITY, 1 STOP BIT, AND
>90  REM SOFTWARE HANDSHAKING ENABLED
>100 PUSH 8 0,1,1,0 : CALL 30
>110 CALL 31

19200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
>
```

CALL 78 – Einstellen der Baudrate des Programmierports

Funktion:

Mit CALL 78 wird die Baudrate des Programmierports vom Vorgabewert (1200 Baud) auf einen der folgenden Werte geändert: 300, 600, 1200, 2400, 4800, 9600 oder 19200 Baud. Die vorgegebene Baudrate beträgt 1200 Baud, wenn PRT1 als Programmierport konfiguriert wurde, und 19200 Baud, wenn der DH485-Port als Programmierport verwendet wird, Die gewünschte Kommunikationsrate wird mit einer PUSH-Anweisung in den Stapelspeicher geladen und mit CALL 78 aktiviert. Der Programmierport bleibt so lange für diese Baudrate konfiguriert, bis CALL 73 ausgeführt wird, oder bis einer der folgenden Zustände eintritt:

- Die Batterie ist erschöpft oder wurde entfernt.
- Der batteriegestützte Kondensator wurde entladen.
- Der EEPROM-Speicher ist nicht vorhanden oder nicht programmiert.
- Die Netzversorgung wird aus- und wieder eingeschaltet.

In diesem Fall wird die Baudrate auf den Vorgabewert 1200 zurückgesetzt.

Syntax:

PUSH [Baudrate]
CALL 78

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 4800
>20 CALL 78
```

CALL 99 – Rücksetzen der Funktion “Kopfzeiger drucken”

Funktion:

Mit CALL 99 wird der interne Druckkopf-Zeichenzähler des Druckers beim Drucken von breiten Formaten zurückgesetzt. Dieser Aufruf verhindert den automatischen Wagenrücklauf/Zeilenvorschub bei Zeichen 79. Sie müssen die Position der Zeichen in jeder Zeile selbst überprüfen.

Syntax:

CALL 99

Beispiel:

```
>10 REM EXAMPLE PROGRAM
>20 REM THIS PRINTS TIME BEYOND 80TH COLUMN
>30 PRINT TAB(79)
>40 CALL 99
>50 PRINT TAB(41), “TIME -”,
>60 PRINT H,“:”,M,“:”,S
>70 END
```

CALL 105 – Rücksetzen von PRT1 auf Voreinstellungen

Funktion:

Mit CALL 105 werden die Portparameter für PRT1 auf ihre Vorgabewerte zurückgesetzt. Diese Vorgabewerte sind in Tabelle 14.C aufgelistet.

Tabelle 14.C
Vorgabewerte der Portparameter für PRT1

Portparameter für PRT1	Voreinstellungen
Baudrate	1200 Baud
Anzahl der Datenbits	8 Bits
Anzahl der Stoppbits	1 Bit
Parität	keine Parität
Handshake-Quittierung	Software-Handshake-Quittierung

Syntax:

CALL 105

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 105
```

CALL 119 – Rücksetzen von PRT2 auf Voreinstellungen

Funktion:

Mit CALL 119 werden die Portparameter für PRT2 auf ihre Vorgabewerte zurückgesetzt. Diese Vorgabewerte sind in Tabelle 14.D aufgelistet.

Tabelle 14.D
Vorgabewerte der Portparameter für PRT2

Portparameter für PRT2	Vorgabeeinstellungen
Baudrate	1200 Baud
Anzahl der Datenbits	8 Bits
Anzahl der Stoppbits	1 Bit
Parität	keine Parität
Handshake-Quittierung	Software-Handshake-Quittierung

Syntax:

CALL 119

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 119
```

MODE

Funktion:

Mit dem MODE-Befehl werden die Portparameter von PRT1, PRT2 und DH485 konfiguriert.

Wichtig: Beim Drucken von Daten über den seriellen Port unter Verwendung der Hardware- oder Software-Handshake-Quittierung (Xon/Xoff) müssen Sie sicherstellen, daß ein ausreichender Pufferspeicherplatz vorhanden ist. Anderenfalls wird die Ausführung des BASIC-Programms so lange gestoppt, bis genügend Pufferspeicherplatz vorhanden ist. Anschließend setzt das BASIC-Modul die Abarbeitung des Programms an der Stelle fort, an der die Ausführung gestoppt wurde. Im Ausgangspuffer jedes Ports können maximal 256 Zeichen abgelegt werden. Weitere Hinweise sind in der Beschreibung der Aufrufe 36, 37, 95 und 96 enthalten.

Bei aktivierter Hardware-Handshake-Quittierung gelten für das BASIC-Modul die folgenden Regeln:

- Eine Übertragung findet erst statt, wenn CTS aktiviert wird.
- Nach dem Empfang eines Zeichens überprüft das BASIC-Modul das DSR-Signal. Ist dieses aktiv, wird das Zeichen in die Eingangswarteschlange eingereiht. Ist das DSR-Signal nicht aktiv, wird das Zeichen als Störsignal verworfen.

Eine Auflistung der Portparameter für PRT1 und PRT2 ist in Tabelle 14.E enthalten:

Tabelle 14.E
Portparameter für PRT1 und PRT2

Portparameter	Einstellungsmöglichkeiten	Vorgabe-einstellung
Baudrate	300, 600, 1200, 2400, 4800, 9600, 19200	1200
Arg1 (Parität)	keine (N), gerade (E), ungerade (O)	N
Arg2 (Anzahl der Datenbits)	7 oder 8	8
Arg3 (Anzahl der Stoppbits)	1 oder 2	1
Arg4 (Handshake-Quittierung)	keine Handshake-Quittierung (N) Software-Handshake-Quittierung (S) Hardware-Handshake-Quittierung (H) Hardware- und Software-Handshake-Quittierung (B)	S
Arg5 (Speicherung)	im Anwender-ROM und -RAM (E). im batteriegestützten RAM (R).	R

Wichtig: Wenn ein Argument (außer der Portbezeichnung und der Baudrate) nicht spezifiziert wird, wird ihm automatisch der zuletzt spezifizierte Wert zugeordnet.

Tabelle 14.F enthält eine Auflistung der Portparameter für DH485.

Tabelle 14.F
Portparameter für DH485

Portparameter	Einstellungsmöglichkeiten	Vorgabe-einstellung
Baudrate	300, 600, 1200, 2400, 4800, 9600, 19200	19200
Arg1 (Host-Netzknotenadresse)	0 bis 31	0
Arg2 (Modul-Netzknotenadresse)	1 bis 31	1
Arg3 (maximale Netzknotenadresse)	1 bis 31	31
Arg4 (nicht belegt)		
Arg5 (Speicherung)	im Anwender-ROM und -RAM (E). im batteriegestützten RAM (R).	R

Wichtig: Wenn MODE als Anweisung verwendet wird, ist die Speicheroption E nicht zulässig.

