MC-1A Befehlsdokumentation

Befehlsreferenz für die Programmierung des MC100-Systems in der MC-1A Sprache

Stand dieses Dokuments: März 2003

MC-1A Befehlsdokumentation

Befehlsreferenz für die Programmierung des MC100-Systems in der MC-1A Sprache

Waldweg 55, 88690 Uhldingen, Deutschland

Telefon +49-7556-9218-0, Telefax +49-7556-9218-50

E-Mail: technik@microdesign.de

http://www.microdesign.de

We like to move it!™

Inhaltsverzeichnis

n Raum für Ihre Notizen

Kapitel 1 Einführung

Wir gratulieren Ihnen zu Ihrer Entscheidung, Ihr Projekt mit einem MC100 System und der MC-1A Programmiersprache zu entwickeln! Denn das MC100 System ist eine umfassende Steuerungsfamilie, die Ihnen zugleich eine enorme Flexibilität – gepaart mit einfacher Handhabung – als auch eine Fülle von Erweiterungsmöglichkeiten bietet, die nahezu alle denkbaren Anwendungen abdecken sollte.

n Das MC100 System

Die MC100 ist ein modulares Steuerungssystem in 19"-Einschub-oder Gehäuse-Technik. Das MC100 System kann mit bis zu 8 Schritt- und 8 Servomotorachsen ausgebaut werden und mit SPS-Komponenten, wie EA-Module, analoge Ein-Ausgangsmodule, schnelle Zähler oder einem Temperaturregler erweitert werden. Nur einige der wichtigsten Merkmale des MC100 Systems in Kürze:

- **n** Durch die offene SPS-Struktur der MC-Systeme können die Schritt- oder Servomotorachsen gleichzeitig oder unabhängig voneinander verfahren und simultan komplette Maschinen und Anlagen steuern.
- **n** Für die MC100 steht verschiedene SPS-Software zur Verfügung, wie z.B. Software für Palettierer oder für Rundschalttische.
- **n** Die MC100 kann mit 2- oder 5-Phasen-Schrittmotorleistungsteilen in den verschiedenen Leistungsbereichen geliefert werden.
- **n** Als Servomotorantriebe können sowohl Gleichstrommotore, bürstenlose Synchronmotore oder AC-Servomotore eingesetzt werden.

1.1 Über diese Dokumentation

Die hier vorliegende Dokumentation ist logisch in folgende Kapitel gegliedert:

Kapitel 1 - Fehler! Ungültiger Eigenverweis auf Textmarke. **(ab Seite 3)**

Dieses Kapitel lesen Sie gerade. Es gibt Ihnen einen kurzen Überblick über das MC100 System und den Aufbach dieser Dokumentation. Außerdem finden Sie hier wichtige Hinweise wie Sie am besten mit dem Handbuch umgehen, damit Sie die gewünschten Information auch schnell auffinden können.

Kapitel 2 - Programmieren mit MC-1A (ab Seite 3)

Im zweiten Kapitel wenden wir uns der grundsätzlichen Idee der MC-1A Programmierung zu. Dieses Kapitel ist insbesondere dann für Sie interessant, wenn Sie bislang noch keine Programme in MC-1A entwickelt haben. Hier erfahren Sie viel über die Struktur der Programme, die Bearbeitung von Projekten, wie Programme in der Steuerung ausgeführt werden und vieles mehr.

Kapitel 3 - Befehlsübersicht (ab Seite 3)

Das wohl wichtigste Kapitel dieser Dokumentation ist die Befehlsreferenz. Hier finden Sie alle innerhalb der MC-1A Sprache verfügbaren SPS-Befehle und Makros einmal nach Funktionsgruppen und einmal alphabetisch sortiert. Zu jedem Befehl wird auch ein Beispiel für dessen Anwendung gegeben.

Fehler! Verweisquelle konnte nicht gefunden werden. **-** Fehler! Verweisquelle konnte nicht gefunden werden. **(ab Seite** Fehler! Textmarke nicht definiert.**)**

In diesem Kapitel wird die Programmierung spezieller Module sowie die Anwendung von Sonderfunktionen beschrieben. Hier finden Sie z.B. die Programmierung von Displays, die Einbindung serieller Erweiterungsmodule oder analoger Ein-/Ausgangsmodule, aber auch eine Beschreibung der erweiterten Messfunktionen für die MC100 Familie.

Fehler! Verweisquelle konnte nicht gefunden werden. **-** Fehler! Verweisquelle konnte nicht gefunden werden. **(ab Seite** Fehler! Textmarke nicht definiert.**)**

In diesem Kapitel erfahren Sie alles, was Sie über die Parametrierung des MC100 Systems wissen müssen. In erster Linie geht es hier natürlich um Achsparameter – aber auch die Konfiguration der seriellen Erweiterungsmodule oder einiger Systemdaten finden sich in diesem Kapitel.

Fehler! Verweisquelle konnte nicht gefunden werden. **-** Fehler! Verweisquelle konnte nicht gefunden werden. **(ab Seite** Fehler! Textmarke nicht definiert.**)**

Was wir hier ganz nüchtern mit "Systemdaten" umschrieben haben ist eine für Sie sehr nützliche Übersicht aller Systemvariablen- und Merker. Durch diese Daten erhalten Sie Informationen über den aktuellen Zustand des Systems und der angeschlossenen Geräte. Die beinhaltet z.B. die angeschlossenen Achsen, Displays, Analog I/O-Module oder Zähler. Natürlich enthält dieses Kapitel auch die Variablen und Merker die Sie innerhalb Ihres SPS-Programms immer wieder verwenden, wie z.B. die Ergebnisvariablen.

Fehler! Verweisquelle konnte nicht gefunden werden. **-** Fehler! Verweisquelle konnte nicht gefunden werden. **(ab Seite** Fehler! Textmarke nicht definiert.**)**

Nach so viel Theorie geht es hier dann wirklich zur Sache: Anhand von Beispielen wird beschrieben wie Sie am schnellsten zu brauchbaren Ergebnissen mit der MC-1A Sprache kommen.

Anhänge (ab Seite Fehler! Textmarke nicht definiert.**)**

In den Anhängen finden Sie eine Anzahl von Übersichten, wie z.B.

- ⇒ eine Wahrheitstabelle für logische Verknüpfungen
- ⇒ ein Tabellenverzeichnis,
- ⇒ eine Abbildungsübersicht,
- ⇒ einen Index der benötigten PC-Software

⇒ und einiges mehr.

n Struktur und Nummerierung

Zur besseren Übersicht haben wir darauf verzichtet jede Überschrift mit einer Kapitelnummer zu versehen. Lediglich die wichtigsten Abschnitte sind mit einer Kapitelnummer gekennzeichnet. Falls Sie spezielle Informationen zu einem Thema suchen, verwenden Sie am besten das Inhaltsverzeichnis am Anfang dieser Dokumentation.

n Formatierung in dieser Dokumentation

Damit Sie sich in dieser Dokumentation schnell zurechtfinden können, werden spezielle Informationen stets durch eine besondere Formatierung gekennzeichnet. Wenn Sie sich mit dieser Formatierung vertraut machen, werden Sie sich wesentlich einfacher innerhalb dieser Dokumentation zurechtfinden können.

Wichtige Hinweise

Besonders wichtige Informationen, wie die grundsätzliche Syntax eines MC-1A Befehls, werden stets durch **Fettdruck** und eine Kennzeichnung am linken Rand hervorgehoben, z.B.:

LAD_VV Variable, Variable

Beispiele

Die in dieser Dokumentation häufig anzutreffenden Programmbeispiele sind durch eine `andere Schrift` und durch eine Wellenlinie am linken Rand gekennzeichnet, so z.B.:

```
LAD_M      M_TEST1                // Wenn M_TEST1
UND_M      M_TEST2                // und M_TEST2 eingeschaltet sind,
SPRINGJ    TESTZYKLUS             // dann springe zu Label TESTZYKLUS
```

Tabellen

Wenn viele Informationen auf einmal dargestellt werden müssen verwenden wir in dieser Dokumentation Tabellen. Eine Übersicht aller Tabellen finden Sie im Anhang dieses Handbuchs.

n Dokumentation der PC-Software

Die vollständige Dokumentation der PC-Software VMC Workbench finden Sie als Online-Dokumente bzw. als Windows-Hilfedateien auf der VMC Workbench CD. Eine gedruckte Fassung der VMC Workbench Dokumentation ist nicht verfügbar.

Kapitel 2 Programmieren mit MC-1A

Wer schon einmal in SPS-Sprachen programmiert hat, dem wird die MC-1A Sprache auf Anhieb bekannt vorkommen: denn die grundsätzliche Struktur ähnelt sehr bekannten SPS-Dialekten, die in Form von Anweisungslisten (AWL) programmiert werden. Grundsätzliche Merkmale solcher Sprachen sind stets:

- Pro Zeile ist nur ein Befehl erlaubt
- Es gibt die Möglichkeit einer bedingten Befehlsausführung
- Es gibt die Datentypen Merker, Variable, Eingänge, Ausgänge und Konstante
- Abfrageergebnisse werden in Statusregistern zurückgeliefert

Das alles trifft auch auf die MC-1A Sprache zu. Deshalb wollen wir uns an dieser Stelle nicht mit den Grundlagen der SPS-Programmierung als solches beschäftigen, sondern ganz gezielt auf die Besonderheiten des MC-1A Systems eingehen.

Warum nicht IEC1131?

Wir haben lange darüber nachgedacht ob wir für das MC100 System eine IEC1131-kompatible Programmiersprache entwickeln. Letzten Endes haben wir uns dagegen entschieden um die vielen Vorteile, wie die einfache Integration von Achsen oder Zusatzaggregaten, die sehr schnelle Ausführung der Programme und mehr, nicht zu untergraben. Wir sind überzeugt, daß ein erfahrener SPS-Programmierer mit der MC-1A Sprache wesentlich schneller zu verwendbaren Resultaten kommt als mit IEC1131 und einem Neueinsteiger, durch die logische Struktur der Sprache in sich, der Anfang sehr leicht gemacht wird.

2.1 Grundsätzliche Vereinbarungen

Zunächst wollen wir die Grundlagen der MC-1A Sprache einmal definieren. Dies beinhaltet z.B. wie Befehle geschrieben werden müssen oder wie Kommentare zu deklarieren sind. Wenn Sie bislang noch nicht mit der MC-1A Sprache gearbeitet haben, dann sollten Sie sich unbedingt mit diesem Abschnitt beschäftigen.

n Sprachdefinition

In der MC-1A Sprache erfolgt die Programmierung grundsätzlich in der Form einer Anweisungsliste in folgendem festen Format:

SPS-Befehl Parameter

Die Anzahl der Parameter hängt vom jeweiligen Befehl ab. Es gibt auch SPS-Befehle die keinerlei Parameter benötigen.

Ebenfalls vom jeweiligen Befehl abhängig ist der Datentyp des oder der Parameter. Zumeist der erwartete Datentyp bereits aus dem Befehl selbst ersichtlich. Hier einige Beispiele:

```
LAD_M 1                 // LAD_M heißt: LADe Merker. Der erwartete
                        // Parameter ist deshalb immer ein Merker,
                        // in diesem Fall der Merker 1.
EIN_A 12                // EIN_A heißt: EINschalten Ausgang. Der
                        // Parameter muß deshalb ein Ausgang sein, in
                        // diesem Fall der Ausgang 12.
SPRING Schleife         // SPRING verzweigt die Ausführung des Programms
                        // und erwartet deshalb als Parameter immer ein
                        // Label (eine Sprungmarke).
```

Parameter-Übergabe

Bei den meisten Befehlen ist es möglich direkt eine Nummer für die entsprechende Resource anzugeben, wie in obigen Beispiel die "12" für den Eingang 12 oder die "1" für den Merker 1. Die Bedeutung des Parameters ergibt sich aus dem Befehl.

Symbolische Namen

Meistens werden Sie jedoch mit symbolischen Namen für die Resourcen arbeiten, denn die MC-1A Sprache unterstützt die Vergabe von freien Namen für Variablen, Merker, Konstanten sowie Ein- und Ausgänge. Hierzu weisen Sie am Anfang Ihres Programms oder in einer speziellen Definitionsdatei die jeweils gewünschten Namen zu. Dadurch ergibt sich eine wesentlich bessere Lesbarkeit Ihrer Programme. Wir wollen einmal obiges Beispiel mit symbolischen Namen darstellen:

```
DEF_M 1,M_START         // Dem Merker 1 den Namen "M_START" zuweisen
DEF_A 12,A_LAMPE        // Dem Ausgang 12 den Namen "A_LAMPE" zuweisen

LAD_M M_START           // Merker M_START (= Merker 1) laden
EIN_A A_LAMPE           // Ausgang A_LAMPE (= Ausgang 12) setzen
```

Einschränkungen für symbolische Namen

Bitte beachten Sie folgende Einschränkungen für symbolische Namen:

- Der symbolische Name darf nicht länger als 23 Zeichen sein.
- Der Name darf nur die Buchstaben von A-Z sowie Unterstriche, Bindestriche und die Ziffern 0-9 enthalten. Alle anderen Zeichen, wie z.B. das Leerzeichen, Umlaute oder sonstige Sonderzeichen, sind innerhalb symbolischer Namen nicht erlaubt.
- Die Groß- und Kleinschreibung wird in symbolischen Namen ignoriert.

n Kommentare

Kommentare oder Hinweise zum Ablauf des Programms erleichtern die Lesbarkeit des Quelltextes. Deshalb sollten Sie mit entsprechenden Erläuterungen nicht zu sparsam umgehen. In den bisherigen Beispielen haben Sie bereits gesehen, daß wir hinter jedem Befehl entsprechende Kommentare eingefügt haben und zwar jeweils mit der Zeichenkette "//" beginnend. Dies stellt eine der Möglichkeiten dar Kommentare in Ihren Quelltext einzuflechten.

Rest der aktuellen Zeile als Kommentar markieren

Wenn Sie entweder die Zeichenkette "//" oder das Zeichen ";" (Strichpunkt) in Ihrem Quelltext verwenden, wird der Rest der jeweiligen Zeile als Kommentar markiert und beim Programmablauf nicht berücksichtigt.

Befehl Parameter //Komentar
Befehl Parameter ; Kommentar

Dies ist die übliche Form einen Quelltext zu kommentieren: hinter jeden SPS-Befehl eine Erläuterung zu setzen, die erklärt, was an dieser Stelle gemacht werden soll.

Längere Kommentare

Wenn Sie einen längeren Kommentar einfügen möchten, z.B. um grundsätzliche Funktionen zu erklären, oder Aufgaben zu markieren die noch durchgeführt werden müssen, können Sie einen Block von mehreren Zeilen als Kommentar kennzeichnen. Hierzu markieren Sie den Anfang des Kommentars mit der Zeichenkette "/*". Alles was nach dieser Zeichenkette kommt, wird vom Compiler als Kommentar behandelt und beim Programmablauf nicht berücksichtigt. Um den Kommentar zu beenden verwenden Sie die Zeichenkette "*/". Danach wird der Quelltext vom Compiler wieder berücksichtigt.

/* Kommentar */

Beispiel für die Verwendung von Kommentaren

```
/* Hier beginnt unser Kommentar. Wir können nun mehrere Zeilen Erläuterung
zu dem Programm schreiben. Alles, was innerhalb des Kommentars steht, wird vom
Compiler ignoriert. */

Start:                          // Label (Sprungmarke)
          LAD_M M_START         ; Merker M_START prüfen
          SPRINGN START         /* Zurück zum Anfang */
```

In dem oben gezeigten Beispiel sehen Sie alle Möglichkeiten, Kommentare in Ihren Quelltext einzuflechten, auf einen Blick. Bitte beachten Sie, daß Sie jeden Kommentar den Sie mit "/*" beginnen, auch mit "*/" abschließen müssen!

Kommentare werden bei der Compilierung Ihres SPS-Programms nicht berücksichtigt und auch nicht in die Steuerung übertragen. Deshalb belegen Kommentare auch keinerlei Speicherplatz innerhalb der Steuerung.

n Labels (Sprungmarken)

In dieser Dokumentation sprechen wir grundsätzlich von Labels, meinen damit aber natürlich genauso Sprungmarken. Labels ist lediglich der englische Begriff für die gleiche Sache. Da sich die Bezeichnung "Label" jedoch im Allgemeinen technischen Sprachgebrauch eingebürgert hat, bleiben wir künftig auch bei diesem Begriff.

Mit einem Label definieren Sie eine bestimmte Stelle innerhalb Ihres SPS-Programms, zu der Sie von einer anderen Stelle des Programms aus verzweigen möchten. Dies kann z.B. der Fall sein, wenn Sie

- **n** eine Schleife programmieren möchten,
- **n** den Anfang eines Unterprogramms deklarieren möchten oder
- **n** je nach Ergebnis einer Abfrage unterschiedliche Abläufe ausführen möchten.

Labels geben also einer bestimmten Stelle im Programm einen Namen, den Sie in Ihrem SPS-Programm jederzeit verwenden können. Dies wird anhand der folgenden Beispiele deutlicher:

Programmierung einer Schleife

```
Start:                          // Definiert das Label "Start". Diese Zeile
                                // Ihres SPS-Programms heißt also in Zukunft
                                // einfach "Start".
          LAD_E E_SteuerungEin  // Wir fragen den Eingang E_SteuerungEin ab,
                                // dem mit dem Schlüsselschalter zu Einschalten
                                // der Steuerung verbunden ist
          SPRINGN Start         // Wenn der Eingang nicht aktiv ist, springen
                                // wir zurück zur Programmzeile, in der das
                                // Label "Start" steht.
```

Programmierung eines Unterprogramms

```
Haupt:                          // Definiert das Label "Start" für die aktuelle
                                // Programmzeile
          GEHUPR Achsen         // Rufe das Unterprogramm "Achsen" auf
          SPRING Haupt          // Zurück zum Label "Haupt"

Achsen:                         // Definiert das Label "Achsen" für die aktuelle
          ...                   // Zeile. Dieses Label wird für den Aufruf des
          ...                   // Unterprogramms verwendet.
          ...                   // Ablauf des Unterprogramms
          UPREND                // Unterprogramm beenden
```

Programmierung einer bedingten Verzweigung

```
          LAD_M M_Display       // Frägt einen Merker ab, ob die Anzeige
                                // Aktualisiert werden soll
          SPRINGN Weiter        // Wenn nicht, springe zu Label weiter
          ...                   // Hier der Quellcode für die Anzeige
Weiter:                         // Definiert das Label "Weiter"
          ...                   // Weiterer Programmablauf
```

Einschränkungen für Labelnamen

Bitte beachten Sie folgende Einschränkungen für Labelnamen:

- **n** Der Labelname darf nicht länger als 23 Zeichen sein.
- **n** Der Name darf nur die Buchstaben von A-Z sowie Unterstriche, Binderstriche und die Ziffern 0-9 enthalten. Alle anderen Zeichen, wie z.B. das Leerzeichen, Umlaute oder sonstige Sonderzeichen, sind innerhalb von Labelnamen nicht erlaubt.
- **n** Die Groß- und Kleinschreibung wird in Labelnamen ignoriert.

n Makros

Die MC-1A Sprache unterstützt die Verwendung von Makros. Dies bedeutet, daß Sie sich häufig verwendete Abläufe einmal innerhalb eines Makros definieren und ab da nur noch das Makro aufrufen. Dies verschafft Ihnen eine größere Übersicht während der Programmierung; jedoch können Sie diese Methode nur dann anwenden, wenn die jeweiligen Abläufe tatsächlich exakt identisch sind. Hier einmal ein Beispiel für ein Makro:

```
TOGGLE_A   MACRO Ausgang          // Anfang der Makrodefinition
           NLAD_A Ausgang         // Invertierten Status des Ausgangs laden
           MOD_A Ausgang          // Zustand des Ausgangs umschalten
           ENDM                   // Ende der Makrodefinition

// Makro im Programm verwenden

           TOGGLE_A 14            // Ausgang 14 umschalten
           TOGGLE_A A_Blinklicht  // Ausgang A_Blinklicht umschalten
```

Eine Übersicht aller Makro-Befehle sowie weitere Erläuterungen zu diesem Thema finden Sie im Kapitel 3.1 - Befehle nach Funktionsgruppen unter "Makros" (Seite 3).

n Struktur des Quelltext-Projekts

Ihr MC-1A Projekt kann prinzipiell aus beliebig vielen einzelnen Dateien bestehen. Mit Hilfe der mitgelieferten Entwicklungsoberfläche VMC Workbench können Sie jederzeit neue Quelltextdateien zu Ihrem SPS-Projekt hinzufügen oder auch Dateien aus dem Projekt entfernen.

Normalerweise verwendet man innerhalb eines MC-1A Projekts zumindest vier verschiedene Dateien für folgende Aufgaben:

- ⇒ SPS-Quelltext, enthält das eigentliche Programm
- ⇒ Definitionsdatei, enthält symbolische Definitionen und Konstanten
- ⇒ Makrodatei, enthält Makrodefinitionen
- ⇒ SPS-Textdatei, enthält Text für die Benutzerführung

Natürlich können Sie alle diese Aufgaben auch in einer einzigen Datei zusammenführen; damit Sie jedoch die Übersicht über Ihr Projekt behalten empfehlen wir, daß Sie für jede Aufgabe eine getrennte Datei verwenden.

In der Praxis geht dies zumeist noch weiter: Für größere Projekte verwendet man in der Regel eine Vielzahl einzelner Dateien, von denen jede ein spezielles Aggregat der Maschine oder aber einen speziellen Ablauf beinhaltet. So ist es z.B. üblich, die Initialisierungsroutinen in einer getrennten Datei zu schreiben, ebenso die Achsverwaltung, die Störungsverwaltung usw. Sie erhalten somit eine bessere Übersicht über Ihr Projekt. Eine strikte Trennung ermöglicht Ihnen auch eine einfache modulare Programmierung und dadurch die schnelle Anpassung von Serienmaschinen an bestimmte Aufgaben.

Weitere Informationen zur Verwaltung von SPS-Projekten entnehmen Sie bitte der Online-Dokumentation Ihrer Entwicklungsoberfläche, dem VMC Workbench Studio.

n Modulare Programmierung

Wenn Sie für Ihr SPS-Projekt von Anfang an ein durchgängiges Konzept entwickeln, dann können Sie sich auf einfache Art und Weise ein Standard-Projekt erstellen, welches Sie für spezielle Maschinenvarianten meist nur noch an bestimmten Stellen anpassen müssen. Grundlage hierfür ist zum Einen eine modulare, durchdachte Auslegung Ihres Programms und natürlich die Fähigkeit der MC-1A Sprache zur bedingten Compilierung.

Nehmen wir einmal an, Sie fertigen eine Serienmaschine, die in mehreren Ausführungen erhältlich ist. So bieten Sie einmal eine Basismaschine an, die über keine integrierte Anzeige verfügt. Die nächstgrößere Variante Ihrer Maschine verfügt über ein Display, aber über keine Warnlampe mit Sirene. Erst die größte Ausführung Ihrer Maschine enthält alle diese Optionen.

Bei der Planung Ihres SPS-Progamms programmieren Sie jetzt als erstes die größte Maschine, die alle denkbaren Optionen enthält. Achten Sie dabei darauf, daß Sie alle Funktionen, die bei den kleineren Varianten nicht erhältlich sind, logisch von den restlichen Funktionen der Maschine abtrennen. Es macht keinen Sinn, wenn Sie die Display-Verwaltung immer wieder im Ablauf ansprechen. Besser ist es, den entsprechenden Quellcode deutlich auszulagern.

Sobald das Projekt für die größte Variante Ihrer Maschine fertig ist, beginnen Sie nun, die Optionen, die bei den kleineren Ausführungen nicht erhältlich sind, systematisch herauszunehmen. Dies geht am einfachsten mit der bedingten Compilierung.

Festlegen von Konstanten für die Optionen

Legen Sie für jede Option, die möglicherweise nicht an einer kleineren Ausführung Ihrer Maschine erhältlich ist, eine Konstante an. Diese Konstante soll später entscheiden welcher Quelltext in Ihrem Projekt verwendet wird.

```
          DEF_W 1,OptionDisplay   // 0 = Kein Display, 1 = Display vorhanden
          DEF_W 0,OptionSirene    // 0 = Keine Sirene, 1 = Sirene vorhanden
```

Verwenden der Konstanten im Programm

Im nächsten Schritt bauen Sie Schalter für die bedingte Compilierung an allen Stellen ein, die z.B. auf die Anzeige oder die Sirene zugreifen:

```
          ...                     // Wir steigen mitten im Ablauf ein
          LAD_M M_Stoerung        // Stoerung aufgetreten?

#IF OptionSirene EQ 1             // Nur wenn die Option "Sirene" gesetzt ist
          EIN_A A_Sirene          // wird der Befehl zum Einschalten der Sirene
#ENDIF                            // mit dem Quelltext compiliert

#IF OptionDisplay EQ 1            // Nur wenn die Option "Display" gesetzt ist
          LAD_VA V_Text,V_Stoer   // laden wir die Nummer der Störung in unsere
          GEHUPR StoerungAnz      // Textvariable und zeigen die entsprechende
#ENDIF                            // Störung im Display an
```

Sobald Ihr Programm die entsprechenden Schalter für die bedingte Compilierung enthält, brauchen Sie in Zukunft nur noch die Werte für die Konstanten "OptionDisplay" und "OptionSirene" zu ändern, um das Projekt für die jeweilige Maschinenvariante zu erstellen.

Sondermaschinen mit modularer Programmierung

Besonders mächtig wird diese Form der Programmierung dann, wenn Sie spezielle Anpassungen für Sondermaschinen programmieren müssen. Verwenden Sie einfach Ihr Standardprogramm und fügen Sie die speziellen Ergänzungen, stets umrahmt von Schaltern, für bedingte Compilierung ein. So behalten Sie stets eine aktuelle Version Ihres Quelltextes, der sich mit der Zeit über die entsprechenden Schalter für nahezu alle Gegebenheiten anpassen läßt.

Eine Übersicht zum Thema "Bedingte Compilierung" finden Sie im Kapitel 3.1 - Befehle nach Funktionsgruppen unter "Compilieranweisungen" (Seite 3).

2.2 Datentypen

Im MC-1A System gibt es insgesamt sechs unterschiedliche Datentypen, die im Folgenden erläutert werden:

n Variablen

SPS-Variablen sind im MC-1A System stets 24 Bit breit, dies bedeutet folgende Limitierungen:

⇒ Größter möglicher Wert: 8388607

⇒ Kleinster möglicher Wert: - 8388608

Im MC-1A System können nur ganze Zahlen gespeichert werden. In der Fachsprache nennt man dies "Integer-Zahlen". Kommastellen, oder sogenannte "echte Zahlen", werden nicht unterstützt. Wenn Sie Zahlen mit Nachkommastellen verwalten müssen, dann multiplizieren Sie den Wert sooft mit dem Faktor 10, bis eine ganze Zahl dabei herauskommt.

Variablen können mit MC-1A universell eingesetzt werden. Es gibt grundsätzlich keine Limitierung dafür was tatsächlich in einer Variable enthalten sein muß. So können Variablen einen normalen Wert enthalten, einen Text repräsentieren oder aber ein Zeiger auf anderen Daten sein. Für die Arbeit mit Variablen steht Ihnen ein komplexer Satz an Befehlen zur Verfügung, der von einfachen Lade- und Vergleichsoperationen bis hin zu arithmetischen Berechnungen reicht.

n Merker

SPS-Merker können nur 1 Bit Informationen speichern. Merker können also nur den Zustand "ein" oder "aus" annehmen.

Der Einsatz von Merkern empfiehlt sich immer dann, wenn nur eine einzelne Status-Information gespeichert oder an einen anderen Programmteil weitergegeben werden muß. Die Vorteile der Programmierung mit Merkern sind:

⇒ Einfache Programmierung ohne Konstanten oder Zahlenwerte

⇒ Schnelle Auswertung der Bedingungen im SPS-Programm

⇒ Einfachste Verknüpfung mehrerer Bedingungen

Häufig verwendet man Merker um den grundsätzlichen Zustand eines Ablaufs zu speichern, oder andere Funktionen innerhalb eines Ablaufs zu aktivieren. Natürlich können dabei auch weitere Informationen in einer Variable gespeichert werden.

So ließe sich eine Störungsverwaltung in der Form programmieren, daß die Störungsüberwachung sowohl die Fehlercode in eine Variable schreibt, als auch einen generellen Merker setzt um zu signalisieren: Es ist eine Störung aufgetreten. Im Hauptprogramm wird dabei stets nur der Störungsmerker abgefragt, anstatt jedes mal den Wert der Störungsvariable zu überprüfen. Erst wenn der Merker gesetzt wurde, wird eine genaue Prüfung des Fehlercodes vorgenommen.

n Ausgänge

Ähnlich wie die SPS-Merker enthält ein Ausgang nur die Information "ein" oder "aus". Jedoch repräsentiert dieser Datentyp immer auch einen wirklich existierenden, digitalen SPS-Ausgang. Das Verändern eines Ausgangs ändert den jeweiligen Ausgang sofort – unabhängig vom Ablauf des SPS-Programms.

n Eingänge

Eine Variable des Datentyps "Eingang" repräsentiert einen tatsächlich vorhandenen, digitalen Eingang. Der Wert wird dabei fortlaufend aktualisiert.

n Konstanten

Eine Konstante definiert einen symbolischen Namen als eine Zahl. Im SPS-Programm kann dieser symbolische Name dann verwendet werden, um bei Befehlen die konstante Zahlenwerte erwarten, statt der Zahl einen Klartextnamen anzugeben.

Besonders sinnvoll ist der Einsatz von Konstanten dann, wenn Sie den gleichen Wert an mehreren Stellen im Programm verwenden, z.B. um die Geschwindigkeit einer Positionierbewegung anzugeben. Statt im Falle einer Änderung dann an mehreren Stellen den Zahlenwert auszutauschen, müssen Sie dann nur noch den Wert bei der Definition der Konstante verändern.

n Labels

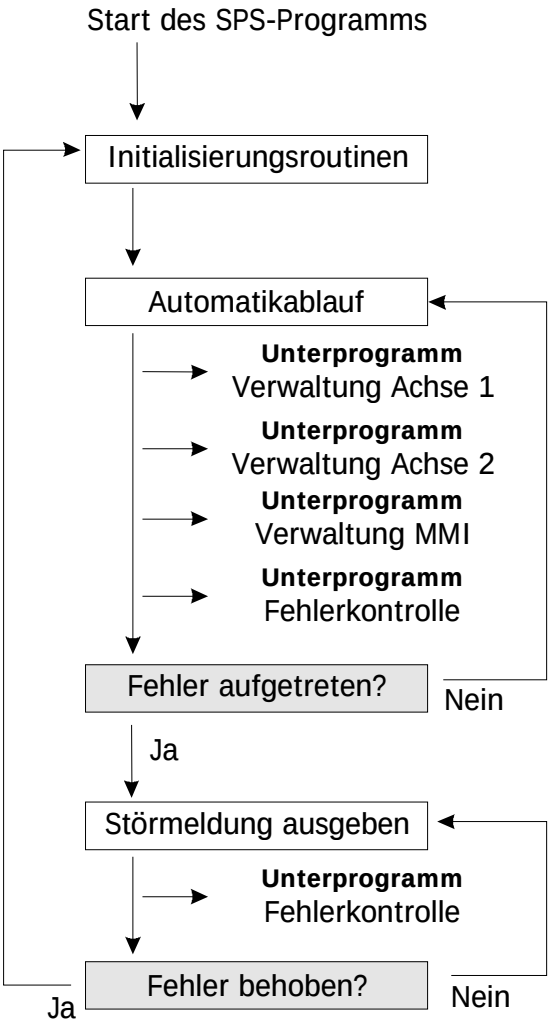
Ein Label stellt eine bestimmte Stelle Ihres SPS-Programms dar, oder in anderen Worten, eine Programmadresse. Labels werden in der Regel nur zusammen mit Sprungbefehlen verwendet. Sie können jedoch auch die Programmadresse eines Labels in eine Variable übertragen, um eine dynamische Zyklusprogrammierung zu erstellen.

2.3 Zyklen, Abläufe und Tasks

Das MC100 System ist von Haus aus eine Mehrtask-Maschine, d.h. es können mehrere Abläufe voneinander unabhängig bearbeitet werden. Diese Möglichkeit wird von der MC-1A Programmiersprache in vollem Umfang unterstützt. Noch wichtiger ist hingegen zu verstehen, daß das MC100 System nicht als eine Zyklus-Maschine im eigentlichen Sinn arbeitet. Im Gegensatz zu einer Zyklusmaschine arbeitet die MC100 stets asyncron; sämtliche Betriebszustände – wie z.B. der Zustand von Ein- und Ausgängen – werden fortlaufend aktualisert, auch die Programmausführung ist nicht an einen festen Zyklus gebunden.

n Wie läuft ein Programm in MC-1A ab?

Die wichtigste Grundlage, die man sich bei der Programmierung eines Systems mit MC-1A verinnerlichen muß, ist: es wird nicht mit einem festen Zyklus-Ablauf gearbeitet! Mit MC-1A können Sie die Art und Weise, wie das SPS-Programm abgearbeitet werden soll, selbst bestimmen:



n Abbildung 1 – Typischer Ablauf eines SPS-Programms in MC-1A

Wie man in diesem Beispiel sieht, gibt es mit MC-1A nicht nur einen Ablauf. Sie können Ihr Hauptprogramm beliebig schachteln, Programmschleifen bilden, von einem Programmteil in einen ganz anderen springen, Unterprogramme aufrufen (die natürlich auch wieder Abläufe bzw. Zyklen nachbilden können) und vieles mehr.