Syntax:

MODE (Portbezeichnung, Baudrate, Arg1, Arg2, Arg3, Arg4, Arg5)

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 MODE(DH485,19200,0,1,2,,R)
>.
.
>25 MODE(PRT1,1200,N,8,,,)
```

Kapitel 15

Zeichenkettenfunktionen

Dieses Kapitel enthält eine Beschreibung und Darstellung der Befehle, die zur Manipulierung von Zeichenkettendaten im BASIC-Programm und von der Befehlszeile aus verwendet werden. Eine Auflistung der entsprechenden Mnemonik ist in Tabelle 15.A enthalten.

Tabelle 15.A
Kapitelinhalt

CALL 60 – Zeichenkettenwiederholung

Funktion:

Mit CALL 60 wird ein Zeichen wiederholt und in einer Zeichenkette abgelegt. Dieser Aufruf eignet sich zur Erstellung von Ausgabeformaten. Spezifizieren Sie zunächst mit einer PUSH-Anweisung, wie oft das betreffende Zeichen wiederholt werden soll, und anschließend ebenfalls mit einer PUSH-Anweisung die Nummer der Zeichenkette, die das zu wiederholende Zeichen enthält. POP-Anweisungen sind nicht erforderlich. Die Anzahl der zu wiederholenden Zeichen kann die maximale Länge der Zeichenkette nicht überschreiten.

Syntax:

PUSH [Anzahl der Wiederholungen]
PUSH [Nummer der Basiszeichenkette]
CALL 60

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM STRING REPEAT EXAMPLE PROGRAM
>20 STRING 200,48
>30 $(1) = "*"
>40 PUSH 40 : REM THE NUMBER OF TIMES TO REPEAT CHARACTER
>50 PUSH 1 : REM BASE STRING NUMBER
>60 CALL 60
>70 PRINT $(1)
>80 END

READY
>RUN

****************************************

READY
>
```

CALL 61 – Zeichenkettenanhang

Funktion:

Mit CALL 61 wird eine Zeichenkette an das Ende einer zweiten Zeichenkette angehängt. Bei diesem Aufruf müssen zwei Zeichenkettenargumente spezifiziert werden: die Nummer der anzuhängenden Zeichenkette und die Nummer der Basiszeichenkette. Wenn die resultierende Zeichenkette länger als die maximale Zeichenkettenlänge ist, gehen die angehängten Zeichen verloren. Dieser Aufruf enthält keine Ausgangsargumente. Eine Zeichenketten-Verkettung wird wie folgt zugeordnet: (Beispiel: **`$(1)=$(1)+$(2)`**).

Wichtig: Überschreitet die Länge der neuen Zeichenkette die durch den STRING-Befehl zugewiesene Länge, wird auf dem Terminal eine Fehlermeldung ausgegeben, und das BASIC-Modul schaltet in den Befehlsmodus um.

Syntax:

PUSH [Nummer der anzuhängenden Zeichenkette]
PUSH [Nummer der Basiszeichenkette]
CALL 61

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  STRING 200,20
>20  $(1) = “How are ”
>30  $(2) = “you?”
>40  PRINT “BEFORE ”
>50  PRINT “$(1) = ”,$(1)
>60  PRINT “$(2) = ”,$(2)
>70  PUSH 2 : REM STRING NUMBER TO BE APPENDED
>80  PUSH 1 : REM BASE STRING NUMBER
>90  CALL 61 : REM INVOKE STRING APPEND ROUTINE
>100 PRINT “AFTER:”
>110 PRINT “$(1) = ”,$(1)
>120 PRINT “$(2) = ”,$(2)
>130 END

READY
>RUN

BEFORE:
$(1) = How are
$(2) = you?
AFTER:
$(1) = How are you?
$(2) = you?

READY
>
```

CALL 62 – Umwandlung einer Zahl in eine Zeichenkette

Funktion:

Mit CALL 62 wird eine Zahl oder eine numerische Variable in eine Zeichenkette umgewandelt. Hierzu müssen der Zeichenkette mindestens 14 Zeichen zugeordnet werden. Wenn der Exponent des umzuwandelnden Wertes voraussichtlich 100 oder größer ist, müssen 15 Zeichen zugeordnet werden. Mit der Fehlereingrenzungsfunktion werden nur Zeichenkettenzuordnungen von weniger als 14 Zeichen eingegrenzt. Dieser Aufruf enthält keine Ausgangsargumente.

Syntax:

PUSH [umzuwandelnde Zahl]
PUSH [Nummer der Zeichenkette, in die der Wert übertragen werden soll]
CALL 62

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,14
>20 INPUT “ENTER A NUMBER TO CONVERT TO A STRING ”,N
>30 PUSH N : REM NUMBER TO CONVERT TO STRING
>40 PUSH 1 : REM CONVERT NUMBER TO STRING 1
>50 CALL 62 : REM DO THE CONVERSION
>60 PRINT $(1)
>70 END

READY
>RUN

ENTER A NUMBER TO CONVERT TO A STRING 2E3
2000

READY
>RUN

ENTER A NUMBER TO CONVERT TO A STRING 1.233
1.233

READY
>
```

CALL 63 – Umwandlung einer Zeichenkette in eine Zahl

Funktion:

Mit CALL 63 wird die erste Dezimalzahl in der spezifizierten Zeichenkette in eine Zahl umgewandelt, die in den Argumentstapel übertragen wird. Gültige Zahlen und dazugehörige Zeichen sind 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., E, + und –. Das Komma ist kein gültiges Zahlenzeichen und beendet die Umwandlung.

Enthält die Zeichenkette keinen gültigen Wert, wird Null ausgegeben. Gültige Werte liegen zwischen 1 und 255. Geben Sie die Nummer der umzuwandelnden Zeichenkette mit einer PUSH-Anweisung ein. Es sind zwei POP-Anweisungen erforderlich, einmal zur Prüfung der Gültigkeit des Wertes und anschließend zur Übertragung des tatsächlichen Wertes. Wenn in einer Zeichenkette auf eine Zahl der Buchstabe E und ein weiterer Buchstabe oder ein nichtnumerisches Zeichen folgt, wird vorausgesetzt, daß keine Zahl gefunden wurde, da der Buchstabe kein gültiger Exponent ist. (CALL 63 gibt im ersten mit einer POP-Anweisung übertragenen Argument den Wert Null aus, um darauf hinzuweisen, daß die Zeichenkette keine gültige Zahl enthielt.)