Obiges Beispiel als MC-1A Programm

Wenn wir obigen Ablauf einmal in der MC-1A Sprache darstellen wollten, dann könnte das wie folgt aussehen:

```
Start:                          // Anfang des Programms
          GEHUPR Willkommen     // Einschalt-Nachricht auf dem Display ausgeben
                                // Funktion wird als Unterprogramm aufgerufen

Init:                           // Label (Sprungmarke) Initialisierungsroutine
                                // Ruft die Unterprogramme auf, die zur
                                // Initialisierung des SPS-Programms dienen
          GEHUPR MemLoeschen    // Löscht interne Daten des letzten Durchlaufs
          GEHUPR Referenzfahrt  // Führt Referenzfahrt für alle Achsen durch

Automatik:                      // Label (Sprungmarke) für Automatikablauf
          GEHUPRI Achse1Verwalt // Unterprogramm Verwaltungs Achse 1
          GEHUPRI Achse2Verwalt // Unterprogramm Verwaltungs Achse 2
          GEHUPRI StatusAnzeige // Unterprogramm zur Anzeige des Status auf
                                // dem Display
          GEHUPRI FehlerTest    // Prüfe, ob Störungen aufgetreten
          LAD_M M_Stoerung      // Prüfen, ob das Unterprogramm eine Störung
                                // zurückgemeldet hat
          SPRINGN Automatik     // Keine Störung, weiter mit Automatik-Ablauf

Stoerung:                       // Label (Sprungmarke) für Störungsverwaltung
          GEHUPRI FehlerMeldung // Störungsmeldung auf der Anzeige ausgeben
          GEHUPRI FehlerTest    // Prüfen, ob Störung immer noch vorhanden
          SPRINGJ Stoerung      // Ja, wir warten weiter auf Behebung
          SPRINGN Init          // Nein, Störung behoben, neu initialisieren
```

Sie sehen, ein SPS-Programm kann durchaus einfach und übersichtlich aussehen. Vielleicht ist Ihnen auch aufgefallen, daß alle Befehle der deutschen Sprache entstammen; wo immer möglich, haben wir die Befehle aus deutschen Begriffen zusammengesetzt. So heißt der Befehl GEHUPR z.B. "GEHe UnterPRogramm", GEHUPRI steht für "GEHe UnterPRogramm Immer", SPRINGJ bedeutet "SPRINGe wenn Ja" usw.

n Verwenden von Tasks

Noch leistungsfähiger stellt sich das Programmablauf-Konzept der MC-1A Sprache dar, wenn Sie nicht nur mit Unterprogrammen, sondern auch mit Tasks programmieren.

Wir wollen eine "Task" einmal als ein unabhängiges, zusätzliches Hauptprogramm definieren, welches ohne direkten Zusammenhang mit den anderen "Tasks" (oder eben Hauptprogrammen) abläuft. Häufig wird hierfür auch der Begriff "Parallelprogramm" verwendet.

Das Verwenden von Tasks bietet Ihnen also zusätzliche, neue Möglichkeiten: wenn Sie – z.B. bei der Verwaltung einer Achsbewegung – im Hauptprogramm darauf warten, daß die Achse ihre Bewegung durchgeführt hat, also in Position ist, haben Sie ohne Tasks zwei Möglichkeiten der Programmierung:

- Sie programmieren im Sinne der klassischen Zyklus-Maschine, also unter Verwendung von Hilfsmerkern- und Variablen. Der Nachteil ist, daß Sie dafür jedesmal Ihren vollständigen Zyklus durchlaufen müssen und natürlich entsprechende Vorkehrungen in Ihrem Programm treffen müssen.
- Sie programmieren eine Warteschleife, die im SPS-Programm darauf wartet, daß die Bedingung erfüllt ist (in diesem Fall: die Achse ihre Zielposition erreicht hat). Der Nachteil hierbei ist natürlich, daß Ihr Programm solange nichts anderes machen kann. Nicht nur, daß an einer komplexeren Maschine ja nicht nur eine einzige Achse zu verwalten ist. Zudem können Sie auch nicht ohne weiteres auf eine Störung reagieren, denn Ihr Programm "hängt" ja in einer Warteschleife.

Eine Task die Arbeit machen lassen

Einfacher wird das Ganze, wenn Sie entsprechend kritische oder zeitintensive Funktionen in Tasks auslagern. So kann z.B. die Task 2 beliebig lange in einer Schleife darauf warten, ob eine Bedingung erfüllt wird. Sie können sogar vollständige Aggregate, d.h. Bestandteile Ihre Maschinen, in getrennte Tasks auslagern. Das Hauptprogramm ist hiervon nicht betroffen und arbeitet den normalen Programmablauf einfach weiter ab.

Tasks sind einfach zu programmieren

Um Programmteile in eine Task auszulagern, schreiben Sie einfach in der MC-1A Sprache einen ganz normalen Ablauf. Um diesen Ablauf in eine Task zu verwandeln, genügt an einer beliebigen Stelle der Befehl:

```
LAD_P2 Ueberwachung
```

Schon würde der Programmablauf, der bei dem Label (der Sprungmarke) "Ueberwachung" beginnt, als eine separate, unabhängige Task in Ihrem SPS-Programm ablaufen – in diesem Beispiel als Task 2. Das einzige was Sie bei der Programmierung von Tasks beachten müssen ist, daß sich die einzelnen, voneinander unabhängigen Tasks nicht gegenseitig "in die Quere" kommen, also nicht Variablen oder Merker benutzen die gleichzeitig von einer anderen Task verwendet werden.

n Zyklen programmieren

Weil MC-1A Programme für sich genommen nicht als Zyklusmaschine laufen, müssen Sie – falls gewünscht – die Funktion einer Zyklusmaschine nachbilden. Die geht jedoch sehr einfach, indem Sie schlicht eine Zyklusvariable definieren und in dieser Variable den Zykluszähler speichern:

```
          DEF_V 200,V_ZYKLUS      // Variable 200 als Zyklusvariable definieren

Automatik:                        // Label (Sprungmarke) für Zyklusablauf
          LAD_VA V_ZYKLUS,1       // Mit erstem Zyklus beginnen

Zyklus1:                          // Label (Sprungmarke) für ersten Zyklus
          VERGL_VA V_ZYKLUS,1     // Wenn Zyklus 1 bereits abgearbeitet ist,
                                  // muß die Zyklusvariable größer als 1 sein
          NLAD_M M_GROESSER       // Vergleichsergebnis prüfen (wenn nicht größer,
                                  // ist das Bitergebnis gesetzt)
          SPRINGN Zyklus2         // Springe zu Zyklus2, wenn Bitergebnis aus

          ...                     // Ablauf für Zyklus 1

          LAD_VA V_Zyklus,2       // Zyklus-Variable auf nächsten Zyklus setzen

Zyklus2:                          // Label (Sprungmarke) für zweiten Zyklus
          VERGL_VA V_ZYKLUS,2     // Wenn Zyklus 2 bereits abgearbeitet ist,
                                  // muß die Zyklusvariable größer als 2 sein
          NLAD_M M_GROESSER       // Vergleichsergebnis prüfen (wenn nicht größer,
                                  // ist das Bitergebnis gesetzt)
          SPRINGN Zyklus3         // Springe zu Zyklus3, wenn Bitergebnis aus

          ...                     // Ablauf für Zyklus 1

          LAD_VA V_Zyklus,3       // Zyklus-Variable auf nächsten Zyklus setzen
```

Auch eine "klassische" Zyklusmaschine ist also mit der MC-1A Sprache sehr einfach zu realisieren. Noch besser funktioniert das Ganze, wenn Sie einige Tips beachten, die Ihnen die Programmierung und Verwaltung von Zyklusabläufen einfacher machen:

Tips und Tricks zur Zyklusprogrammierung

- **n** Lassen Sie Lücken in Ihren Zyklusbezeichnungen! Beginnen Sie beim ersten Entwurf des Ablaufs mit gut 20 freien Abläufen zwischen jedem Zyklus. Sie würden also hier dann nummerieren: Zyklus1, Zyklus21, Zyklus41 usw. Dies gibt Ihnen die Möglichkeit, zu einem späteren Zeitpunkt, wenn – wie fast immer – noch zusätzliche Abläufe oder Aggregate integriert werden müssen, einfach und unkompliziert die entsprechenden Befehle zwischen bereits bestehende Zyklen zu schieben.
- **n** Programmieren Sie Zyklen nicht mit Merkern! Zwar bieten sich diese 1 Bit breiten Variablen an sich geradezu für diesen Zweck an; wenn Sie jedoch eine Vielzahl von Abläufen verwalten müssen, wird das sehr schnell unübersichtlich.
- **n** Lassen Sie Ihre Zyklusmaschine in einem Unterprogramm oder einer Task laufen! Wenn Sie den Zyklusablauf unabhängig von der Verwaltung sonstiger Aggregate an Ihrer Maschine programmieren, lassen sich zusätzliche Überwachungen oder ein einfaches Mensch-Maschine-Interface wesentlich einfacher und auch wirkungsvoller einbinden.

n Zyklen mit dynamischen Sprüngen

Die MC-1A Sprache erlaubt Ihnen neben der klassischen Zyklusprogrammierung auch eine dynamische Verwaltung Ihres Ablaufs: statt nämlich in einer Zyklusvariable nur einen Zykluszähler zu speichern, können Sie hier stattdessen auch gleich das Sprungziel für den jeweiligen Zyklus ablegen:

```
          DEF_V 200, V_ZYKLUS     // Variable 200 als Zyklusvariable definieren
          LAD_VL V_ZYKLUS,Zyklus1 // Zyklusvariable mit Sprungmarke des ersten
                                  // Zyklusablaufs vorladen

Automatik:                        // Label (Sprungmarke) für Automatikschleife
          GEHUPRI Display         // Unterprogramm Anzeigeverwaltung aufrufen
          GEHUPRI Achsen          // Unterprogramm Achsenverwaltung aufrufen
          ...                     // Weitere Verwaltungs-Unterprogramme

// Jetzt kommt der entscheidende Schritt: Statt ein generelles Zyklusunterprogramm
// aufzurufen, springen wir gezielt in das Unterprogramm, auf das die Variable
// V_ZYKLUS zeigt

          GEHUPRV V_Zyklus        // Springe in das Unterprogramm, auf das die
                                  // Variable V_ZYKLUS zeigt
          SPRING Automatik        // Weiter mit der Automatikschleife

// Hier wollen wir nun einen typischen Zyklus für die dynamische Programmierung
// zeigen:

Zyklus1:                          // Label (Sprungmarke) für den ersten Zyklus
          ...                     // Programmcode für den Ablauf dieses Zyklus
          ...                     // Es sind an dieser Stelle keine weiteren
          ...                     // Prüfungen nötig, das Unterprogramm wird
          ...                     // nur aufgerufen, wenn dieser Zyklus auch
          ...                     // wirklich ausgeführt werden soll
          LAD_VL V_ZYKLUS,Zyklus2 // Zyklus-Variable auf die Sprungmarke des
                                  // nächsten Zyklus laden
          UPREND                  // Unterprogramm für Zyklus 1 beenden
```

Wie aus obigen Beispiel erkennbar ist, benötigen Sie für die Programmierung einer dynamischen Zyklusmaschine wesentlich geringeren Aufwand als bei der "klassischen" Variante: denn hier müssen Sie nicht zu Beginn eines jeden Ablaufs kontrollieren, ob auch tatsächlich dieser Zyklus aktiv ist: sobald das Zyklusunterprogramm aufgerufen wird, können Sie sicher sein, daß es auch ausgeführt werden soll.

2.4 Unser Ergebnisspeicher: das Bitergebnis

Die grundsätzliche Philosophie der MC-1A Sprache beinhaltet, daß einige Befehle nur dann ausgeführt werden, wenn der zentrale Ergebnisspeicher – das Bitergebnis – eingeschaltet ist. Dies ermöglicht Ihnen eine einfache, effektive und schnelle Programmierung, denn nach der Abfrage einer Bedingung müssen Sie nicht zwangsweise mit einem Sprung verzweigen. Die durch das Bitergebnis bedingte Ausführung dieser Befehle erlaubt Ihnen selbst komplexe Abfragen ohne einen einzigen Sprungbefehl.

Das Verständnis für die Funktionsweise des Bitergebnis und dessen Auswirkung auf den Programmablauf ist außerordentlich wichtig für das Programmieren mit der MC-1A Sprache. Bitte lesen Sie diesen Abschnitt deshalb sorgfältig durch.

n Wie wird das Bitergebnis beeinflußt?

Jede Abfrage eines bitorientierten Datentyps (Merker, Ausgänge und Eingänge) überträgt den Status der abgefragten Daten in das Bitergebnis. Wird ein ausgeschalteter Eingang abgefragt, ist hinterher auch das Bitergebnis ausgeschaltet. War der Eingang geschaltet, ist auch das Bitergebnis ein.

n Wie wirkt sich das Bitergebnis aus?

Alle vom Bitergebnis abhängigen Befehle werden nur dann ausgeführt, wenn das Bitergebnis den entsprechenden Zustand hat. Das Einschalten eines Ausgangs wird z.B. nur dann ausgeführt, wenn zum Zeitpunkt der Ausführung das Bitergebnis eingeschaltet ist. Einige Befehle, wie z.B. alle Sprungbefehle, sind in verschiedenen Varianten verfügbar, um auf den aktuellen Status des Bitergebnis zu reagieren.

Beispiel für das Arbeiten mit dem Bitergebnis

```
// Wir gehen jetzt einmal davon aus, daß der Eingang 1 nicht geschaltet ist

          LAD_E 1               // Zustand des Eingang 1 in das Bitergebnis
                                // übertragen. Weil der Eingang nicht geschaltet
                                // ist, wird das Bitergebnis ausgeschaltet
          EIN_A 1               // Dieser Befehl wird nicht ausgeführt, weil
                                // das Bitergebnis ausgeschaltet ist
          AUS_A 2               // Auch dieser Befehl wird nicht ausgeführt,
                                // weil das Bitergbnis immer noch aus ist
          LAD_M M_EIN           // Alle Lade- und Vergleichsbefehle werden
                                // unabhängig vom Bitergebnis ausgeführt. Hier
                                // übertragen wir den Zustand des Merkers
                                // "immer ein" in das Bitergebnis. Damit ist
                                // sichergestellt, daß das Bitergebnis nun
                                // eingeschaltet ist.
          AUS_A 1               // Dieser Befehl wird jetzt ausgeführt, weil das
                                // Bitergebnis eingeschaltet ist
          SPRINGN Irgendwo      // Springe zum Label "Irgendwo", wenn das
                                // Bitergebnis ausgeschaltet ist
          SPRINGJ Sonstwo       // Springe zum Label "Sonstwo", wenn das
                                // Bitergebnis eingeschaltet ist
```

In der Befehlübersicht, die im nächsten Kapitel beginnt, wird bei jedem einzelnen Befehl darauf hingewiesen, welche Auswirkung der Befehl auf das Bitergebnis hat, und ob die Ausführung des Befehls vom Bitergebnis abhängig ist.

n Bitergebnisschieberegister

Mit Hilfe des Bitergebnisschieberegisters haben Sie direkten Zugriff auf die 16 letzten Bitergebnisse. Wann immer ein Befehl das Bitergebnis direkt verändert, wird der Inhalt des Bitergebnisschieberegisters um eine Stelle nach links geschoben und der neue Wert an die erste Stelle des Bitergebnisschieberegisters gespeichert. Sie können diese Funktion verwenden, um z.B. verschiedene Abfragen zu verschachteln, also etwas ähnliches wie Klammerebenen zu realisieren. Weitere Informationen zum Bitergebnisschieberegister finden Sie im Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** - **Fehler! Verweisquelle konnte nicht gefunden werden.** (Seite **Fehler! Textmarke nicht definiert.**)

Kapitel 3 Befehlsübersicht

In diesem Kapitel finden Sie eine vollständige Übersicht aller MC-1A Befehle sowie aller innerhalb der Entwicklungsoberfläche vordefinierten Makros. Zur besseren Übersicht und zum einfacheren Auffinden der jeweiligen Befehle besteht die Referenz aus zwei Teilen:

Abschnitt 3.1 - Befehle nach Funktionsgruppen (ab Seite 3)

In diesem Abschnitt finden Sie eine Übersicht aller MC-1A Befehle sortiert nach Funktionsgruppe. Zu jedem Befehl ist die entsprechende Syntax sowie eine Kurzbeschreibung mit aufgeführt.

Abschnitt 3.2 - Alphabetische Befehlsübersicht (Seite 3)

Hier finden Sie eine vollständige, alphabetische Referenz aller MC-1A SPS-Befehle. Jede Befehlsbeschreibung wird ergänzt durch ein Programmierbeispiel: so können Sie auf einen Blick erkennen, wie der jeweilige Befehl eingesetzt wird.

n Kennzeichnung der Parameter

Auch die Parameter zu jedem Befehl sind mit kleinen Symbolen markiert, wie z.B.

LAD_MI Erster Merker V

Diese Information definiert den Datentyp (vgl. Kapitel 2.2 - Datentypen ab Seite 3) , der von diesem Befehl erwartet wird. Folgende Datentypen sind hierbei zulässig:

- **n** V - Variable
- **n** M - Merker
- **n** E - Eingang
- **n** A - Ausgang
- **n** K - Konstante oder Label

n Zusammenspiel mit dem Bitergebnisspeicher

Am Anfang jeder Befehlsbeschreibung finden Sie einen kleinen Kasten:

Gruppe	Abhängig von BES	Verändert BES
Definitionen	**y Nein**	**n Nein**

Die Informationen haben folgende Bedeutung:

- **n** "Gruppe" ordnet den Befehl einer Funktionsgruppe zu. Eine Übersicht aller Befehle sortiert nach Funktionsgruppe finden Sie in Kapitel 3.1 ab Seite 3.
- **n** "Abhängig von BES" verrät Ihnen auf den ersten Blick, ob dieser Befehl nur dann ausgeführt wird, wenn der Bitergebnisspeicher einen bestimmten Wert hat. Steht hier "Nein", wird der Befehl immer ausgeführt. "Ja" hingegen bedeutet, daß der Befehl nur dann ausgeführt wird, wenn der BES den Zustand "ein" hat. In einigen speziellen Fällen kann hier auch "Wenn aus" stehen. Dies bedeutet, daß der Befehl nur dann ausgeführt wird, wenn der BES ausgeschaltet ist.
- **n** "Verändert BES" besagt, ob der Zustand des BES durch diesen Befehl verändert wird.

n Wichtiger Hinweis

Die Befehlsübersicht setzt voraus, daß Sie mit den Grundlagen der MC-1A Programmierung bereits vertraut sind. Beachten Sie ggf. die Erläuterungen in Kapitel 2 - Programmieren mit MC-1A ab Seite 3.