Syntax:

PUSH [Nummer der umzuwandelnden Zeichenkette]
CALL 63
POP [Gültigkeit des Wertes]
POP [tatsächlicher Wert]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>20 INPUT “INPUT A STRING TO CONVERT ”,$(1)
>30 PUSH 1 : REM CONVERT STRING 1
>40 CALL 63
>50 POP V,N
>60 IF (V<>0) THEN PRINT $(1),“ ” N:GOTO 80
>70 PRINT “INVALID OR NO VALUE FOUND”
>80 END

READY
>RUN

INPUT A STRING TO CONVERT 123ABC
123ABC  123

READY
>RUN

INPUT A STRING TO CONVERT 1.2E–7
1.2E–7  1.2 E–7

READY
>RUN

INPUT A STRING TO CONVERT 1.3.6
INVALID OR NO VALUE FOUND

READY
>
```

CALL 64 – Auffinden einer Zeichenkette innerhalb einer Zeichenkette

Funktion:

Mit CALL 64 kann eine Zeichenkette innerhalb einer Zeichenkette gefunden werden, wobei das erstmalige Auftreten (die erste Position) dieser Zeichenkette festgestellt wird. Dieser Aufruf verfügt über zwei Eingangsargumente. Das erste ist die gesuchte Zeichenkette und das zweite ist die nach einer übereinstimmenden Zeichenkette abzusuchende Zeichenkette. Es ist ein Ausgangsargument erforderlich. Wenn der Wert ungleich Null ist, wurde eine übereinstimmende Zeichenkette an der durch den Wert des Ausgangsarguments gekennzeichneten Position gefunden. Diese Routine ähnelt der Routine BASIC INSTR$(Zeichenkettefinden$,Zeichenkette$) (Beispiel: **`L=INSTR$($(1),$(2))`**.

Syntax:

PUSH [Nummer der gesuchten Zeichenkette]
PUSH [Nummer der Basiszeichenkette]
CALL 64
POP [Position der übereinstimmenden Zeichenkette]

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE FIND STRING IN STRING ROUTINE
>20  STRING 100,20
>30  $(1) = “456”
>40  $(2) = “12345678”
>50  PUSH 1 : REM STRING NUMBER OF STRING TO BE FOUND
>60  PUSH 2 : REM BASE STRING NUMBER
>70  CALL 64 : REM GET THE LOCATION OF FIRST CHARACTER
>80  POP L
>90  IF (L=0) THEN PRINT “NOT FOUND”
>100 IF(L>0) THEN PRINT “FOUND AT LOCATION”,L
>110 END

READY
>RUN

FOUND AT LOCATION 4

READY
>
```

CALL 65 – Ersetzen einer Zeichenkette innerhalb einer Zeichenkette

Funktion:

Mit CALL 65 wird eine Zeichenkette innerhalb einer zweiten Zeichenkette ersetzt. Hierzu sind drei Argumente erforderlich. Das erste Argument ist die Nummer der Zeichenkette, welche die Zeichenkette ersetzt, die durch die Zeichenkettennummer des zweiten Arguments definiert ist. Das dritte Argument ist die Basiszeichennummer. Dieser Aufruf verfügt über keine Ausgangsargumente.

Wichtig: Überschreitet die Länge der neuen Zeichenkette die durch den STRING-Befehl zugewiesene Länge, wird auf dem Terminal eine Fehlermeldung ausgegeben, und das BASIC-Modul schaltet in den Befehlsmodus um.

Syntax:

PUSH [Nummer der neuen Zeichenkette]
PUSH [Nummer der zu ersetzenden Zeichenkette]
PUSH [Nummer der Basiszeichenkette]
CALL 65

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE OF REPLACE STRING IN STRING
>20  STRING 100,20
>30  $(0) = “RED–LINES”
>40  $(1) = “RED”
>50  $(2) = “BLUE”
>60  PRINT “BEFORE:”
>70  PRINT “$(0) = ”,$(0)
>80  PUSH 2 : REM STRING NUMBER OF THE STRING
     TO BE REPLACED WITH
>90  PUSH 1 : REM STRING NUMBER OF THE STRING TO BE
    REPLACED
>100 PUSH 0 : REM BASE STRING NUMBER
>110 CALL 65 : REM INVOKE REPLACE STRING IN STRING
>120 PRINT “AFTER:”
>130 PRINT “$(0) = ”,$(0)
>140 END

READY
>RUN

BEFORE:
$(0) = RED–LINES
AFTER:
$(0) = BLUE–LINES

READY
>
```

CALL 66 – Einfügen einer Zeichenkette in eine Zeichenkette

Funktion:

Mit CALL 66 wird eine Zeichenkette in eine zweite Zeichenkette eingefügt. Dieser Aufruf erfordert drei Argumente. Das erste Argument ist die Position, an der die neue Zeichenkette eingefügt werden soll. Das zweite Argument ist die Nummer der Zeichenkette, welche die in die Basiszeichenkette einzufügenden Zeichen enthält. Das dritte Argument ist die Nummer der Basiszeichenkette. Diese Routine verfügt über keine Ausgangsargumente.

Wichtig: Überschreitet die Länge der neuen Zeichenkette die durch den STRING-Befehl zugewiesene Länge, wird auf dem Terminal eine Fehlermeldung ausgegeben, und das BASIC-Modul schaltet in den Befehlsmodus um.

Syntax:

PUSH [Position der einzufügenden Zeichenkette]
PUSH [Nummer der einzufügenden Zeichenkette]
PUSH [Nummer der Basiszeichenkette]
CALL 66

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE ROUTINE TO INSERT A STRING IN A STRING
>20  STRING 100,15
>30  $(0) = “1234590”
>40  $(1) = “67890”
>50  PRINT “BEFORE:”
>60  PRINT “$(0) = ”,$(0)
>70  PUSH 6 : REM POSITION TO START THE INSERT
>80  PUSH 1 : REM STRING NUMBER TO BE INSERTED
>85  PUSH 0 : REM BASE STRING NUMBER
>90  CALL 66 : REM INVOKE INSERT A STRING IN A STRING
>100 PRINT “$(0) = ”, (0)
>110 END

READY
>RUN

BEFORE:
$(0) = 1234590
$(0) = 123456789090

READY
>
```

CALL 67 – Löschen einer Zeichenkette aus einer Zeichenkette

Funktion:

Mit CALL 67 wird eine Zeichenkette aus einer zweiten Zeichenkette gelöscht. Für diesen Aufruf sind zwei Argumente erforderlich. Das erste Argument ist die Nummer der Basiszeichenkette und das zweite die Nummer der aus der Basiszeichenkette zu löschenden Zeichenkette. Diese Routine verfügt über keine Ausgangsargumente.

Wichtig: Mit dieser Routine wird nur das erste Vorkommen der Zeichenkette gelöscht.

Syntax:

PUSH [Nummer der Basiszeichenkette]
PUSH [Nummer der zu löschenden Zeichenkette]
CALL 67

Beispiel:

```
>1   REM EXAMPLE PROGRAM
>10  REM ROUTINE TO DELETE A STRING IN A STRING
>20  STRING 200,14
>30  $(1) = “123456789012”
>40  $(2) = “12”
>50  PRINT “BEFORE:”
>60  PRINT “$(1) = ”,$(1)
>70  PUSH 1 : REM BASE STRING NUMBER
>80  PUSH 2 : REM STRING NUMBER OF THE STRING DELETED
>90  CALL 67 : REM INVOKE STRING DELETE ROUTINE
>100 PRINT “AFTER:”
>110 PRINT “$(1) = ”,$(1)
>120 END

READY
>RUN

BEFORE:
$(1) = 123456789012
AFTER:
$(1) = 3456789012

READY
>
```

CALL 68 – Bestimmung der Länge einer Zeichenkette

Funktion:

Mit CALL 68 wird die Länge einer Zeichenkette bestimmt. Dieser Aufruf erfordert ein Eingangsargument – die Nummer der Zeichenkette, an der dieser Aufruf ausgeführt wird – und ein Ausgangsargument – die tatsächliche Anzahl der Zeichen in der Zeichenkette (ausgenommen der Wagenrücklaufzeichen). Diese Routine ähnelt dem BASIC-Befehl LEN(str$) (Beispiel: **`L=LEN($l)`**). Zur genauen Festlegung der Länge muß die Zeichenkette mit einem Wagenrücklaufzeichen beendet werden. Wenn die Zeichenkette unter Verwendung des ASC-Befehls gefüllt wird, muß das letzte Zeichen ein Wagenrücklauf sein.