3.1 Befehle nach Funktionsgruppen

n Definitionsbefehle

Befehl	Bedeutung	Seite
DEF_A Ausgangsnr. K, Symbolname K	DEF_A weist einem Ausgang einen symbolischen Namen zu.	3
DEF_E Eingangsnr. K, Symbolname K	DEF_E weist einem Eingang einen symbolischen Namen zu.	3
DEF_M Merkernr. K, Symbolname K	DEF_M weist einem Merker einen symbolischen Namen zu.	3
DEF_V Variablennr. K, Symbolname K	DEF_V weist einer Variablen einen symbolischen Namen zu.	3
DEF_W Wert K, Symbolname K	DEF_W weist dem Wert einer Konstanten einen symbolischen Namen zu.	3

n Tabelle 1 – Definitionsbefehle

n Compilieranweisungen

Befehl	Bedeutung	Seite
#ELSE	#ELSE wird für Abfragen im Zusammenhang mit der bedingten Compilierung verwendet. Vor dem Befehl wird eine Bedingung mit dem Befehl EQ abgefragt. Ist die erste abhängige Anweisung nicht erfüllt, wird die zweite abhängige Anweisung durchgeführt.	3
#ENDIF	#ENDIF beendet eine bedingte Compilierung.	3
#IF Abfragebedingung K #IF Wert1 K EQ Wert2 K	#IF wird für Abfragen im Zusammenhang mit der bedingten Compilierung verwendet. Vor dem Befehl wird eine Bedingung mit dem Befehl EQ abgefragt. Wenn die erste abhängige Bedingung erfüllt ist, wird die erste Anweisung durchgeführt, ist die erste abhängige Anweisung nicht erfüllt, wird die zweite abhängige Anweisung (wenn eine vorhanden ist) durchgeführt.	3
#INCLUDE Quellcodedateiname K	#INCLUDE fügt eine weitere Quellcodedatei an der aktuellen Stelle ein.	3

n Tabelle 2 – Compileranweisungen

n Merkerbefehle

Befehl	Bedeutung	Seite
AUS_M Merker M	Der Befehl AUS_M löscht den angegebenen Merker.	3
AUS_MI Zeiger V	Der Befehl AUS_MI löscht den Merker, der durch den Zeiger bestimmt wird.	3
EIN_M Merker M	EIN_M setzt den angegebenen Merker.	3
EIN_MI Zeiger V	EIN_MI setzt den durch den angegebenen Zeiger bestimmten Merker.	3
LAD_M Merker M	LAD_M lädt den Zustand des angegebenen Merkers in den Bitergebnisspeicher.	3
LAD_MI Zeiger V	LAD_MI überträgt den Zustand des durch den angegebenen Zeiger bestimmten Merkers in den Bitergebnisspeicher.	3
MOD_M Merker M	MOD_M überträgt den Zustand des Bitergebnisspeichers in den angegebenen Merker.	3
MOD_MI Zeiger V	MOD_MI überträgt den Zustand des Bitergebnisspeichers in den durch den angegebenen Zeiger bestimmten Merker.	3
NLAD_M Merker M	NLAD_M lädt den invertierten Zustand des Merkers in den Bitergebnisspeicher.	3
NODER_M Merker M	NODER_M verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Merkers mit einem logischen ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
NUND_M Merker M	NUND_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Merkers mit einem logischen UND. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
ODER_M Merker M	ODER_M verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Merkers mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
UND_M Merker M	UND_M verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Merkers mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.	3
XODER_M Merker M	XODER_M verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Merkers mit einem logischen Exclusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.	3

n Tabelle 3 – Merkerbefehle

n Ein-/Ausgangsbefehle

Befehl	Bedeutung	Seite
AUS_A Ausgang A	Der Befehl AUS_A schaltet den angegebenen Ausgang aus.	3
AUS_AI Zeiger V	Der Befehl AUS_AI schaltet den Ausgang aus, der durch den Zeiger bestimmt wird.	3
EIN_A Ausgang A	EIN_A schaltet den angegebenen Ausgang ein.	3
EIN_AI Zeiger V	EIN_AI schaltet den durch den angegebenen Zeiger bestimmten Ausgang ein.	3
LAD_A Ausgang A	LAD_A lädt den Zustand des angegebenen Ausgangs in den Bitergebnisspeicher.	3
LAD_AI Zeiger V	LAD_AI lädt den Zustand des durch den Zeiger bestimmten Ausgangs in den Bitergebnisspeicher.	3
LAD_E Eingang E	LAD_E lädt den Zustand des angegebenen Eingangs in den Bitergebnisspeicher.	3
LAD_EI Zeiger V	LAD_EI lädt den Zustand des durch den Zeiger angegebenen Eingangs in den Bitergebnisspeicher.	3
MOD_A Ausgang A	MOD_A überträgt den Zustand des Bitergebnisspeichers auf den angegebenen Ausgang.	3
MOD_AI Zeiger V	MOD_AI überträgt den Zustand des Bitergebnisspeichers in den durch den angegebenen Zeiger bestimmten Ausgang.	3
NLAD_E Eingang E	NLAD_E lädt den invertierten Zustand eines Eingangs in den Bitergebnisspeicher.	3
NLAD_A Ausgang A	NLAD_A lädt den invertierten Zustand des Ausgangs in den Bitergebnisspeicher.	3
NUND_A Ausgang A	NUND_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Ausgangs mit einem logischen UND. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
NUND_E Eingang E	NUND_E verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Eingangs mit einem logischen UND. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
UND_E Eingang E	UND_E verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Eingangs mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.	3
UND_A Ausgang A	UND_A verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Ausgangs mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.	3
NODER_A Ausgang A	NODER_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Ausgangs mit einem logischen ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
NODER_E Eingang E	NODER_E verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Eingangs mit einem logischen ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3

Befehl	Bedeutung	Seite
ODER_A Ausgang A	ODER_A verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Ausgangs mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
ODER_E Eingang E	ODER_E verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Eingangs mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	3
XODER_A Ausgang A	XODER_A verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Ausgangs mit einem logischen Exclusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.	3
XODER_E Eingang E	XODER_E verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Eingangs mit einem logischen Exclusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.	3

n Tabelle 4 – Ein-/ Ausgangsbefehle

n Variablenbefehle

Befehl	Bedeutung	Seite
ADD_IV Zeiger V, Variable V	Der Befehl ADD_IV addiert den Inhalt der Variable, die durch den Zeiger bestimmt wird, mit dem Inhalt der zweiten angegebenen Variable. Das Ergebnis der Addition wird in VARERG gespeichert	3
ADD_VA Variable V, Wert K	Der Befehl ADD_VA addiert zum Inhalt der Variable den angegebenen konstanten Wert. Das Ergebnis der Addition wird in der Ergebnisvariable VARERG gespeichert.	3
ADD_VI Variable V, Zeiger V	Der Befehl ADD_VI addiert den Inhalt der angegebenen Variable mit dem Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis der Addition wird in der Ergebnisvariable VARERG gespeichert.	3
ADD_VV Variable1 V, Variable2 V	Der Befehl ADD_VV addiert die Inhalte der beiden angegebenen Variablen. Das Ergebnis der Addition wird in der Ergebnisvariable VARERG gespeichert.	3
DEC_V Variable V, Wert K	Der Befehl DEC_V verringert den Inhalt der Variable um den angegeben Wert Wert. Der Ergebnisspeicher VARERG wird nicht beeinflußt.	3
DIV_IV Zeiger V, Variable V	DIV_IV dividiert den Inhalt der durch den angegebenen Zeiger bestimmten Variable durch den Inhalt der zweiten Variable. Das Ergebnis der Division wird in VARERG und DIVREST gespeichert.	3
DIV_VA Variable V, Wert K	DIV_VA dividiert den Inhalt der Variable durch den angegebenen Wert. Das ganzzahlige Ergebnis der Division wird in der Ergebnisvariable VARERG, der Divisionsrest wird in DIVREST gespeichert.	3
DIV_VI Variable V, Zeiger V	DIV_VI dividiert den Inhalt der angegebenen Variable durch den Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis der Division wird in VARERG und DIVREST gespeichert.	3
DIV_VV Variable1 V, Variable2 V	DIV_VV dividiert den Inhalt der Variable1 durch den Inhalt der Variable2. Das ganzzahlige Ergebnis der Division wird in der Ergebnisvariable VARERG der Divisionsrest wird in DIVREST	3

Befehl	Bedeutung	Seite
	gespeichert.	
LAD_IV Zeiger V, Variable V	LAD_IV überträgt den Inhalt der angegebenen Variablen in die durch den Zeiger bestimmte Variable.	3
LAD_VA Variable V, Wert K	LAD_VA lädt den angegebenen Wert in die Variable.	3
LAD_VI Variable V, Zeiger V	LAD_VI überträgt den Inhalt der durch den Zeiger bestimmten Variable in die angegebene Variable.	3
LAD_VV Variable1 V, Variable2 V	LAD_VV lädt den Inhalt der Variable2 in die Variable1.	3
MUL_IV Zeiger V, Variable V	MUL_IV multipliziert den Inhalt der durch den Zeiger bestimmten Variable mit dem Inhalt der angegebenen Variable. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.	3
MUL_VA Variable V, Wert K	MUL_VA multipliziert den Inhalt der Variable mit dem angegebenen Wert. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.	3
MUL_VI Variable V, Zeiger V	MUL_VI multipliziert den Inhalt der durch den Zeiger bestimmten Variable mit dem Inhalt der angegebenen Variable. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.	3
MUL_VV Variable1 V, Variable2 V	MUL_VV multipliziert die Inhalte von Variable1 und Variable2. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.	3
INC_V Variable V, Wert K	INC_V erhöht den Inhalt der Variable den angegebenen Wert. Der Ergebnisspeicher VARERG wird nicht beeinflußt.	3
ODER_IV Zeiger V, Variable V	ODER_IV verknüpft die Inhalte der durch den Zeiger bestimmten und der angegebenen Variable mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.	3
ODER_VI Variable V, Zeiger V	ODER_VI verknüpft die Inhalte der durch den Zeiger bestimmten und der angegebenen Variable mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.	3
ODER_VA Variable V, Wert K	ODER_VA verknüpft den Inhalt der Variablen und den angegebenen Wert mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.	3
ODER_VV Variable1 V, Variable2 V	ODER_VV der beiden angegebenen Variablen mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.	3
SRL_VA Variable V, Wert K	SRL_VA schiebt den Inhalt der Variable um den angegeben Wert nach rechts. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.	3
SRL_VV Variable1 V, Variable2 V	SRL_VV schiebt den Inhalt der Variable1 um den Inhalt der Variable2 nach rechts. Das Ergebnis wird in der Ergebnisvariable	3

Befehl	Bedeutung	Seite
	VARERG gespeichert.	
SLL_VA Variable V, Wert K	SLL_VA schiebt den Inhalt der Variable, die als erster Parameter angegeben wird, um den angegebenen Wert nach links. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.	3
SLL_VV Variable1 V, Variable2 V	SLL_VV schiebt den Inhalt der Variable1 um den Inhalt der Variable2 nach links. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.	3
SUB_VA Variable V, Wert K	SUB_VA subtrahiert vom Inhalt der Variable den angegebenen Wert. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	3
SUB_VV Variable1 V, Variable2 V	SUB_VV subtrahiert vom Inhalt der Variable1 den Inhalt der Variable2. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	3
SUB_VI Variable V, Zeiger V	SUB_VI subtrahiert vom Inhalt der angebenen Variable den Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	3
SUB_IV Zeiger V, Variable V	SUB_IV subtrahiert vom Inhalt der durch den Zeiger bestimmten Variable den Inhalt der angegebenen Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	3
UND_IV Zeiger V, Variable V	UND_IV verknüpft den Inhalt der durch den Zeiger bestimmten Variable und den Inhalt der angegebenen Variable mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.	3
UND_VA Variable V, Wert K	UND_VA verknüpft den Inhalt der Variable und den angegebenen Wert mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.	3
UND_VI Variable V, Zeiger V	UND_VI verknüpft die Inhalte der angegebenen Variable und der durch den Zeiger bestimmten Variable mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.	3
UND_VA Variable1 V, Variable2 V	UND_VV verknüpft die Inhalte der beiden angegebenen Variablen mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.	3

n Tabelle 5 – Variablenbefehle

n Variablenvergleichsbefehle

Befehl	Bedeutung	Seite
VERG_IV Zeiger V, Variable V	VERG_IV vergleicht den Inhalt der durch den Zeiger bestimmten Variable mit dem Inhalt der angegebenen Variable. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	3
VERG_VA Variable V, Wert K	VERG_VA vergleicht den Inhalt der angegebenen Variable mit dem angegebenen Wert. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	3
VERG_VI Variable V, Zeiger V	VERG_VI vergleicht den Inhalt der angegebenen Variable mit dem Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	3
VERG_VV Variable1 V, Variable2 V	VERG_VV vergleicht die Inhalte der beiden angegebenen Variablen. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	3

n Tabelle 6 – Variablenvergleichbefehle

n Datenkonvertierungsbefehle

Befehl	Bedeutung	Seite
LAD_MV Erster Merker M, Variable V	LAD_MV überträgt den Inhalt der Variablen auf 32 Merker. Die Übertragung erfolgt bitweise, d.h. das erste Bit der Variable wird auf den ersten Merker übertragen, das zweite Bit der Variable auf die zweite Variable usw.	3
LAD_VM Variable V, Erster Merker M	LAD_VM überträgt den Zustand von 32 Merkern ab dem angegebenen ersten Merker in eine Variable. Die Übertragung erfolgt bitweise, d.h. der erste angegebene Merker wird in das erste Bit der Variable übertragen, der zweite Merker in das zweite Bit usw.	3

n Tabelle 7 – Datenkonvertierungsbefehle

n Makros

Befehl	Bedeutung	Seite
ENDM	ENDM schließt eine Makrodefinition ab. Geht dem Befehl ENDM keine MACRO Anweisung voraus, so führt dies zu einem Compilierungsfehler.	3
EXITM	EXITM beendet die Makrodefinition in Abhängigkeit einer bedingten Compilierung. Verwenden Sie EXITM, um ein Makro in Abhängigkeit von symbolischen Konstanten abzubrechen.	3
Name K MACRO [Par 1 K, [Par 2 K, [...]]]	MACRO bestimmt den Anfang einer Makro-Definition. Innerhalb des Makro-Definitionsblocks geben Sie dann den Namen des Makros, etwaige Parameter und den zugeordneten SPS-Programmcode an. Die eckigen Klammern in der Syntax deuten an, daß Sie hier selbst entscheiden können, wieviele Parameter Sie an einen Makrobefehl übergeben möchten.	3

n Tabelle 8 – Makrobefehle

n Programmablaufbefehle

Befehl	Bedeutung	Seite
GEHUPRI Label K	GEHUPRI ruft ein Unterprogramm ab dem angegebenen Label auf.	3
GEHUPRJ Label K	GEHUPRJ ruft ein Unterprogramm ab dem angegebenen Label auf. Der Aufruf wird nur ausgeführt, wenn der Bitergebnisspeicher zum Zeitpunkt der Befehlsausführung eingeschaltet ist.	3
GEHUPRN Label K	GEHUPRJ ruft ein Unterprogramm ab dem angegebenen Label auf. Der Aufruf wird nur ausgeführt, wenn der Bitergebnisspeicher zum Zeitpunkt der Befehlsausführung ausgeschaltet ist.	3
GEHUPRV Zieladresse V	GEHUPRV ruft ein Unterprogramm auf, dessen Startadresse in der angegebenen Variable enthalten ist. Verwenden Sie diesen Befehl um eine dynamische Zyklusprogrammierung zu erstellen.	3
LAD_P1 Label K	LAD_P1 lädt den Programmzähler des ersten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	3
LAD_P2 Label K	LAD_P2 lädt den Programmzähler des zweiten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	3
LAD_P3 Label K	LAD_P3 lädt den Programmzähler des dritten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	3
LAD_P4 Label K	LAD_P4 lädt den Programmzähler des vierten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	3
LAD_VL Variable V, Label K	LAD_VL überträgt die Programmadresse des Labels in die angegebene Variable. Der Befehl LAD_VL ist notwendig, um ein Unterprogramm mit GEHUPRV anzuspringen.	3
SPRING Label K	SPRING setzt den Programmablauf ab dem angegebenen Label fort.	3
SPRINGJ Label K	SPRINGJ setzt den Programmablauf ab dem angegebenen Label fort, wenn der Bitergebnisspeicher zu diesem Zeitpunkt eingeschaltet ist.	3
SPRINGN Label K	SPRINGN setzt den Programmablauf ab der angegebenen Label fort, wenn der Bitergebnisspeicher zu diesZeitpunkt ausgeschaltet ist.	3
UPREND	UPREND beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.	3
UPRENDJ	UPRENDJ beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.	3
UPRENDN	UPRENDN beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.	3
WART_A...WART_E	WART_A kann zur Programmierung einfacher Schleifen verwendet werden. Die Schleife wird so lange durchlaufen bis der Bitergebnisspeicher eingeschaltet ist.	3

n Tabelle 9 – Programmablaufbefehle

n Raum für Ihre Notizen

3.2 Alphabetische Befehlsübersicht

n ADD_IV

ADD_IV Zeiger V, Variable V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

Der Befehl ADD_IV addiert den Inhalt der Variable, die durch den Zeiger bestimmt wird, mit dem Inhalt der zweiten angegebenen Variable. Das Ergebnis der Addition wird in VARERG gespeichert

Operation

```
(VARERG) ç (Zeiger à Variable) + (Variable)
```

Beispiel

```
ADD_IV     V_ZEIGER,V_TEST        // Der Inhalt der Variable, auf die V_ZEIGER
                                  // zeigt, wird zur Variable V_TEST addiert.
                                  // Das Ergebnis wird in VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n ADD_VA

ADD_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

Der Befehl ADD_VA addiert zum Inhalt der Variable den angegebenen konstanten Wert. Das Ergebnis der Addition wird in der Ergebnisvariable VARERG gespeichert.

Operation

```
(VARERG) ç (Variable) + Wert
```

Beispiel

```
DEF_W      10,ZAHL                // Definiert ZAHL mit dem Wert 100
ADD_VA     V_TEST,ZAHL            // addiert die Variable V_TEST zum Wert ZAHL.
                                  // Das Ergebnis wird in VARERG speichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n ADD_VI

ADD_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

Der Befehl ADD_VI addiert den Inhalt der angegebenen Variable mit dem Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis der Addition wird in der Ergebnisvariable VARERG gespeichert.

Operation

`(VARERG) ç (Variable) + (Zeiger à Variable)`

Beispiel

```
ADD_VI     V_TEST,V_ZEIGER        // Der Inhalt der Variable, auf die V_ZEIGER
                                  // zeigt wird zur Variable V_TEST addiert.
                                  // Das Ergebnis wird in VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n ADD_VV

ADD_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

Der Befehl ADD_VV addiert die Inhalte der beiden angegebenen Variablen. Das Ergebnis der Addition wird in der Ergebnisvariable VARERG gespeichert.

Operation

`(VARERG) ç (Variable1) + (Variable2)`

Beispiel

```
LAD_VA     V_ZAHL1,10             // In die Variable V_ZAHL1 wird
                                  // der Wert 10 geladen
LAD_VA     V_ZAHL2,20             // in die Variable V_ZAHL2 wird
                                  // der Wert 20 geladen
ADD_VV     V_ZAHL1,V_ZAHL2        // die Variablen V_ZAHL1 und V_ZAHL2
                                  // werden
                                  // addiert und das Ergebnis wird in
                                  // VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n AUS_A

AUS_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**- Ja**	**n Nein**

Der Befehl AUS_A schaltet den angegebenen Ausgang aus.

Operation

`(Ausgang)` **ç** `AUS`

Beispiel

```
AUS_A      A_TEST                  // Schaltet den Ausgang A_TEST aus
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n AUS_AI

AUS_AI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**- Ja**	**n Nein**

Der Befehl AUS_AI schaltet den Ausgang aus, der durch den Zeiger bestimmt wird.

Operation

`(Zeiger` **à** `Ausgang)` **ç** `AUS`

Beispiel

```
LAD_VA     V_TEST,320              // Lädt die Variable V_TEST mit dem Wert 320
AUS_AI     V_TEST                  // schaltet den Ausgang 320 aus
INC_VA     V_TEST,1                // erhöht den Variablenwert um 1
AUS_AI     V_TEST                  // schaltet den Ausgang 321 aus
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n AUS_M

AUS_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**- Ja**	**n Nein**

Der Befehl AUS_M löscht den angegebenen Merker.

Operation

`(Merker) ç AUS`

Beispiel

```
AUS_M     M_TEST                // Der Merker M_TEST wird gelöscht
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Merkerbefehle (Seite 3)

n AUS_MI

AUS_MI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**- Ja**	**n Nein**

Der Befehl AUS_MI löscht den Merker, der durch den Zeiger bestimmt wird.

Operation

`(Zeiger à Merker) ç AUS`

Beispiel

```
LAD_VA    V_TEST,320            // Lädt die Variable V_TEST mit dem Wert 320
AUS_MI    V_TEST                // löscht den Merker 320
```

Hinweise

Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Merkerbefehle (Seite 3)

n DEC_V

DEC_V Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

Der Befehl DEC_V verringert den Inhalt der Variable um den angegeben Wert Wert. Der Ergebnisspeicher VARERG wird nicht beeinflußt.

Nach dem dekrementieren der Variable1 wird automatisch vom System ein Vergleich gegen "0" durchgeführt und die Variablen-Vergleichsmerker dementsprechend gesetzt:

- M_GLEICH wird gesetzt, wenn nach der Operation die Variable1 den Wert "0" enthält
- M_GROESSER wird gesetzt, wenn nach der Operation die Variable1 größer als "0" ist
- M_KLEINER wird gesetzt, wenn nach der Operation die Variable1 kleiner als "0" ist.

Operation

```
(Variable1)   ç (Variable) - Wert
(M_GLEICH)    ç (Variable = 0)
(M_GROESSER)  ç (Variable > 0)
(M_KLEINER)   ç (Variable < 0)
```

Beispiel

```
// In diesem Beispiel haben wir ein Variablenfeld, welches z.B. vom PC geschrieben
// wird und setzen die unteren 16 Bit. Wir verwenden den Befehl DEC_V, um die
// Zeigervariable bei jedem Schleifendurchlauf auf den nächsten Eintrag in der
// Tabelle zu initialisieren.

DEF_W      4000,TABELLE_ENDE       // Anfang des Variablenfeldes
LAD_VA     V_ZEIGER,TABELLE_ENDE   // Zeigervariable initialisieren
LAD_VA     V_MASKE,65535           // entspricht FFFFh

LOOP:
ODER_VI    V_MASKE,V_ZEIGER        // Verknüpfung durchführen
LAD_IV     V_ZEIGER,VARERG         // Ergebnis der Verknüpfung wieder in Tabelle
DEC_V      V_ZEIGER,1              // Tabellenzeiger auf vorherigen Eintrag
SPRING     LOOP                    // Zurück zur Schleife
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n DEF_A

DEF_A Ausgangsnr. K, Symbolname K

Gruppe	Abhängig von BES	Verändert BES
Definitionen	**y Nein**	**n Nein**

DEF_A weist einem Ausgang einen symbolischen Namen zu.

Beispiel

```
DEF_A      10,A_TEST              // Weist dem Ausgang 10 den Namen A_TEST zu
```

Bei der Programmierung kann anschließend der symbolische Name des Ausgangs verwendet werden.

Hinweise

- **n** Dieser Befehl belegt keinen zusätzlichen Speicher innerhalb der Steuerung. Die Größe des SPS-Programms ist unabhängig davon, ob Sie symbolisch oder unter Verwendung der Eingangsnummern arbeiten.
- **n** Sie können Definitionen sowohl in einer globalen Definitionsdatei als auch innerhalb eines MC-1A Quelltextes vornehmen. Bitte beachten Sie aber, daß Definitionen innerhalb des Quelltexts nur für die entsprechende Quelltextdatei, nicht aber für das gesamte Projekt gelten.

Siehe auch

Definitionsbefehle (Seite 3)

n DEF_E

DEF_E Eingangsnr. K, Symbolname K

Gruppe	Abhängig von BES	Verändert BES
Definitionen	**y Nein**	**n Nein**

DEF_E weist einem Eingang einen symbolischen Namen zu.

Beispiel

```
DEF_E      10,E_TEST              // Weist dem Eingang 10 den Namen E_TEST zu
```

Bei der Programmierung kann anschließend der symbolische Name des Eingangs verwendet werden.

Hinweise

- **n** Dieser Befehl belegt keinen zusätzlichen Speicher innerhalb der Steuerung. Die Größe des SPS-Programms ist unabhängig davon, ob Sie symbolisch oder unter Verwendung der Eingangsnummern arbeiten.
- **n** Sie können Definitionen sowohl in einer globalen Definitionsdatei als auch innerhalb eines MC-1A Quelltextes vornehmen. Bitte beachten Sie aber, daß Definitionen innerhalb des Quelltexts nur für die entsprechende Quelltextdatei, nicht aber für das gesamte Projekt gelten.

Siehe auch

Definitionsbefehle (Seite 3)

n DEF_M

DEF_M Merkernr. K, Symbolname K

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**n Nein**

DEF_M weist einem Merker einen symbolischen Namen zu.

Beispiel

```
DEF_M      10,M_TEST              // Weist dem Merker 10 den Namen M_TEST zu
```

Bei der Programmierung kann anschließend der symbolische Name des Merkers verwendet werden.

Hinweise

- **n** Dieser Befehl belegt keinen zusätzlichen Speicher innerhalb der Steuerung. Die Größe des SPS-Programms ist unabhängig davon, ob Sie symbolisch oder unter Verwendung der Eingangsnummern arbeiten.
- **n** Sie können Definitionen sowohl in einer globalen Definitionsdatei als auch innerhalb eines MC-1A Quelltextes vornehmen. Bitte beachten Sie aber, daß Definitionen innerhalb des Quelltexts nur für die entsprechende Quelltextdatei, nicht aber für das gesamte Projekt gelten.

Siehe auch

Definitionsbefehle (Seite 3)

n DEF_V

DEF_V Variablennr. K, Symbolname K

Gruppe	Abhängig von BES	Verändert BES
Definitionen	**y Nein**	**n Nein**

DEF_V weist einer Variablen einen symbolischen Namen zu.

Beispiel

```
DEF_V      10,V_TEST              // Weist der Variablen 10 den Namen V_TEST zu
```

Bei der Programmierung kann anschließend der symbolische Name der Variablen verwendet werden.

Hinweise

- **n** Dieser Befehl belegt keinen zusätzlichen Speicher innerhalb der Steuerung. Die Größe des SPS-Programms ist unabhängig davon, ob Sie symbolisch oder unter Verwendung der Eingangsnummern arbeiten.
- **n** Sie können Definitionen sowohl in einer globalen Definitionsdatei als auch innerhalb eines MC-1A Quelltextes vornehmen. Bitte beachten Sie aber, daß Definitionen innerhalb des Quelltexts nur für die entsprechende Quelltextdatei, nicht aber für das gesamte Projekt gelten.

Siehe auch

Definitionsbefehle (Seite 3)

n DEF_W

DEF_W Wert K, Symbolname K

Gruppe	Abhängig von BES	Verändert BES
Definitionen	**y Nein**	**n Nein**

DEF_W weist dem Wert einer Konstanten einen symbolischen Namen zu.

Beispiel

```
DEF_W     10,TEST               // Weist der Konstanten 10 den Namen TEST zu
```

Bei der Programmierung kann anschließend der symbolische Name des konstanten Werts verwendet werden.

Hinweise

- **n** Dieser Befehl belegt keinen zusätzlichen Speicher innerhalb der Steuerung. Die Größe des SPS-Programms ist unabhängig davon, ob Sie symbolisch oder unter Verwendung der Eingangsnummern arbeiten.
- **n** Sie können Definitionen sowohl in einer globalen Definitionsdatei als auch innerhalb eines MC-1A Quelltextes vornehmen. Bitte beachten Sie aber, daß Definitionen innerhalb des Quelltexts nur für die entsprechende Quelltextdatei, nicht aber für das gesamte Projekt gelten.

Siehe auch

Definitionsbefehle (Seite 3)

n DIV_IV

DIV_IV Zeiger V, Variable V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

DIV_IV dividiert den Inhalt der durch den angegebenen Zeiger bestimmten Variable durch den Inhalt der zweiten Variable. Das Ergebnis der Division wird in VARERG und DIVREST gespeichert.

Operation

```
(VARERG)  ç (Zeiger à Variable) / (Variable)
(DIVREST) ç (Zeiger à Variable) % (Variable)
```

Beispiel

```
DIV_IV    V_ZEIGER,V_TEST       // Der Inhalt der Variable, auf die V_ZEIGER
                                // zeigt, wird durch den Inhalt der Variablen
                                // V_TEST dividiert. Das Ergebnis wird in VARERG
                                // und DIVREST gespeichert
```

Hinweise

- **n** Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n DIV_VA

DIV_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

DIV_VA dividiert den Inhalt der Variable durch den angegebenen Wert. Das ganzzahlige Ergebnis der Division wird in der Ergebnisvariable VARERG, der Divisionsrest wird in DIVREST gespeichert.

Operation

```
(VARERG)  ç (Variable) / Wert
(DIVREST) ç (Variable) % Wert
```

Beispiel

```
DEF_W     10,ZAHL               // Lädt ZAHL mit dem Wert 10
DIV_VA    V_TEST,ZAHL           // dividiert die Variable durch den Wert ZAHL
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n DIV_VI

DIV_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

DIV_VI dividiert den Inhalt der angegebenen Variable durch den Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis der Division wird in VARERG und DIVREST gespeichert.

Operation

```
(VARERG)  ç (Variable) / (Zeiger à Variable)
(DIVREST) ç (Variable) % (Zeiger à Variable)
```

Beispiel

```
DIV_VI    V_TEST,V_ZEIGER       // Der Inhalt der Variable V_TEST, wird durch den
                                // Inhalt der Variable, auf die V_ZEIGER zeigt,
                                // dividiert. Das Ergebnis wird in VARERG und
                                // DIVREST gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n DIV_VV

DIV_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

DIV_VV dividiert den Inhalt der Variable1 durch den Inhalt der Variable2. Das ganzzahlige Ergebnis der Division wird in der Ergebnisvariable VARERG der Divisionsrest wird in DIVREST gespeichert.

Operation

```
(VARERG)  ç (Variable1) / (Variable2)
(DIVREST) ç (Variable1) % (Variable2)
```

Beispiel

```
LAD_VA     V_ZAHL1,24             // In V_ZAHL1 wird der Wert 24 geladen
LAD_VA     V_ZAHL2,10             // In V_ZAHL2 wird der Wert 10 geladen
DIV_VV     V_ZAHL1,V_ZAHL2        // Dividiert V_ZAHL1 durch V_ZAHL2
                                  // Das Ergebnis (2) wird in VARERG, der Rest (4)
                                  // wird in DIVREST gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n EIN_A

EIN_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**- Ja**	**n Nein**

EIN_A schaltet den angegebenen Ausgang ein.

Operation

```
(Ausgang) ç EIN
```

Beispiel

```
EIN_A      A_TEST                 // Schaltet den Ausgang A_TEST ein
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n EIN_AI

EIN_AI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**- Ja**	**n Nein**

EIN_AI schaltet den durch den angegebenen Zeiger bestimmten Ausgang ein.

Operation

`(Zeiger à Ausgang) ç EIN`

Beispiel

```
LAD_VA     V_TEST,320            // Lädt die Variable V_TEST mit dem Wert 320
EIN_AI     V_TEST                // schaltet den Ausgang 320 ein
INC_V      V_TEST,1              // erhöht den Variablenwert um 1
EIN_AI     V_TEST                // schaltet den Ausgang 321 ein
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- Verwenden Sie EIN_AI, um einen Ausgang zu setzen, dessen Nummer bei der Programmerstellung noch nicht feststeht.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n EIN_M

EIN_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**- Ja**	**n Nein**

EIN_M setzt den angegebenen Merker.

Operation

`(Merker) ç EIN`

Beispiel

```
EIN_M      M_TEST                // Der Merker M_TEST wird gesetzt
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Merkerbefehle (Seite 3)

n EIN_MI

EIN_MI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**- Ja**	**n Nein**

EIN_MI setzt den durch den angegebenen Zeiger bestimmten Merker.

Operation

`(Zeiger à Merker) ç EIN`

Beispiel

```
LAD_VA     V_TEST,320            // Lädt die Variable V_TEST mit dem Wert 320
EIN_MI     V_TEST                // setzt den Merker 320
INC_V      V_TEST,1              // erhöht den Variablenwert um 1
EIN_MI     V_TEST                // setzt den Merker 321
```

Hinweise

- **n** Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- **n** Verwenden Sie EIN_MI, um einen Merker zu setzen, dessen Nummer bei der Programmerstellung noch nicht feststeht.

Siehe auch

Merkerbefehle (Seite 3)

n ENDM

ENDM

Gruppe	Abhängig von BES	Verändert BES
Makros	**y Nein**	**n Nein**

ENDM schließt eine Makrodefinition ab. Geht dem Befehl ENDM keine MACRO Anweisung voraus, so führt dies zu einem Compilierungsfehler.

Beispiel

```
// BES_EIN schaltet den Bitergebnisspeicher ein. Es werden kene Parameter benötigt.

BES_EIN    MACRO                 // Makro BES_EIN definieren
LAD_M      M_EIN                 // Quelltext des Makro
ENDM                             // Makro Ende
```

Siehe auch

Makros (Seite 3)

n EXITM

EXITM

Gruppe	Abhängig von BES	Verändert BES
Makros	**y Nein**	**n Nein**

EXITM beendet die Makrodefinition in Abhängigkeit einer bedingten Compilierung. Verwenden Sie EXITM, um ein Makro in Abhängigkeit von symbolischen Konstanten abzubrechen.