Syntax:

PUSH [Nummer der Zeichenkette]
CALL 68
POP [Anzahl der Zeichen]

Beispiel:

```
>1  REM EXAMPLE PROGRAM
>10 REM SAMPLE OF STRING LENGTH
>20 STRING 1 0,10
>30 $(1) = “1234567”
>40 PUSH 1 : REM BASE STRING
>50 CALL 68 : REM INVOKE STRING LENGTH ROUTINE
>60 POP L : REM GET LENGTH OF BASE STRING
>70 PRINT “THE LENGTH OF ”,$(1),“ IS”,L
>80 END

READY
>RUN

THE LENGTH OF 1234567 IS 7

READY
>
```

STRING

Funktion:

Mit der STRING-Anweisung wird ein Speicherbereich für Zeichenketten zugeordnet. Vorgabemäßig ist kein Speicherbereich für Zeichenketten zugeordnet. Wenn versucht wird, eine Zeichenkette mit einer Anweisung wie z.B. LET $(1)=“HALLO” zu definieren, bevor ein Speicherbereich für Zeichenketten zugeordnet wurde, wird die Fehlermeldung **`ERROR: MEMORY ALLOCATION`** angezeigt. Der erste Ausdruck ([Ausdr]) der STRING-Anweisung ist die gesamte Anzahl der Speicherbytes, die Zeichenketten zugeordnet werden sollen. Der zweite Ausdruck ([Ausdr]) definiert die maximale Byteanzahl jeder Zeichenkette. Der zweite Wert darf nicht größer als 254 sein. Diese zwei Zahlen bestimmen die gesamte Anzahl der definierten Zeichenkettenvariablen.

Das BASIC-Modul benötigt ein zusätzliches Byte für jede Zeichenkette sowie ein weiteres zusätzliches Byte. Das zusätzliche Zeichen für jede Zeichenkette ist für das abschließende Wagenrücklaufzeichen vorgesehen. Beispiel: Mit der Anweisung STRING 100,10 wird ein ausreichender Speicherbereich für neun 10-Byte-Zeichenkettenvariablen von $(0) bis $(8) zugeordnet, und alle 100 zugeordneten Bytes werden belegt. Beachten Sie bitte, daß $(0) eine gültige Zeichenkette des BASIC-Moduls ist.

Wichtig: Wenn die Zeichenkette ein ASCII-Nullzeichen enthält, fungiert dieses als Markierer, der das Ende der Zeichenkette kennzeichnet.

Wichtig: Nach der Zuordnung eines Speicherbereichs für Zeichenketten kann dieser nicht durch Befehle (z.B. **NEW**) und Anweisungen (z.B. **CLEAR**) neu zugeordnet werden. Auch kann er durch Aus- und erneutes Einschalten des Moduls nur dann neu zugeordnet werden, wenn die Batteriepufferung deaktiviert ist. Zur Neuzuordnung des Speicherbereichs muß die Anweisung STRING 0,0 ausgeführt werden. Mit dieser Anweisung wird kein Speicherbereich für Zeichenkettenvariablen zugeordnet.

Wichtig: Bei jeder Ausführung der Anweisung STRING[Ausdr],[Ausdr] führt das BASIC-Modul einen der CLEAR-Anweisung entsprechenden Befehl aus. Dies ist notwendig, weil Zeichenkettenvariablen und numerische Variablen denselben externen Speicherbereich belegen. Nach Ausführung der STRING-Anweisung werden alle Variablen gelöscht. Deshalb sollte die Speicherzuordnung für Zeichenketten möglichst am Beginn des Programms (möglichst in der ersten Anweisung) durchgeführt werden. Wenn der für die Zeichenketten reservierte Speicherbereich neu zugeordnet wird, gehen alle definierten Variablen verloren.

Syntax:

STRING [Ausdr], [Ausdr]

Beispiele:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,30
>20 $(0) = "-----MONTHLY REPORT-----"
>30 PRINT $(0)