Beispiel

```
// Folgendes Makro soll Ein-/Ausgänge einer Schnittstelle testen. Je nach
// Ausbau wird das Macro schon vorher verlassen.

DEF_W     2,Interface            // Hier gehen wir davon aus: 2 Eingänge prüfen

INTERF    MACRO                  // Makro INTERF definieren
EIN_A     10                     // Ausgang 10 einschalten
LAD_E     11                     // Eingang 11 laden
UND_E     12                     // mit Eingang 12 verknüpfen
#IF Interface EQ 2               // wenn 2 Eingänge getestet werden sollen
MOD_M     INTER_OK               // dann Status in Merker übertragen
EXITM                            // und Makro beenden
#ENDIF
UND_E     13                     // ansonsten noch mit Eingang 13 verknüpfen
MOD_M     INTER_OK               // und in Merker übertragen
ENDM                             // Ende der Makrodefinition
```

Siehe auch

Makros (Seite 3)

n GEHUPRI

GEHUPRI Label k

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**y Nein**	**Siehe Text**

GEHUPRI ruft ein Unterprogramm ab dem angegebenen Label auf.

Operation

```
(Stackpointer)   ç  Programmadresse
Stackpointer     ç  Stackpointer - 2
Programmadresse  ç  Label-Adresse
(Bitergebnis)    ç  EIN
```

Beispiel

```
GEHUPRI    INIT                  // Rufe Unterprogramm INIT auf. Nach der Rückkehr
                                 // aus dem Unterprogramm ist der
                                 // Bitergebnisspeicher immer eingeschaltet
```

Hinweise

- **n** Dieser Befehl ist nicht abhängig vom Status des Bitergebnisspeichers und wird immer ausgeführt.
- **n** Nach Ausführung des Befehls ist der Bitergebnisspeicher immer eingeschaltet.
- **n** Es können maximal 8 Unterprogramme ineinander verschachtelt werden.

Siehe auch

Programmablaufbefehle (Seite 3)

n GEHUPRJ

GEHUPRJ Label K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**Siehe Text**

GEHUPRJ ruft ein Unterprogramm ab dem angegebenen Label auf. Der Aufruf wird nur ausgeführt, wenn der Bitergebnisspeicher zum Zeitpunkt der Befehlsausführung eingeschaltet ist.

Operation

```
(Stackpointer)   ç Programmadresse
Stackpointer     ç Stackpointer - 2
Programmadresse  ç Label-Adresse
(Bitergebnis)    ç EIN
```

Beispiel

```
GEHUPRJ   INIT                  // Wenn Bitergebnisspeicher eingeschaltet,
                                // dann rufe Unterprogramm INIT auf. Nach der
                                // Rückkehr ist der Bitergebnisspeicher immer EIN
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- Nach der Ausführung des Befehls ist der Bitergebnisspeicher immer eingeschaltet.
- Es können max. 8 Unterprogramme ineinander verschachtelt werden.

Siehe auch

Programmablaufbefehle (Seite 3)

n GEHUPRN

GEHUPRN Label K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**Siehe Text**

GEHUPRJ ruft ein Unterprogramm ab dem angegebenen Label auf. Der Aufruf wird nur ausgeführt, wenn der Bitergebnisspeicher zum Zeitpunkt der Befehlsausführung ausgeschaltet ist.

Operation

```
(Stackpointer)   ç Programmadresse
Stackpointer     ç Stackpointer - 2
Programmadresse  ç Label-Adresse
(Bitergebnis)    ç EIN
```

Beispiel

```
GEHUPRN   INIT                  // Wenn Bitergebnisspeicher ausgeschaltet,
                                // dann rufe Unterprogramm INIT auf. Nach der
                                // Rückkehr ist der Bitergebnisspeicher immer EIN
```

Hinweise

- Dieser Befehl wird nur ausgeführt, wenn der Bitergebnisspeicher ausgeschaltet ist.
- Nach der Ausführung des Befehls ist der Bitergebnisspeicher immer eingeschaltet.
- Es können maximal 8 Unterprogramme ineinander verschachtelt werden.

Siehe auch

Programmablaufbefehle (Seite 3)

n GEHUPRV

GEHUPRV Zieladresse V

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**Siehe Text**

GEHUPRV ruft ein Unterprogramm auf, dessen Startadresse in der angegebenen Variable enthalten ist. Verwenden Sie diesen Befehl um eine dynamische Zyklusprogrammierung zu erstellen.

Operation

```
(Stackpointer)   ç Programmadresse
Stackpointer     ç Stackpointer - 2
Programmadresse  ç (Zieladresse)
(Bitergebnis)    ç EIN
```

Beispiel

```
LAD_VL     V_ZYKLUS,PROG1          // Lädt Label PROG1 in V_ZYKLUS
                                   // (Initialisierungswert)

LOOP:                              // Hauptprogrammschleife
GEHUPRV    V_ZYKLUS                // Springe zur Startadresse, die als Parameter
                                   // in der Variable V_ZYKLUS enthalten ist
SPRING     LOOP                    // Weiter mit Hauptprogrammschleife

PROG1:                             // Unterprogramm 1
...                                // Programmcode
LAD_VL     V_ZYKLUS,PROG2          // Variable V_ZYKLUS wird mit der Startadresse
                                   // des nächsten Zyklus-Unterprogramms geladen,
                                   // in diesem Beispiel PROG2
UPREND                             // Ende des Zyklus-Unterprogramms

PROG2:                             // Unterprogramm 2
                                   // Programmcode
LAD_VL     V_ZYKLUS,PROG1          // Variable V_ZYKLUS wird mit der Startadresse
                                   // des nächsten Zyklus-Unterprogramms geladen,
                                   // in diesem Beispiel wieder PROG1
UPREND                             // Ende des Zyklus-Unterprogramms
```

Hinweise

- Dieser Befehl wird nur ausgeführt, wenn der Bitergebnisspeicher ausgeschaltet ist.
- Nach der Ausführung des Befehls ist der Bitergebnisspeicher immer eingeschaltet.
- Es können maximal 8 Unterprogramme ineinander verschachtelt werden.

Siehe auch

Programmablaufbefehle (Seite 3)

n INC_V

INC_V Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

INC_V erhöht den Inhalt der Variable den angegebenen Wert. Der Ergebnisspeicher VARERG wird nicht beeinflußt.

Operation

(Variable) ç (Variable) + Wert

Beispiel

```
// Hier bearbeiten wir ein Variablenfeld und setzen in jeder Variable die unteren
// 16 Bit. Mit INC_V setzen wir den Zeiger dann auf den nächsten Tabelleneintrag.
LAD_VA     V_ZEIGER,TABELLE_START  // Zeigervariable initialisieren
LAD_VA     V_MASKE,0xFFFF          // hexadezimal FFFF = untere 16 Bit
LOOP:
ODER_VI    V_MASKE,V_ZEIGER        // Verknüpfung durchführen
LAD_IV     V_ZEIGER,VARERG         // Ergebnis der Verknüpfung wieder in Tabelle
INC_V      V_ZEIGER,1              // Tabellenzeiger auf nächsten Eintrag
SPRING     LOOP                    // Zurück zur Schleife
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n LAD_A

LAD_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

LAD_A lädt den Zustand des angegebenen Ausgangs in den Bitergebnisspeicher.

Operation

(Bitergebnis) ç (Ausgang)

Beispiel

```
LAD_A      A_TEST                  // Lädt den physikalischen Zustand des Ausgangs
                                   // in den Bitergebnisspeicher
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n LAD_AI

LAD_AI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

LAD_AI lädt den Zustand des durch den Zeiger bestimmten Ausgangs in den Bitergebnisspeicher.

Operation

`(Bitergebnis) ç (Zeiger à Ausgang)`

Beispiel

```
LAD_AI     V_ZEIGER              // Lädt den Zustand des durch V_ZEIGER bestimmten
                                 // Ausgangs in den Bitergebnisspeicher
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n LAD_DT

LAD_DT Formatmaske V, Zeile K, Spalte K, Länge K

Gruppe	Abhängig von BES	Verändert BES
Display/Tastatur	**- Ja**	**n Nein**

LAD_DT legt das Format der folgenden Textanzeige durch die Argumente Zeile, Startposition innerhalb der Zeile und Länge des anzuzeigenden Textes fest. Die kodierte Formatbeschreibung wird in die angegebene Variable übertragen.

Gültige Werte für die Formatmaske

Name	Nummer	Bedeutung
V_ANZMSK1	91	Display-Anzeige auf dem ersten angeschlossenen Display
V_ANZMSK2	95	Display-Anzeige auf dem zweiten angeschlossenen Display
V_ANZMSK3	100	Text-Ausgabe über die interne serielle Schnittstelle
V_SERMSK	100	Text-Ausgabe über ein serielles Erweiterungsmodul

n Tabelle 10 – Gültige Werte für die Formatmaske bei LAD_DT

Beispiel

```
// Es soll ein Text in Zeile 7 ab Position 7 mit der Länge 10 ausgegeben werden

LAD_DT     V_ANZMSK1,ZEILE7,7,10   // Format für die Anzeige festlegen
LAD_VA     V_ANZNR1,1              // Textstring Nummer 1 auswählen
SETDSP     1,1                     // Text auf Display Nr. 1 ausgeben
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Fehler! Verweisquelle konnte nicht gefunden werden. (Seite **Fehler! Textmarke nicht definiert.**)
LAD_DV (Seite 3)

n LAD_DV

Gruppe	Abhängig von BES	Verändert BES
Display/Tastatur	**- Ja**	**n Nein**

LAD_DV Formatmaske V, Zeile K, Spalte K, Länge K, Dez. K, Vorzeichen K

LAD_DV legt das Format der folgenden Variableanzeige bzw. des folgenden Editorbefehls durch die Argumentezeile, Startposition innerhalb der Zeile und Länge des Feldes, die Anzahl der Nachkommastellen und die Freigabe des Vorzeichens fest. Die kodierte Formatbeschreibung wird in die Variable übertragen.

Gültige Werte für die Formatmaske

Name	Nummer	Bedeutung
V_ANZMSK1	91	Display-Anzeige auf dem ersten angeschlossenen Display
V_INPMSK1	93	Variablen-Eingabe auf dem ersten angeschlossenen Display
V_ANZMSK2	95	Display-Anzeige auf dem zweiten angeschlossenen Display
V_INPMSK2	97	Variablen-Eingabe auf dem zweiten angeschlossenen Display
V_ANZMSK3	100	Text-Ausgabe über die interne serielle Schnittstelle
V_SERMSK	100	Text-Ausgabe über ein serielles Erweiterungsmodul

n Tabelle 11 – Gültige Werte für die Formatmaske bei LAD_DV

Beispiel

```
// Im Beispiel soll die Variable 214 in Zeile 1, an der Pos 4
// mit 5 Stellen, davon 2 Nachkommastellen, und mit Vorzeichen
// ausgegeben werden

DEF_V      214,V_TEST              // Anzuzeigende Variable

LAD_DV     V_ANZMSK1,0,4,5,2,1     // Format für die Ausgabe festlegen
                                   // Erste Zeile = 0
LAD_VV     V_ANZNR1,V_TEST         // Wert für Anzeige laden
SETDSP     1,1                     // Variable anzeigen

// Im nächsten Beispiel soll ein Variablenwert am Display editiert
// werden.

DEF_V      214,V_TEST              // Variable, die editiert werden soll

LAD_DV     V_INPMSK1,0,4,5,2,1     // Format für Bearbeitung festlegen
                                   // Erste Zeile = 0
LAD_VV     V_INPDAT1,V_TEST        // Wert für Editor festlegen
SETEDI     1,1                     // Variable editieren
LAD_VV     V_TEST,V_INPDAT1        // Editierten Wert wieder in Variable speichern
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Fehler! Verweisquelle konnte nicht gefunden werden. (Seite **Fehler! Textmarke nicht definiert.**)
LAD_DT (Seite 3)

n LAD_E

LAD_E Eingang E

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

LAD_E lädt den Zustand des angegebenen Eingangs in den Bitergebnisspeicher.

Operation

```
(Bitergebnis) ç (Eingang)
```

Beispiel

```
LAD_E      E_TEST                // Lädt den Zustand des Eingangs
                                 // E_TEST in den Bitergebnisspeicher
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n LAD_EI

LAD_EI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

LAD_EI lädt den Zustand des durch den Zeiger angegebenen Eingangs in den Bitergebnisspeicher.

Operation

```
(Bitergebnis) ç (Zeiger à Eingang)
```

Beispiel

```
LAD_EI     V_ZEIGER              // Lädt den physikalischen Zustand des Eingangs
                                 // dessen Nummer in der Variablen enthalten ist
                                 // in den Bitergebnisspeicher
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n LAD_IV

LAD_IV Zeiger V, Variable V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

LAD_IV überträgt den Inhalt der angegebenen Variablen in die durch den Zeiger bestimmte Variable.

Operation

`(Zeiger à Variable) ç (Variable)`

Beispiel

```
// In diesem Beispiel werden Meßwerte in eine Tabelle geschrieben.

LAD_VA     V_ZEIGER,1800          // Tabelle beginnt bei Variable 1800
LAD_IV     V_ZEIGER,V_WERT        // lädt in die Variable V_WERT in den Inhalt der
                                  // Variable 1800 auf die V_ZEIGER zeigt.
INC_V      V_ZEIGER,1             // Erhöht V_ZEIGER auf nächsten Tablleneintrag
```

Hinweise

Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n LAD_M

LAD_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

LAD_M lädt den Zustand des angegebenen Merkers in den Bitergebnisspeicher.

Operation

`(Bitergebnis) ç (Merker)`

Beispiel

```
LAD_M      M_TEST                 // Lädt den Zustand von M_TEST in den
                                  // Bitergebnisspeicher
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)

n LAD_MI

LAD_MI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

LAD_MI überträgt den Zustand des durch den angegebenen Zeiger bestimmten Merkers in den Bitergebnisspeicher.

Operation

`(Bitergebnis) ç (Zeiger à Merker)`

Beispiel

```
LAD_MI     V_TEST                // Der Inhalt des Merkers, auf den V_TEST zeigt,
                                 // wird in den Bitergebnisspeicher übertragen
```

Hinweise

n Dieser Befehl ist nicht abhängig Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)

n LAD_MV

LAD_MV Erster Merker M, Variable V

Gruppe	Abhängig von BES	Verändert BES
Datenkonvert.	**- Ja**	**n Nein**

LAD_MV überträgt den Inhalt der Variablen auf 32 Merker. Die Übertragung erfolgt bitweise, d.h. das erste Bit der Variable wird auf den ersten Merker übertragen, das zweite Bit der Variable auf die zweite Variable usw.

Bit-Nummer	Wert dezimal	Wert hexadezimal	Merker-Nummer
Erstes Bit (Bit 0)	1	00000001	Erster Merker
Zweites Bit (Bit 1)	2	00000002	Zweiter Merker
Drittes Bit (Bit 2)	4	00000004	Dritter Merker
Viertes Bit (Bit 3)	8	00000008	Vierter Merker
Fünftes Bit (Bit 4)	16	00000010	Fünfter Merker
Sechstes Bit (Bit 5)	32	00000020	Sechster Merker
Siebtes Bit (Bit 6)	64	00000040	Siebter Merker
Achtes Bit (Bit 7)	128	00000080	Achter Merker
...	...	...	...
32. Bit (Bit 31)	2147483648	80000000	32. Merker

n Tabelle 12 – LAD_MV Übersicht

Operation

```
(Erster Merker + 0)  ç (Variable & 0x00000001)
(Erster Merker + 1)  ç (Variable & 0x00000002)
(Erster Merker + 2)  ç (Variable & 0x00000004)
...
(Erster Merker + 31) ç (Variable & 0x80000000)
```

Beispiel

```
// In diesem Beispiel verwenden wir den Befehl LAD_MV, um eine Variable in
// 32 Merker zu transportieren. Uns interessieren hier aber nur die ersten
// 16 Merker, deshalb maskieren wir die anderen mit UND_VV aus.