READY
>RUN

-----MONTHLY REPORT-----

READY
>
```

Umwandlungstabelle (dezimal/hexadezimal/oktal/ASCII)

Überblick über die mathematischen Umwandlungen

Die folgende Tabelle enthält eine Auflistung der Dezimal-, Hexadezimal-, Oktal- und ASCII-Umwandlungen.

Tabelle A.1
Umwandlungstabelle

Spalte 1				Spalte 2				Spalte 3				Spalte 4			
DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC
00	00	000	NUL	32	20	040	SP	64	40	100	@	96	60	140	`
01	01	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
02	02	002	STX	34	22	042	”	66	42	102	B	98	62	142	b
03	03	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
04	04	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
05	05	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
06	06	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
07	07	007	BEL	39	27	047	’	71	47	107	G	103	67	147	g
08	08	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
09	09	011	HT	41	29	051	)	73	49	111	I	105	69	151	i
10	0A	012	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	VT	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	FF	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	0D	015	CR	45	2D	055	–	77	4D	115	M	109	6D	155	m
14	0E	016	SO	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	0F	017	SI	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS	60	3C	074	<	92	5C	134	\	124	7C	174	.
29	1D	035	GS	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS	62	3E	076	>	94	5E	135	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

Schnellinformation zu BASIC-Befehlen, Anweisungen und Aufrufen

Auflistung der Mnemonik – Überblick

Die folgende Tabelle enthält eine Auflistung der in diesem Handbuch enthaltenen Mnemonik sowie deren Beschreibung und Seitenverweise.

Tabelle B.1
Schnellinformation

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehlsmodus aktiv	Seite
ABS( )		Absoluter Wert		3-11
ASC( )		Ganzzahlwert eines ASCII-Zeichens		3-13
ATN( )		Arkustangens eines Arguments		3-10
BRKPNT		Konfiguration einer Programmunterbrechung	*	4-2
CALL 14	PUSH [Wortanzahl im Eingangspuffer des BASIC-Moduls] POP [umgewandelter Wert]	16-Bit-Ganzzahl mit Vorzeichen in BASIC-Fließkommawert		9-1
CALL 15	PUSH [Wortanzahl im Eingangspuffer des BASIC-Moduls] POP [umgewandelter Wert]	16-Bit-Ganzzahl ohne Vorzeichen in BASIC-Fließkommawert		9-2
CALL 16	PUSH [BASIC-Zeilennummer]	Aktivierung der Interrupt-Funktion bei Empfang eines DF1-Datenpakets		8-2
CALL 17	keine	Deaktivierung eines DF1-Datenpaket-Interrupts		8-3
CALL 18	keine	Erneute Aktivierung der Abbruchfunktion **[CTRL-C]**		4-6
CALL 19	keine	Deaktivierung der Unterbrechungsfunktion **[CTRL-C]**		4-6
CALL 20	PUSH [BASIC-Zeilennummer]	Aktivierung der Interrupt-Funktion eines SLC-Prozessors		8-3
CALL 21	keine	Deaktivierung der Interrupt-Funktion eines SLC-Prozessors		8-4
CALL 22	PUSH [Nummer des Quellports] PUSH [maximale Anzahl der zu übertragenden Zeichen] PUSH [Dezimalwert des Zeichenbegrenzers] PUSH [Wahl des Zielfiles und/oder der Ziel-Zeichenkette] PUSH [Wortversatz im Zielfile] PUSH [Zeichenkettennummer] PUSH [Wahl der Bytefolge] POP [Status von CALL 22]	Datenübertragung von PRT1 oder PRT2 an die SLC-E/A oder in die M-Files		13-2

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
CALL 23	PUSH [Nummer des Zielports und/oder der internen Zeichenkette] PUSH [Wahl des Quellfiles] PUSH [Wortversatz im Quellfile] PUSH [Nummer der Zeichenkette] PUSH [Wahl der Bytefolge] POP [Status von CALL 23]	Datentransfer von den SLC-E/A bzw. den M-Files an PRT1 oder PRT2		12-2
CALL 24	PUSH [umzuwandelnder Wert] PUSH [Wortanzahl im Ausgangspuffer des BASIC-Moduls]	BASIC-Fließkommawert in 16-Bit-Ganzzahl mit Vorzeichen		9-3
CALL 25	PUSH [umzuwandelnder Wert] PUSH [Wortanzahl im Ausgangspuffer des BASIC-Moduls]	BASIC-Fließkommawert in 16-Bit-Binärwert		9-3
CALL 26	POP[Status des SLC-Prozessors]	Erteilung eines SLC-Prozessorinterrupts		8-5
CALL 27	PUSH [Typ des READ-Befehls] PUSH [dezentrale Netzknotenadresse] PUSH [dezentrale Filenummer] PUSH [dezentraler Filetyp] PUSH [Versatz des beginnenden Wortes im dezentralen File] PUSH [Anzahl der zu übertragenden Worte] PUSH [Zeitablaufwert der Nachricht] PUSH [Wahl des Zielfiles] PUSH [Wortversatz im Zielfile] PUSH [Zeichenkettennummer] POP [Status von CALL 27]	Datentransfer vom dezentralen DH-485-Datenfile an den SLC-Prozessor		13-9
CALL 28	PUSH [Typ des WRITE-Befehls] PUSH [dezentrale Netzknotenadresse] PUSH [dezentrale Filenummer] PUSH [dezentraler Filetyp] PUSH [Versatz des dezentralen beginnenden Wortes] PUSH [Anzahl der zu übertragenden Worte] PUSH [Zeitablaufwert der Nachricht] PUSH [Wahl des Quellfiles] PUSH [Wortversatz im Quellfile] PUSH [Zeichenkettennummer] POP [Status von CALL 28]	Datentransfer vom SLC-Prozessor in den dezentralen DH-485-Datenfile		12-9
CALL 29	PUSH [CALL 27, 28, 122 oder 123 für den zu aktivierenden Aufruf] POP [Transaktionsstatus]	Bearbeitung aller Fehler, die von der ONERR-Anweisung nicht behandelt werden		12-16, 13-17
CALL 30	PUSH [Bits je Wort] PUSH [Parität aktivieren] PUSH [Anzahl der Stoppbits] PUSH [Software-Handshake-Quittierung ein/aus] PUSH [Hardware-Handshake-Quittierung ein/aus]	Konfiguration der PRT2-Portparameter		14-1
CALL 31	keine	Anzeige der aktuellen PRT2-Portparameter		12-17
CALL 35	POP [ASCII-Wert des Zeichens]	Abruf des numerischen Eingangszeichens von PRT2		13-19
CALL 36	PUSH [Wahl des Puffers] POP [Anzahl der Zeichen]	Abruf der Zeichenanzahl in den PRT2-Puffern		11-2

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
CALL 37	PUSH [Wahl des Puffers]	Löschen des PRT2-Ein- und Ausgangspuffers		12-18
CALL 38	PUSH [0 oder 1]	Einleitung von Transaktionen, die in den Aufrufen 27, 28, 122 und 123 definiert sind		8-6
CALL 40	PUSH [Stunden] PUSH [Minuten] PUSH [Sekunden]	Einstellen der Uhrzeit (Stunde, Minute, Sekunde)		10-2
CALL 41	PUSH [Datum] PUSH [Monat] PUSH [Jahr]	Einstellung des Datums (Tag, Monat, Jahr)		10-3
CALL 42	PUSH [Wochentag]	Einstellung des Wochentags		10-4