LAD_VA     V_MASKE,65535          // hex FFFF zum Ausmaskieren der oberen Bits
UND_VV     V_DATEN,V_MASKE        // Ausmaskieren der oberen 16 Bit (= Merker)
LAD_VV     V_DATEN,VARERG         // Ergebnis wieder in die Variable speichern
LAD_MV     ERSTER_MERKER,V_DATEN  // Übertragt die Variable auf 32 Merker
```

Hinweise

Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Bitte beachten Sie, daß die Merkerstartadresse –1 ein durch 8 teilbarer Wert sein muß, also z.B. 1, 9, 17 oder 33.

Siehe auch

Datenkonvertierungsbefehle (Seite 3)
LAD_VM (Seite 3)

n LAD_P1

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**Siehe Text**

Task starten: LAD_P1 Label K
Task beenden: LAD_P1 TASK_STOP K

LAD_P1 lädt den Programmzähler des ersten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.

Beispiel

```
LAD_P1     LABEL                  // Startet die Task1
```

Bitergebnisspeicher (BES)

Der Bitergebnisspeicher der aufrufenden Task bleibt unverändert. Innerhalb der neuen Task ist der Bitergebnisspeicher immer eingeschaltet.

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- Sie können auch innerhalb der Task mit dem LAD_P1-Befehl die Ausführungsposition verändern.
- Das erste Parallelprogramm ist das Programm, daß bei einem Neustart des Systems automatisch gestartet wird - sozusagen das Hauptprogramm. LAD_P1 bedeutet für ein SPS-Programm mit nur einer Task das gleiche wie der Befehl SPRING.

Siehe auch

Programmablaufbefehle (Seite 3)
LAD_P2 (Seite 3)
LAD_P3 (Seite 3)
LAD_P4 (Seite 3)

n LAD_P2

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**Siehe Text**

Task starten: LAD_P2 Label K
Task beenden: LAD_P2 TASK_STOP K

LAD_P2 lädt den Programmzähler des zweiten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.

Beispiel

```
LAD_P2     LABEL                  // Startet die Task2
```

Bitergebnisspeicher (BES)

Der Bitergebnisspeicher der aufrufenden Task bleibt unverändert. Innerhalb der neuen Task ist der Bitergebnisspeicher immer eingeschaltet.

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- Sie können auch innerhalb der Task mit dem LAD_P2-Befehl die Ausführungsposition verändern.

Siehe auch

Programmablaufbefehle (Seite 3)
LAD_P1 (Seite 3)
LAD_P3 (Seite 3)
LAD_P4 (Seite 3)

n LAD_P3

Task starten: LAD_P3 Label K
Task beenden: LAD_P3 TASK_STOP K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**Siehe Text**

LAD_P3 lädt den Programmzähler des dritten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.

Beispiel

```
LAD_P3     LABEL                  // Startet die Task3
```

Bitergebnisspeicher (BES)

Der Bitergebnisspeicher der aufrufenden Task bleibt unverändert. Innerhalb der neuen Task ist der Bitergebnisspeicher immer eingeschaltet.

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- Sie können auch innerhalb der Task mit dem LAD_P3-Befehl die Ausführungsposition verändern.

Siehe auch

Programmablaufbefehle (Seite 3)
LAD_P1 (Seite 3)
LAD_P2 (Seite 3)
LAD_P4 (Seite 3)

n LAD_P4

Task starten: LAD_P4 Label K
Task beenden: LAD_P4 TASK_STOP K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**Siehe Text**

LAD_P4 lädt den Programmzähler des vierten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.

Beispiel

```
LAD_P4     LABEL                  // Startet die Task4
```

Bitergebnisspeicher (BES)

Der Bitergebnisspeicher der aufrufenden Task bleibt unverändert. Innerhalb der neuen Task ist der Bitergebnisspeicher immer eingeschaltet.

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- Sie können auch innerhalb der Task mit dem LAD_P4-Befehl die Ausführungsposition verändern.

Siehe auch

Programmablaufbefehle (Seite 3)
LAD_P1 (Seite 3)
LAD_P2 (Seite 3)
LAD_P3 (Seite 3)

n LAD_VA

LAD_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

LAD_VA lädt den angegebenen Wert in die Variable.

Operation

```
(Variable)  ç  Wert
```

Beispiel

```
DEF_W      100,ZAHL                // ZAHL wird mit dem Wert 100 definiert

LAD_VA     V_TEST,ZAHL             // lädt W_ZAHL in die Variable V_TEST

// oder aber:

LAD_VA     V_TEST,100              // lädt den Wert 100 in die Variable V_TEST
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n LAD_VI

LAD_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

LAD_VI überträgt den Inhalt der durch den Zeiger bestimmten Variable in die angegebene Variable.

Operation

```
(Variable)  ç  (Zeiger  à  Variable)
```

Beispiel

```
// In diesem Beispiel sollen Werte aus einer Tabelle gelesen werden.

LAD_VA     V_ZEIGER,1800           // Tabelle beginnt bei Variable 1800
LAD_VI     V_WERT,V_ZEIGER         // lädt in die Variable V_WERT den Inhalt der
                                   // Variable 1800 auf die V_ZEIGER zeigt.
INC_V      V_ZEIGER,1              // Erhöht V_ZEIGER auf den nächsten Eintrag
                                   // in der Tabelle
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n LAD_VL

LAD_VL Variable V, Label K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**n Nein**

LAD_VL überträgt die Programmadresse des Labels in die angegebene Variable. Der Befehl LAD_VL ist notwendig, um ein Unterprogramm mit GEHUPRV anzuspringen.

Operation

(Variable) **ç** (Label-Adresse)

Beispiel

```
// In diesem Beispiel programmieren wir eine Schrittkette mit Hilfe der Befehle
// LAD_VL und GEHUPRV. Dabei wird deutlich, wie viel Programmcode durch die
// dynamische Zyklusprogrammierung gespart werden kann.

LAD_VL     V_ZYKLUS,Schritt1      // Zyklusvariable mit Label "Schritt1" belegen

Haupt:                            // Hauptprogrammschleife
GEHUPRV    V_ZYKLUS               // Aktuellen Schritt des Zyklus aufrufen
GEHURPI    Control                // Überwachungsfunktionen immer aufrufen
SPRING     Haupt                  // Schleife

Schritt1:                         // Erster Zyklus der Schrittkette
LAD_E      E_START                // Starttaster gedrückt?
NUND_E     E_NOTAUS               // und nicht Taster Notaus gedrückt?
STHOME     1,NORMAL               // dann Referenzstart starten
LAD_VL     V_ZYKLUS,Schritt2      // und nächsten Schritt aktivieren
UPREND                            // Ende Unterprogramm Schritt1

Schritt2:                         // Zweiter Schritt der Schrittkette
LAD_M      M_INPOS_A1             // Achse 1 Referenzfahrt beendet?
UPRENDN                           // Wenn nicht, dann im zweiten Schritt bleiben
STPABS     1,V_ZIELPOS_1          // Ansonsten Achse 1 auf Position starten
LAD_VL     V_ZYKLUS,Schritt3      // und nächsten Schritt aktivieren
UPREND                            // Ende Unterprogramm Schritt2

Schritt3:                         // Dritter Schritt der Schrittkette
LAD_M      M_INPOS_A1             // Achse 1 in Position?
UPRENDN                           // Nein, noch warten
EIN_A      A_ZYLINDER             // Zylinder ausfahren
LAD_VL     V_ZYKLUS,Schritt1      // Ablauf neu starten
UPREND                            // Unterprogramm Ende
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Programmablaufbefehle (Seite 3)
GEHUPRV (Seite 3)

n LAD_VM

LAD_VM Variable V, Erster Merker M

Gruppe	Abhängig von BES	Verändert BES
Datenkonvert.	**- Ja**	**n Nein**

LAD_VM überträgt den Zustand von 32 Merkern ab dem angegebenen ersten Merker in eine Variable. Die Übertragung erfolgt bitweise, d.h. der erste angegebene Merker wird in das erste Bit der Variable übertragen, der zweite Merker in das zweite Bit usw.

Merker-Nummer	Bit-Nummer	Wert dezimal	Wert hexadezimal
Erster Merker	Erstes Bit (Bit 0)	1	00000001
Zweiter Merker	Zweites Bit (Bit 1)	2	00000002
Dritter Merker	Drittes Bit (Bit 2)	4	00000004
Vierter Merker	Viertes Bit (Bit 3)	8	00000008
Fünfter Merker	Fünftes Bit (Bit 4)	16	00000010
Sechster Merker	Sechstes Bit (Bit 5)	32	00000020
Siebter Merker	Siebtes Bit (Bit 6)	64	00000040
Achter Merker	Achtes Bit (Bit 7)	128	00000080
...	...	...	...
32. Merker	32. Bit (Bit 31)	2147483648	80000000

n Tabelle 13 – LAD_VM Übersicht

Operation

```
(Bit 0 der Variable)  ç (Erster Merker + 0)
(Bit 1 der Variable)  ç (Erster Merker + 1)
(Bit 2 der Variable)  ç (Erster Merker + 2)
...
(Bit 31 der Variable) ç (Erster Merker + 31)
```

Beispiel

```
// In diesem Beispiel verwenden wir den Befehl LAD_VM, um 32 Merker in einer
// Variable zu transportieren. Uns interessieren hier aber nur die ersten
// 16 Merker, deshalb maskieren wir die anderen mit UND_VV aus.

LAD_VM     V_DATEN,ERSTER_MERKER   // Übertragen von 32 Merkern in die Variable
UND_VA     V_DATEN,0xFFFF          // Ausmaskieren der oberen 16 Bit (= Merker)
LAD_VV     V_DATEN,VAREG           // Ergebnis wieder in die Variable speichern
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- Bitte beachten Sie, daß die Merkerstartadresse –1 ein durch 8 teilbarer Wert sein muß, also z.B. 1, 9, 17 oder 33. Jede andere Merkerangabe führt zu einem Compilerfehler.

Siehe auch

Datenkonvertierungsbefehle (Seite 3)
LAD_MV (Seite 3)

n LAD_VV

LAD_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

LAD_VV lädt den Inhalt der Variable2 in die Variable1.

Operation

`(Variable1) ç (Variable2)`

Beispiel

```
DEF_V     10,V_TEST             // Weist der Variable 10 den Namen V_TEST zu
DEF_V     11,V_LAENGE           // weist der Variable 11 den Namen V_LAENGE zu
LAD_VV    V_TEST,V_LAENGE       // lädt den Inhalt der Variable V_LAENGE in
                                // die Variable V_TEST
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n MACRO

Name K MACRO [Par 1 K, [Par 2 K, [...]]]

Gruppe	Abhängig von BES	Verändert BES
Makros	**y Nein**	**n Nein**

MACRO bestimmt den Anfang einer Makro-Definition. Innerhalb des Makro-Definitionsblocks geben Sie dann den Namen des Makros, etwaige Parameter und den zugeordneten SPS-Programmcode an. Die eckigen Klammern in der Syntax deuten an, daß Sie hier selbst entscheiden können, wieviele Parameter Sie an einen Makrobefehl übergeben möchten.

Beispiel ohne Parameter

```
// BES_EIN schaltet den Bitergebnisspeicher ein. Es werden keine Parameter benötigt

BES_EIN   MACRO                 // Makro BES_EIN definieren
LAD_M     M_EIN                 // Quelltext des Makro
ENDM                            // Makro Ende
```

Beispiel mit Parametern

```
// Folgendes Makro startet einen Systemtimer mit einem
// angegebenen Wert. Das Makro erhält als Parameter die
// Timernummer sowie die Startzeit für diesen Timer.

TSTART    MACRO TIM,KONST       // Makro TSTART mit den Parametern TIM
                                // und KONST definieren
LAD_VA    V_&TIM,KONST          // Der Parameter TIM wird expandiert in
                                // V_&TIM, wobei &TIM für die übergebene
                                // Zeichenkette steht. Der Parameter
                                // KONST wird unverändert übernommen.
EIN_M     M_&TIM                // Der Parameter TIM wird ein zweites Mal
                                // expandiert, diesmal zu M_&TIM
ENDM                            // Ende der Makrodefinition
```



```
// Zum Aufruf des Makros aus dem SPS-Programm schreiben wir:

TSTART     TIM_15,2000

// was für uns bedeutet: startet den Timer 15 mit dem Wert 2000. Der
// Compiler übersetzt den Makroaufruf in folgenden Programmcode:

LAD_VA     V_TIM_15,2000
EIN_M      M_TIM_15

// Die übergebenen Parameter werden expandiert und zu neuen Werten umgewandelt.
```

Hinweise

- Makros sind keine Funktionen im eigentlichen Sinn! Es wird mit einem Makro also kein Unterprogramm aufgerufen, sondern vielmehr der vollständige Quellcode des Makros anstelle des Makroaufrufs compiliert. Ein zwanzig Zeilen langes Makro belegt bei jedem Aufruf exakt diese zwanzig Zeilen SPS-Programmspeicher.
- Bitte beachten Sie, daß Makronamen stets nur aus den Buchstaben A-Z und den Ziffern 0-9 bestehen dürfen. Der Makroname darf nicht mit einer Zahl beginnen. Leerzeichen und sonstige Sonderzeichen sind innerhalb des Makronamens nicht zulässig. Gleiches gilt für die Namen des Parameter.

Siehe auch

Makros (Seite 3)
ENDM (Seite 3)
EXITM (Seite 3)

n MOD_A

MOD_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**n Nein**

MOD_A überträgt den Zustand des Bitergebnisspeichers auf den angegebenen Ausgang.

Operation

`(Ausgang)` **ç** `(Bitergebnisspeicher)`

Beispiel

```
BES_EIN                          // Schaltet Bitergebnisspeicher ein
MOD_A      A_TEST                // Speichert den Inhalt des Bitergebnisspeichers
                                 // auf den Ausgang
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt. Die Auswirkungen dieses Befehls jedoch hängen vom Zustand des Bitergebnisspeichers ab.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n MOD_AI

MOD_AI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**n Nein**

MOD_AI überträgt den Zustand des Bitergebnisspeichers in den durch den angegebenen Zeiger bestimmten Ausgang.

Operation

`(Zeiger à Ausgang) ç (Bitergebnisspeicher)`

Beispiel

```
LAD_VA     V_AUSGANG,W_TAB        // Lädt Konstante W_TAB in die Variable V_AUSGANG
MOD_AI     V_AUSGANG              // speichert den Inhalt des Bitergebnisspeichers
                                  // auf einen Ausgang dessen Nummer in der
                                  // Variablen enthalten ist
```

Hinweise

- **n** Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt. Die Auswirkungen dieses Befehls jedoch hängen vom Zustand des Bitergebnisspeichers ab.
- **n** Verwenden Sie MOD_AI um den Inhalt des Bitergebnisspeichers auf einen Ausgang zu übertragen, dessen Nummer bei der Programmerstellung noch nicht feststeht.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n MOD_M

MOD_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**n Nein**

MOD_M überträgt den Zustand des Bitergebnisspeichers in den angegebenen Merker.

Operation

`(Merker) ç (Bitergebnisspeicher)`

Beispiel

```
MOD_M      M_TEST                 // Übertragt den Bitergebnisspeicher auf M_TEST
```

Hinweise

- **n** Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt. Die Auswirkungen dieses Befehls jedoch hängen vom Zustand des Bitergebnisspeichers ab.

Siehe auch

Merkerbefehle (Seite 3)

n MOD_MI

MOD_MI Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**n Nein**

MOD_MI überträgt den Zustand des Bitergebnisspeichers in den durch den angegebenen Zeiger bestimmten Merker.

Operation

```
(Zeiger à Merker) ç (Bitergebnisspeicher)
```

Beispiel

```
MOD_MI     V_MERKER              // der Inhalt des Bitergebnisspeichers wird in
                                 // den Merker übertragen, auf den V_MERKER zeigt
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt. Die Auswirkungen dieses Befehls jedoch hängen vom Zustand des Bitergebnisspeichers ab.
- Verwenden Sie MOD_MI um den Inhalt des Bitergebnisspeichers auf einen Merker zu übertragen, dessen Nummer bei der Programmerstellung noch nicht feststeht.

Siehe auch

Merkerbefehle (Seite 3)

n MUL_IV

MUL_IV Zeiger V**, Variable** V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

MUL_IV multipliziert den Inhalt der durch den Zeiger bestimmten Variable mit dem Inhalt der angegebenen Variable. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.

Operation

```
(VARERG)    ç  (Zeiger à Variable) * (Variable)
(MUL_REST)  ç ((Zeiger à Variable) * (Variable)) >> 32
```

Beispiel

```
MUL_IV     V_ZEIGER,V_TEST       // Der Inhalt der Variable, auf die V_ZEIGER
                                 // zeigt, wird mit V_TEST multipliziert. Das
                                 // Ergebnis wird in VARERG gespeichert
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n MUL_VA

MUL_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

MUL_VA multipliziert den Inhalt der Variable mit dem angegebenen Wert. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.

Operation

```
(VARERG)   ç  (Variable) * Wert
(MUL_REST) ç ((Variable) * Wert) >> 32
```

Beispiel

```
DEF_W     10,ZAHL               // Definiert ZAHL mit dem Wert 10
MUL_VA    V_TEST,ZAHL           // Multipliziert die Variable mit dem Wert ZAHL.
                                // Das Ergebnis wird in VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n MUL_VI

MUL_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

MUL_VI multipliziert den Inhalt der durch den Zeiger bestimmten Variable mit dem Inhalt der angegebenen Variable. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.

Operation

```
(VARERG)   ç  (Variable) * (Zeiger à Variable)
(MUL_REST) ç ((Variable) * (Zeiger à Variable)) >> 32
```

Beispiel

```
MUL_VI    V_TEST,V_ZEIGER       // V_TEST wird mit dem Inhalt der Variable, auf
                                // die V_ZEIGER zeigt, multipliziert. Das
                                // Ergebnis wird in VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n MUL_VV

MUL_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

MUL_VV multipliziert die Inhalte von Variable1 und Variable2. Das Ergebnis der Multiplikation wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.

Operation

```
(VARERG)    ç  (Variable1) * (Variable2)
(MUL_REST)  ç  ((Variable1) * (Variable2)) >> 32
```

Beispiel

```
LAD_VA     V_ZAHL1,10            // In die Variable V_ZAHL1 wird der
                                 // Wert 10 geladen
LAD_VA     V_ZAHL2,20            // in die Variable V_ZAHL2 wird der
                                 // Wert 20 geladen
MUL_VV     V_ZAHL1,V_ZAHL2       // multipliziert die Variable V_ZAHL1 mit der
                                 // Variable V_ZAHL2 und das Ergebnis wird in
                                 // VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n NLAD_A

NLAD_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

NLAD_A lädt den invertierten Zustand des Ausgangs in den Bitergebnisspeicher.