CALL 43	PUSH [Zeichenkettennummer]	Abruf der Datum-/Uhrzeit-Zeichenkette		10-4
CALL 44	POP [Tag] POP [Monat] POP [Jahr]	Abruf des Datums, numerisch (Tag, Monat, Jahr)		10-5
CALL 45	PUSH [Zeichenkettennummer]	Abruf der Zeit-Zeichenkette		10-5
CALL 46	POP [Stunde] POP [Minute] POP [Sekunde]	Abruf der Zeit, numerisch		10-6
CALL 47	PUSH [Zeichenkettennummer]	Abruf der Wochentag-Zeichenkette		10-7
CALL 48	POP [Wochentag]	Abruf des Wochentags, numerisch		10-7
CALL 51	POP [Status des Ausgangsabbildpuffers]	Überprüfung des CPU-Ausgangsabbildpuffers		11-3
CALL 52	PUSH [Zeichenkettennummer]	Abruf der Datum-Zeichenkette		10-8
CALL 53	POP [Prozessorstatus]	Übertragung des CPU-Ausgangsabbildpuffers in den Eingangspuffer des BASIC-Moduls		13-21
CALL 54	POP [Prozessormodus]	Übertragung des BASIC-Ausgangspuffers an den CPU-Eingangsabbildpuffer		12-18
CALL 55	POP [Status des Eingangsabbildpuffers]	Überprüfung des CPU-Eingangsabbildpuffers		11-4
CALL 56	PUSH [Anzahl der zu übertragenden Worte] POP [Prozessorstatus]	Übertragung des CPU-M0-Files in den Eingangspuffer des BASIC-Moduls		13-22
CALL 57	PUSH [Anzahl der zu übertragenden Worte] POP [Prozessormodus]	Übertragung des BASIC-Ausgangspuffers in den CPU-M1-File		12-19
CALL 58	POP [Schreibstatus des Modulfiles M0]	Überprüfung des M0-Filestatus		11-5
CALL 59	POP [Lesestatus des Modulfiles M1]	Überprüfung des M1-Filestatus		11-6
CALL 60	PUSH [Anzahl der Zeichenwiederholungen] PUSH [Basiszeichenkette]	Zeichenkettenwiederholung		15-1
CALL 61	PUSH [Nummer der anzuhängenden Zeichenkette] PUSH [Nummer der Basiszeichenkette]	Zeichenkettenanhang (Verkettung)		15-2
CALL 62	PUSH [Nummer der umzuwandelnden Zeichenkette] PUSH [Nummer der Zeichenkette, in die der Wert übertragen werden soll]	Umwandlung einer Zahl in eine Zeichenkette		15-4

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
CALL 63	PUSH [Nummer der umzuwandelnden Zeichenkette] POP [Gültigkeit des Wertes] POP [tatsächlicher Wert]	Umwandlung einer Zeichenkette in eine Zahl		15-5
CALL 64	PUSH [Nummer der gesuchten Zeichenkette] PUSH [Nummer der Basiszeichenkette] POP [übereinstimmende Position]	Auffinden einer Zeichenkette innerhalb einer Zeichenkette		15-6
CALL 65	PUSH [Nummer der neuen Zeichenkette] PUSH [Nummer der zu ersetzenden Zeichenkette] PUSH [Nummer der Basiszeichenkette]	Ersetzen einer Zeichenkette innerhalb einer Zeichenkette		15-7
CALL 66	PUSH [Einfügeposition] PUSH [Zeichenkettennummer des eingefügten Zeichens] PUSH [Nummer der Basiszeichenkette]	Einfügen einer Zeichenkette in eine Zeichenkette		15-8
CALL 67	PUSH [Nummer der Basiszeichenkette] PUSH [Nummer der zu löschenden Zeichenkette]	Löschen einer Zeichenkette aus einer Zeichenkette		15-9
CALL 68	PUSH [Zeichenkettennummer] POP [Anzahl der Zeichen]	Bestimmung der Länge einer Zeichenkette		15-10
CALL 70	keine	Sprung aus einem ROM- in ein RAM-Programm		8-8
CALL 71	PUSH [ROM-Programmnummer]	Sprung aus einem ROM-/RAM-Programm in ein ROM-Programm		8-9
CALL 72	keine	Rücksprung zur RAM-/ROM-Routine		8-10
CALL 73	keine	Deaktivierung der RAM-Speicher-Batteriepufferung		5-2
CALL 74	keine	Aktivierung der RAM-Speicher-Batteriepufferung		5-2
CALL 75	POP [Prozessormodus]	Überprüfung des CPU-Status einer SLC-500-Steuerung		11-7
CALL 77	PUSH [neue MTOP-Adresse]	Geschützter Variablenspeicher		5-3
CALL 78	PUSH [Baudrate]	Einstellen der Baudrate des Programmierports		14-3
CALL 80	POP [Batteriezustand]	Überprüfung der Batterie		11-8
CALL 81	keine	Überprüfung und Beschreibung des Anwenderspeichermoduls	*	5-4
CALL 82	keine	Überprüfung der Belegung des Anwenderspeichermoduls	*	5-5
CALL 84	PUSH [Versatz des beginnenden Wortes im DH-485-Schnittstellenfile] PUSH [Anzahl der zu übertragenden Worte] POP [Übertragungsstatus]	Übertragung des DH-485-Schnittstellenfiles in den Eingangspuffer des BASIC-Moduls		13-23
CALL 85	PUSH [Versatz des beginnenden Wortes im DH-485-Schnittstellenfile] PUSH [Anzahl der zu übertragenden Worte] POP [Übertragungsstatus]	Übertragung des Ausgangspuffers des BASIC-Moduls in den DH-485-Schnittstellenfile		12-20
CALL 86	POP [dezentraler Schreibstatus des DH-485-Schnittstellenfiles]	Überprüfung des dezentralen Schreibstatus des DH-485-Schnittstellenfiles		11-8

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
CALL 87	POP [dezentraler Lesestatus des DH-485-Schnittstellenfiles]	Überprüfung des dezentralen Lesestatus des DH-485-Schnittstellenfiles		11-9
CALL 90	PUSH [Netzknotenadresse des dezentralen Gerätes] PUSH [Filenummer des dezentralen Gerätes] PUSH [Filetyp des dezentralen Gerätes] PUSH [Versatz des beginnenden Elementes (x2) im File des dezentralen Gerätes] PUSH [Anzahl der zu übertragenden Elemente] PUSH [Zeitablaufwert der Nachricht] POP [Status des Nachrichtenbefehls]	Lesetransfer des dezentralen DH-485-Datenfiles in den BASIC-Eingangspuffer		13-24
CALL 91	PUSH [Netzknotenadresse des dezentralen Gerätes] PUSH [Filenummer des dezentralen Gerätes] PUSH [Filetyp des dezentralen Gerätes] PUSH [Versatz des beginnenden Elementes (x2) im File des dezentralen Gerätes] PUSH [Anzahl der zu übertragenden Elemente] PUSH [Zeitablaufwert der Nachricht] POP [Status des Nachrichtenbefehls]	Schreibtransfer des BASIC-Ausgangspuffers in den dezentralen DH-485-Datenfile		12-21
CALL 92	PUSH [Netzknotenadresse des dezentralen Gerätes] PUSH [Versatz des beginnenden Elements (x2) im File des dezentralen Gerätes] PUSH [Anzahl der zu übertragenden Worte] PUSH [Zeitablaufwert der Nachricht] POP [Status des Nachrichtenbefehls]	Lesetransfer des dezentralen DH-485-Schnittstellenfiles in den Eingangspuffer des BASIC-Moduls		13-27
CALL 93	PUSH [Netzknotenadresse des dezentralen Gerätes] PUSH [Versatz des beginnenden Elementes (x2) im File des dezentralen Gerätes] PUSH [Anzahl der zu übertragenden Worte] PUSH [Zeitablaufwert der Nachricht] POP [Status des Nachrichtenbefehls]	Schreibtransfer des Ausgangspuffers des BASIC-Moduls in den dezentralen DH-485-Schnittstellenfile		12-25
CALL 94	keine	Anzeige der aktuellen PRT1-Portkonfiguration		12-27
CALL 95	PUSH [Wahl des Puffers] POP [Zeichenanzahl]	Abruf der Zeichenanzahl aus den PRT1-Puffern		11-10
CALL 96	PUSH [Wahl des Puffers]	Löschen der PRT1-Eingangs- und Ausgangspuffer		12-28
CALL 97	keine	Aktivierung des DTR-Signals am PRT2-Port		11-11
CALL 98	keine	Deaktivierung des PRT2-DTR-Signals		11-11
CALL 99	keine	Rücksetzen der Funktion “Kopfzeiger drucken”		14-4
CALL 101	PUSH [beginnende Adresse] PUSH [Endadresse]	Hochladen des Anwender-Speichermodul-Codes an den Host		5-5
CALL 103	keine	Ausdruck des PRT1-Ausgangspuffers und -Zeigers		5-6
CALL 104	keine	Ausdruck des PRT1-Eingangspuffers und -Zeigers		5-7
CALL 105	keine	Rücksetzen des PRT1-Ports auf Voreinstellungen		14-4

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
CALL 108	PUSH [Betriebscode] PUSH [Abfrage- oder ACK-Zeitablauf] PUSH [erneute Übertragungs- oder ENQ-Versuche] PUSH [RTS-Einschaltverzögerung oder Wiederholungsversuche bei Empfang einer NAK-Meldung] PUSH [RTS-Ausschaltverzögerung oder NULL-Wert] PUSH [DF1-Adresse des BASIC-Moduls]	Aktivierung der DF1-Treiberkommunikation		11-12
CALL 109	keine	Ausdrucke des Argumentstapels		5-8
CALL 110	kein	Ausdruck des PRT2-Ausgangspuffers und -Zeigers		5-9
CALL 111	keine	Ausdruck des PRT2-Eingangspuffers und -Zeigers		5-10
CALL 112	PUSH [Zustand LED1] PUSH [Zustand LED2]	Steuerung der Anwender-LEDs		12-28
CALL 113	keine	Deaktivierung der DF1-Treiberkommunikation		11-19
CALL 114	keine	Übertragung des DF1-Datenpakets		12-29
CALL 115	POP [DF1-Übertragungsstatus]	Überprüfung des DF1-Übertragungsstatus		12-30
CALL 117	POP [Länge des DF1-Datenpakets]	Abruf der DF1-Datenpaketlänge		13-29
CALL 118	PUSH [Aufrufaktivierung/-deaktivierung] PUSH [Wahl des Zielfiles und oder der Zielzeichenkette] PUSH [Wortversatz im Zielfile] PUSH [Zeichenkettennummer] PUSH [maximale Wortlänge] POP [Status von CALL 118]	Ermöglichung freilaufender Schreibtransfers von einem dezentralen SLC- oder PLC-Netzknoten		13-30
CALL 119	keine	Rücksetzen des PRT2-Ports auf Voreinstellungen		14-5
CALL 120	PUSH [entsprechender Dezimalwert]	Löschen des Ein- und Ausgangspuffers des BASIC-Moduls		11-20
CALL 121	POP [Programmkennungsnummer]	Abruf der Programmkennungsnummer des SLC-Prozessors		11-21
CALL 122	PUSH [Typ des PLC-READ-Befehls] PUSH [Netzknotenadresse des dezentralen PLCs] PUSH [Filenummer des dezentralen PLCs] PUSH [Filetyp des dezentralen PLCs] PUSH [Versatz des beginnenden Elementes im dezentralen PLC-File] PUSH [Anzahl der zu übertragenden Elemente] PUSH [Zeitablaufwert der Nachricht] PUSH [Wahl des Zielfiles] PUSH [Wortversatz im Zielfile] PUSH [Zeichenkettennummer] POP [Status von CALL 122]	Lesetransfer eines PLC-Datenfiles		13-36

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
CALL 123	PUSH [Typ des PLC-WRITE-Befehls] PUSH [Netzknotenadresse des dezentralen PLCs] PUSH [Filenummer des dezentralen PLCs] PUSH [Filetyp des dezentralen PLCs] PUSH [Versatz des beginnenden Wortes im dezentralen PLC-File] PUSH [Anzahl der zu übertragenden Elemente] PUSH [Zeitablaufwert der Nachricht] PUSH [Wahl des Quellfiles] PUSH [Wortversatz im Quellfile] PUSH [Zeichenkettennummer] POP [Status von CALL 123]	Schreibtransfer eines PLC-Datenfiles		12-31
CBY( )		Abruf von Daten aus einer spezifizierten Speicheradresse		3-18
CHR( )		Umwandlung eines numerischen Wertes in ASCII-Zeichen		3-16
CLEAR		Löschen von Variablen, Interrupts und Zeichenketten		6-1
CLEARI		Löschen von Interrupts		6-2
CLEARS		Löschen aller Stapelspeicher		6-3
CLOCK0		Deaktivierung der Echtzeituhr		7-2
CLOCK1		Aktivierung der Echtzeituhr		7-1
CONT		Fortsetzung der Programmabarbeitung nach einer STOP-Anweisung oder nach **[CTRL-C]**.	*	4-4
CONTROL C		Stoppen der Ausführung und Umschalten in den Befehlsmodus		4-5
CONTROL Q		Erneutes Starten einer LIST-Anweisung nach **[CTRL-S]**.		4-8
CONTROL S		Unterbrechung einer LIST-Anweisung		4-7
COS( )		Ausgabe des Kosinus eines Arguments		3-10
DATA		Spezifizierung der mit einer READ-Anweisung abgelesenen Daten		6-4
DBY( )		Abruf oder Zuordnung von Daten an den/aus dem internen Datenspeicher des BASIC-Moduls		3-19
DIM		Zuordnung eines Speicherbereichs für Datenfeldvariablen		6-5
DO-UNTIL		Konfiguration einer bedingten DO-Schleife		7-5
DO-WHILE		Konfiguration einer bedingten DO-Schleife		7-3
EDIT		Bearbeiten einer BASIC-Porgrammzeile	*	4-9
END		Abschluß der Programmausführung		7-5
EOF		Prüfen, ob Eingangspuffer leer ist		3-17

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
ERASE		Löschen des letzten im ROM gespeicherten BASIC-Programms durch Ausführung eines PROG-Befehls	*	4-10
EXP( )		“e” (2,7182818) wird mit X potenziert		3-13
FOR-TO-(STEP) -NEXT		Konfiguration der For-Next-Schleife		7-6
FREE		Überprüfung der im RAM frei verfügbaren Bytes		3-17
GET		Ablesen des Terminaleingangs		13-47
GET#		Ablesen des an PRT2 angeschlossenen Terminaleingangs		13-47
GET@		Ablesen des an PRT1 angeschlossenen Terminaleingangs		13-47
GOSUB		Ausführung des Unterprogramms		8-11
GOTO		Übertragung der Steuerung an eine bestimmte Programmzeile		7-8
IDLE		Zwangsweise Aktivierung des Modus “Auf Interrupt warten” für das BASIC-Modul		4-10
IF–THEN–ELSE		Bedingungsprüfung		7-9
INPL		Ablesen der Zeichenzeile aus dem Puffer des Programmierports		13-48
INPL#		Ablesen der Zeichenzeile aus dem PRT2-Puffer		13-48
INPL@		Ablesen der Zeichenzeile aus dem PRT1-Puffer		13-48
INPS		Ablesen der Zeichenkette aus dem Puffer des Programmierports		13-48
INPS#		Ablesen der Zeichenkette aus dem PRT2-Puffer		13-48
INPS@		Ablesen der Zeichenkette aus dem PRT1-Puffer		13-48
INPUT		Eingabe einer Zeichenkette bzw. einer Variablen		13-49
INPUT#		Eingabe einer Zeichenkette oder Variablen über den PRT2-Port		13-49
INPUT@		Eingabe einer Zeichenkette oder Variablen über den PRT1-Port		13-49
INT( )		Ganzzahl		3-11
LD@		Laden einer Variablen		13-51
LEN		Einlesen der Anzahl der Speicherbytes in das aktuelle Programm		3-18
LET		Zuordnung einer Variablen oder eines Zeichenkettenwerts (LET ist optional)		6-6
LIST		Anzeige des Programms auf dem Terminalbildschirm	*	4-11

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
LIST#		Ausgabe des Programms über seriellen Drucker	*	4-12
LIST@		Ausgabe des Programms über das an PRT1 angeschlossene Gerät	*	4-12
LOG( )		Natürlicher Logarithmus		3-13
MODE		Konfiguration der Portparameter für PRT1, PRT2 und DH485		4-13, 14-5
MTOP		Anzeige der letzten gültigen Speicheradresse		3-18
NEW		Löschen des im RAM gespeicherten Programms	*	4-14
NEXT		Überprüfung der Bedingung für For-Next-Schleife		7-10
NOT( )		Einer-Komplement		3-11
NULL		Spezifikation der NULL-Zeichen nach einem Wagenrücklauf/Zeilenvorschub	*	4-14
ONERR		Sprung zur spezifizierten Zeilennummer bei Feststellung eines Fehlers		8-12
ON-GOSUB		Bedingte GOSUB-Anweisung		8-13
ON-GOTO		Bedingte GOTO-Anweisung		7-11
ONTIME		Erstellung eines Interrupts, wenn TIME gleich oder größer ist als die im Argument der ONTIME-Anweisung spezifizierte Zeilennummer		8-14
PH0.		Ausgabe des Hexadezimalwerts mit Nullunterdrückung auf dem Terminal		12-42
PH0.#		Ausgabe des Hexadezimalwerts mit Nullunterdrückung über PRT2		12-42
PH0.@		Ausgabe des Hexadezimalwerts mit Nullunterdrückung über PRT1		12-42
PH1.		Ausgabe des Hexadezimalwerts ohne Nullunterdrückung auf dem Terminal		12-42
PH1.#		Ausgabe des Hexadezimalwerts ohne Nullunterdrückung über PRT2		12-42
PH1.@		Ausgabe des Hexadezimalwerts ohne Nullunterdrückung über PRT1		12-42
PI		PI-3,1415926		3-12
POP		Zuweisung des Argumentstapels an Variablen		8-17
PRINT		Ausgabe von Variablen, Zeichenketten und Buchstaben auf dem Terminal; P. ist die Abkürzung für PRINT		12-39
PRINT#		Ausgabe über PRT2		12-40
PRINT@		Ausgabe über PRT1		12-40
PRINT CR		Ausgabe eines Wagenrücklaufs		12-40

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
PRINT SPC( )		Ausgabe von Leerzeichen		12-41
PRINT TAB( )		Ausgabe von Tabulatorzeichen		12-41
PRINT USING(Fx)		Ausgabe numerischer Werte in mathematischer Schreibweise		12-41
PRINT USING(#.#)		Ausgabe eines numerischen Wertes in Dezimalschreibweise		12-41
PRINT USING(0)		Aufruf des vorgegebenen Ausgabemodus		12-42
PROG		Sichern des aktuellen Programms im EPROM	*	4-15
PROG1		Sichern der Baudrate im EPROM	*	4-16
PROG2		Sichern der Baudrate im EPROM und Programmausführung nach Rücksetzung	*	4-17
PUSH		Einspeichern von Ausdrücken in den Argumentstapel		8-15
RAM		Aufruf des RAM-Modus	*	4-19
READ		Lesen der Daten in einer DATA-Anweisung		13-53
REM		Einfügen einer Anmerkung oder Kommentarzeile		4-19
REN		Neunumerierung des BASIC-Programms		4-20
RESTORE		Rücksetzen des Lesezeigers		6-7
RETI		Rückkehr von einem Interrupt		8-18
RETURN		Rücksprung aus einer Subroutine		8-18
RND		Quasi-Zufallszahl		3-12
ROM		Wahl des ROM-Modus	*	4-21
RROM		Wahl des ROM-Modus und Ausführung des gewählten Programms	*	4-22
RUN		Ausführung eines Programms	*	4-23
SGN		Vorzeichen		3-12
SIN( )		Ausgabe des Sinuswertes eines Arguments		3-9
SNGLSTP		Einleitung einer schrittweisen Programmausführung	*	4-24
SQR( )		Quadratwurzel		3-12
ST@		Speichern einer Variablen		12-43
STOP		Unterbrechung der Programmausführung		8-20
STRING		Speicherzuordnung für Zeichenketten		15-10
TAN( )		Ausgabe des Tangens eines Arguments		3-10
TIME		Abruf/Zuordnung der freilaufenden Uhr		3-20

Mnemonik	Erforderliche PUSH- und POP-Anweisungen	Beschreibung	Nur im Befehls-modus aktiv	Seite
VER		Anzeige der aktuellen Firmware-Version des BASIC-Moduls		4-25
XBY( )		Abruf oder Zuordnung von Daten an einen/von einem externen Datenspeicher des BASIC-Moduls		3-19
XFER		Übertragung eines Programms vom ROM- in den RAM-Speicher	*	4-26
+		Addition		3-5
/		Division		3-5
**		Potenzierung		3-5
*		Multiplikation		3-5
-		Subtraktion		3-5
.AND.		Logisches UND		3-8
.OR.		Logisches ODER		3-8
.XOR.		Logisches Exklusiv-ODER		3-8
@		Direkte Kommunktion über PRT1		3-17
#		Direkte Kommunikation über PRT2		3-17

Symbole

Zahlen

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

Rockwell Automation

Rockwell Automation vereint führende Marken der industriellen Automation und hilft seinen Kunden, den größtmöglichen Gewinn aus ihren Investitionen zu ziehen. Wir bieten ein umfassendes Sortiment an leicht integrierbaren Produkten. Unsere Produkte werden durch Kundendienstmitarbeiter vor Ort und weltweit, über ein globales Netzwerk von Systemanbietern und die Forschungs- und Entwicklungszentren von Rockwell umfassend unterstützt.

Weltweite Niederlassungen.

Ägypten • Argentinien • Australien • Bahrain • Belgien • Bolivien • Brasilien • Bulgarien • Chile • Costa Rica • Dänemark • Deutschland • Dominikanische Republik • Ecuador El Salvador • Finnland • Frankreich • Ghana • Griechenland • Großbritannien • Guatemala • Honduras • Hongkong • Indien • Indonesien • Iran • Irland • Island • Israel • Italien Jamaika • Japan • Jordanien • Kanada • Kolumbien • Korea • Kroatien • Kuwait • Libanon • Macao • Malaysia • Malta • Marokko • Mexiko • Niederlande • Neuseeland • Nigeria Norwegen • Österreich • Oman • Pakistan • Panama • Peru • Philippinen • Polen • Portugal • Puerto Rico • Qatar • Republik Südafrika • Rumänien • Rußland • Saudi-Arabien Singapur • Slowakei • Slowenien • Spanien • Schweden • Schweiz • Taiwan • Thailand • Trinidad • Tschechien • Türkei • Tunesien • Ungarn • Uruguay • Venezuela • Vereinigte Arabische Emirate • Vereinigte Staaten • Volksrepublik China • Zypern

Rockwell Automation weltweite Hauptverwaltung, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
Rockwell Automation Hauptverwaltung Europa, Avenue Herrmann Debrouxlaan, 46, 1160 Brüssel, Belgien, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Rockwell Automation Hauptverwaltung Asien/Pazifik, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hongkong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publikation 1746-6.3DE – November 1994

PN 40072-010-03(A)