Operation

```
(Bitergebnis)  ç  ~(Ausgang)
```

Beispiel

```
NLAD_A     A_TEST                // Lädt den invertierten Zustand von A_TEST
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n NLAD_E

NLAD_E Eingang E

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

NLAD_E lädt den invertierten Zustand eines Eingangs in den Bitergebnisspeicher.

Operation

(Bitergebnis) **ç** ~(Eingang)

Beispiel

```
NLAD_E     E_TEST                  // Lädt den invertierten Zustand von E_TEST
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n NLAD_M

NLAD_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

NLAD_M lädt den invertierten Zustand des Merkers in den Bitergebnisspeicher.

Operation

(Bitergebnis) **ç** ~(Merker)

Beispiel

```
NLAD_M     M_TEST                  // Lädt den invertierten Zustand von M_TEST
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)

n NODER_A

NODER_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

NODER_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Ausgangs mit einem logischen ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

```
(Bitergebnis) ç (Bitergebnis) | ~(Ausgang)
```

Beispiel

```
LAD_A      A_TEST1                 // Wenn A_TEST1
NODER_A    A_TEST2                 // und A_TEST2 ausgeschaltet sind
SPRINGJ    TESTZYKLUS              // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n NODER_E

NODER_E Eingang E

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

NODER_E verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Eingangs mit einem logischen ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

```
(Bitergebnis) ç (Bitergebnis) | ~(Eingang)
```

Beispiel

```
LAD_E      E_TEST1                 // Wenn E_TEST1
NODER_E    E_TEST2                 // und E_TEST2 ausgeschaltet sind
SPRINGJ    TESTZYKLUS              // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n NODER_M

NODER_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

NODER_M verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Merkers mit einem logischen ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) | ~(Merker)`

Beispiel

```
LAD_M     M_TEST1              // Wenn M_TEST1 und
NODER_M   M_TEST2              // M_TEST2 ausgeschaltet sind,
SPRINGJ   TESTZYKLUS           // dann springe zu Label TESTZYKLUS
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)

n NUND_A

NUND_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

NUND_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Ausgangs mit einem logischen UND. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) & ~(Ausgang)`

Beispiel

```
LAD_A     A_TEST1              // Wenn A_TEST1 oder A_TEST2 eingeschaltet ist
NUND_A    A_TEST2              // oder beide Ausgänge ausgeschaltet sind,
SPRINGJ   TESTZYKLUS           // dann springe zu Label TESTZYKLUS
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n NUND_E

NUND_E Eingang E

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

NUND_E verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Eingangs mit einem logischen UND. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

```
(Bitergebnis) ç (Bitergebnis) & ~(Eingang)
```

Beispiel

```
LAD_E      E_TEST1                // Wenn E_TEST1 oder ETEST2 eingeschaltet ist
NUND_E     E_TEST2                // oder beide Eingänge ausgeschaltet sind,
SPRINGJ    TESTZYKLUS             // dann springe zu Label TESTZYKLUS
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n NUND_M

NUND_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

NUND_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Merkers mit einem logischen UND. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

```
(Bitergebnis) ç (Bitergebnis) & ~(Merker)
```

Beispiel

```
LAD_M      M_TEST1                // Wenn M_TEST1 oder M_TEST2 eingeschaltet ist
NUND_M     M_TEST2                // oder beide Merker ausgeschaltet sind,
SPRINGJ    TESTZYKLUS             // dann springe zu Label TESTZYKLUS
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)

n ODER_A

ODER_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

ODER_A verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Ausgangs mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) | (Ausgang)`

Beispiel

```
LAD_A     A_TEST1              // Wenn A_TEST1 oder A_TEST2 eingeschaltet ist
ODER_A    A_TEST2              // oder beide Ausgänge eingeschaltet sind,
SPRINGJ   TESTZYKLUS           // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n ODER_E

ODER_E Eingang E

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

ODER_E verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Eingangs mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) | (Eingang)`

Beispiel

```
LAD_E     E_TEST1              // Wenn E_TEST1 oder E_TEST2 eingeschaltet ist
ODER_E    E_TEST2              // oder beide Eingänge eingeschaltet sind,
SPRINGJ   TESTZYKLUS           // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n ODER_M

ODER_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

ODER_M verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Merkers mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) | (Merker)`

Beispiel

```
LAD_M     M_TEST1              // Wenn M_TEST1
ODER_M    M_TEST2              // oder M_TEST2 eingeschaltet ist,
SPRINGJ   TESTZYKLUS           // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)

n ODER_IV

ODER_IV Zeiger V, Variable V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

ODER_IV verknüpft die Inhalte der durch den Zeiger bestimmten und der angegebenen Variable mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.

Operation

`(VARERG) ç (Zeiger à Variable) | (Variable)`

Beispiel

```
// In diesem Beispiel haben wir ein Variablenfeld, welches z.B. vom PC geschrieben
// wird, in diesem Variablenfeld werden die unteren 16 Bit alle gesetzt. Das
// Ergebnis der Verknüpfung wird wieder in der Tabelle gespeichert.

LAD_VA    V_ZEIGER,2000        // Zeiger-Variable initialisieren
LAD_VA    V_MASKE,0xFFFF       // Maskieren mit Hex FFFF

LOOP:
ODER_IV   V_ZEIGER,V_MASKE     // Verknüpfung durchführen
LAD_IV    V_ZEIGER,VARERG      // Ergebnis der Verknüpfung wieder in Tabelle
INC_V     V_ZEIGER,1           // Tabellenzeiger auf nächsten Eintrag
SPRING    LOOP                 // Zurück zur Schleife
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n ODER_VA

ODER_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

ODER_VA verknüpft den Inhalt der Variablen und den angegebenen Wert mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.

Operation

```
(VARERG) ç (Variable) | Wert
```

Beispiel

```
// In diesem Beispiel verwenden wir den Befehl LAD_VM, um 32 Merker in einer
// Variable zu transportieren. Die ersten 16 Merker sollen gesetzt werden,
// deshalb Verknüpfen wir die Merker mit ODER_VA.

LAD_VM     V_DATEN,ERSTER_MERKER   // Übertragen von 32 Merkern in die Variable
ODER_VA    V_DATEN,0xFFFF          // mit Hex FFFF die unteren 16 Bit setzen
LAD_VV     V_DATEN,VARERG          // Ergebnis wieder in Variable speichern
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n ODER_VI

ODER_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

ODER_VI verknüpft die Inhalte der durch den Zeiger bestimmten und der angegebenen Variable mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.

Operation

```
(VARERG) ç (Variable) | (Zeiger à Variable)
```

Beispiel

```
// In diesem Beispiel in einem Variablenfeld die unteren 16 Bit werden gesetzt.
// Das Ergebnis der wird wieder in der Tabelle gespeichert.

LAD_VA     V_ZEIGER,2000           // Zeiger-Variable initialisieren
LAD_VA     V_MASKE,0xFFFF          // untere 16 Bit maskieren

LOOP:
ODER_VI    V_MASKE,V_ZEIGER        // Verknüpfung durchführen
LAD_IV     V_ZEIGER,VARERG         // Ergebnis der Verknüpfung wieder in Tabelle
INC_V      V_ZEIGER,1              // Tabellenzeiger auf nächsten Eintrag
SPRING     LOOP                    // Zurück zur Schleife
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n ODER_VV

ODER_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

ODER_VV der beiden angegebenen Variablen mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.

Operation

```
(VARERG) ç (Variable1) | (Variable2)
```

Beispiel

```
// In diesem Beispiel verwenden wir den Befehl LAD_VM, um 32 Merker in einer
// Variable zu transportieren. Die ersten 16 Merker sollen gesetzt werden.

LAD_VA     V_MASKE,0xFFFF          // hex FFFF zum der unteren 16 Bits
LAD_VM     V_DATEN,ERSTER_MERKER   // Übertragen von 32 Merkern in die Variable
ODER_VV    V_DATEN,V_MASKE         // mit V_MASKE werden die unteren 16 Bit
                                   // (= Merker) gesetzt
LAD_VV     V_DATEN,VARERG          // Ergebnis wieder in Variable speichern
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SLL_VA

SLL_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SLL_VA schiebt den Inhalt der Variable, die als erster Parameter angegeben wird, um den angegebenen Wert nach links. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.

Operation

```
(VARERG) = (Variable) << Wert
```

Beispiel

```
// Wir schieben mit SLL_V den Wert einer Variable um drei Bits nach links.

LAD_VA     V_ZAHL,1                // V_ZAHL mit dem Wert 1 laden
SLL_VA     V_ZAHL,3                // Inhalt von V_ZAHL (=0001) um drei Bit nach
                                   // links schieben, Ergebnis ist gleich 8(=1000)
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SLL_VV

SLL_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SLL_VV schiebt den Inhalt der Variable1 um den Inhalt der Variable2 nach links. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.

Operation

```
(VARERG) = (Variable1) << (Variable2)
```

Beispiel

```
// Wir schieben mit SLL_VV den Wert einer Variable um drei Bits nach links.

LAD_VA     V_ZAHL,1               // V_ZAHL mit dem Wert 1 laden
LAD_VA     V_ANZAHL,3             // V_ANZAHL mit dem Wert 3 laden
SLL_VV     V_ZAHL,V_ANZAHL        // Inhalt von V_ZAHL (=0001) um drei Bit nach
                                  // links schieben, Ergebnis ist gleich 8(=1000)
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SPRING

SPRING Label K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**y Nein**	**n Nein**

SPRING setzt den Programmablauf ab dem angegebenen Label fort.

Operation

```
Programmadresse ç Label-Adresse
(Bitergebnis)   ç EIN
```

Beispiel

```
SPRING     INIT_TIMER             // Springt im Programm zum Label

INIT_TIMER:                       // Programm INIT_TIMER
...                               // Programmcode
```

Siehe auch

Programmablaufbefehle (Seite 3)

n SPRINGJ

SPRINGJ Label K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**n Nein**

SPRINGJ setzt den Programmablauf ab dem angegebenen Label fort, wenn der Bitergebnisspeicher zu diesem Zeitpunkt eingeschaltet ist.

Operation

```
Programmadresse  ç  Label-Adresse
(Bitergebnis)    ç  EIN
```

Beispiel

```
SPRINGJ    INIT_TIMER            // Springt im Programm zum Label INIT_TIMER,
                                 // wenn der Bitergebnisspeicher eingeschaltet ist
INIT_TIMER:                      // Programm INIT_TIMER
...                              // Programmcode
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Programmablaufbefehle (Seite 3)

n SPRINGN

SPRINGN Label K

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**n Nein**

SPRINGN setzt den Programmablauf ab der angegebenen Label fort, wenn der Bitergebnisspeicher zu diesZeitpunkt ausgeschaltet ist.

Operation

```
Programmadresse  ç  Label-Adresse
(Bitergebnis)    ç  EIN
```

Beispiel

```
SPRINGJ    INIT_TIMER            // Springt im Programm zum Label INIT_TIMER
                                 // wenn der Bitergebnisspeicher ausgeschaltet ist
INIT_TIMER:                      // Programm INIT_TIMER
...                              // Programmcode
```

Hinweise

n Dieser Befehl wird nur ausgeführt, das der Bitergebnisspeicher ausgeschaltet ist.

Siehe auch

Programmablaufbefehle (Seite 3)

n SRL_VA

SRL_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SRL_VA schiebt den Inhalt der Variable um den angegeben Wert nach rechts. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.

Operation

`(VARERG) ç (Variable) >> Wert`

Beispiel

```
// Wir schieben mit SRL_V den Wert einer Variable um drei Bits nach rechts.

LAD_VA     V_ZAHL,8               // V_ZAHL mit dem Wert 8 laden
SRL_VA     V_ZAHL,3               // Inhalt von V_ZAHL (=1000) um drei Bit nach
                                  // rechts schieben, Ergebnis ist gleich 1(=0001)
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SRL_VV

SRL_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SRL_VV schiebt den Inhalt der Variable1 um den Inhalt der Variable2 nach rechts. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.

Operation

`(VARERG) ç (Variable1) >> (Variable2)`

Beispiel

```
// Wir schieben mit SRL_VV den Wert einer Variable um drei Bits nach rechts.

LAD_VA     V_ZAHL,8               // V_ZAHL mit dem Wert 8 laden
LAD_VA     V_ANZAHL,3             // V_ANZAHL mit dem Wert 3 laden
SRL_VV     V_ZAHL,V_ANZAHL        // Inhalt von V_ZAHL (=1000) um drei Bit nach
                                  // rechts schieben, Ergebnis ist gleich 1(=0001)
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SUB_IV

SUB_IV Zeiger V, Variable V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SUB_IV subtrahiert vom Inhalt der durch den Zeiger bestimmten Variable den Inhalt der angegebenen Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.

Operation

```
(VARERG) ç (Zeiger à Variable) – (Variable)
```

Beispiel

```
SUB_IV     V_ZEIGER,V_TEST          // Vom Inhalt der Variable, auf die V_ZEIGER
                                    // zeigt, wird V_TEST subtrahiert. Das Ergebnis
                                    // wird in VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SUB_VA

SUB_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SUB_VA subtrahiert vom Inhalt der Variable den angegebenen Wert. Das Ergebnis der Subtraktion wird in VARERG gespeichert.

Operation

```
(VARERG) ç (Variable) – Wert
```

Beispiel

```
SUB_VA     V_TEST,10                // Subtrahiert 10 vom V_TEST und speichert das
                                    // Ergebnis in VARERG
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SUB_VI

SUB_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SUB_VI subtrahiert vom Inhalt der angebenen Variable den Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.

Operation

`(VARERG) ç (Variable) – (Zeiger à Variable)`

Beispiel

```
SUB_VI     V_TEST,V_ZEIGER         // Subtrahiert den Inhalt der Variable, auf die
                                   // V_ZEIGER zeigt, vom Inhalt der Variable V_TEST
                                   // Das Ergebnis wird in VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n SUB_VV

SUB_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

SUB_VV subtrahiert vom Inhalt der Variable1 den Inhalt der Variable2. Das Ergebnis der Subtraktion wird in VARERG gespeichert.

Operation

`(VARERG) ç (Variable1) – (Variable2)`

Beispiel

```
LAD_VA     V_ZAHL1,20              // Variable V_ZAHL1 mit dem Wert 20 laden
LAD_VA     V_ZAHL2,10              // Variable V_ZAHL2 mit dem Wert 10 laden
SUB_VV     V_ZAHL1,V_ZAHL2         // subtrahiert von der Variable V_ZAHL1 die
                                   // Variable V_ZAHL2 und das Ergebnis wird in
                                   // VARERG gespeichert
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n TEXT

TEXT

Gruppe	Abhängig von BES	Verändert BES
Compiler	**y Nein**	**n Nein**

TEXT deklariert alle folgenden Informationen in der aktuellen Datei bis zum Dateiende als Textdefinition.

Beispiel

```
TEXT

// Ab hier beginnen die Textdefinitionen. Bis zum Ende der Datei dürfen nur noch
// Texte erscheinen

TEXT001    "********************" ;
TEXT002    " MICRO DESIGN GmbH  " ;
TEXT003    "       MC-1A        " ;
TEXT004    "********************" ;
TEXT005    "Version 1.0 30.10.98" ;
TEXT006    "TEXT006             " ;
TEXT007    "" ;
TEXT008    "TEXT008             " ;
```

Sonderzeichen im Text

Sie können spezielle Steuerzeichen oder nichtdruckbare Grafikzeichen innnerhalb eines Textes verwenden. Die Syntax dieser Sonderzeichen orientiert sich an der Sprache C++:

- Zeilenumbruch (CRLF): Zeichenfolge "\n", z.B. "Auswertung\n"
- Zeilenvorschub (LF): Zeichenfolge "\r", z.B. "Neue Zeile\r"
- Sämtliche anderen Zeichencodes: Zeichen "\" gefolgt vom numerischen ASCII-Code des gewünschten Zeichens, z.B. "\025" für das Zeichen mit dem ASCII-Code 25, "\128" für das Zeichen mit dem ASCII-Code 128 usw. Sie können den ASCII-Code auch in hexedezimaler Schreibweise angeben. Stellen Sie hierzu dem Code einfach statt "\" die Kombination "\x" voraus, also z.B. "\x3A" für das Zeichen mit dem ASCII-Code 3A hex.

Hinweise

- Sie können auch mehrere Dateien zur Definition von Texten verwenden.
- Bitte beachten Sie, daß nach dem Befehl "TEXT" keine anderen SPS-Befehle mehr folgen dürfen. Auch symbolische Definitionen werden im Anschluß bis zum Ende der Datei vollständig ignoriert.
- Sie sollten darauf achten, daß der Compiler die Textnummern kontinuierlich durchnummeriert. D.h. wie im oben gezeigten Beispiel sind die Textnummern nur gültig wenn Sie bei 1 beginnen und kontinuierlich durchnummerieren.
- Sie können die Texte in einer gesonderten Datei abspeichern, oder zusammen mit Ihrem Quellcode in der gleichen Datei ablegen. Achten Sie bei der zweiten Möglichkeit bitte darauf, daß nach dem Befehl "TEXT" keine weiteren SPS-Befehle mehr folgen dürfen.

Siehe auch

Fehler! Verweisquelle konnte nicht gefunden werden. (Seite **Fehler! Textmarke nicht definiert.**)
Fehler! Verweisquelle konnte nicht gefunden werden. (Seite **Fehler! Textmarke nicht definiert.**)

n UND_A

UND_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

UND_A verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Ausgangs mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) & (Ausgang)`

Beispiel

```
LAD_A     A_TEST1              // Wenn A_TEST1 eingeschaltet
UND_A     A_TEST2              // und A_TEST2 eingeschaltet
SPRINGJ   TESTZYKLUS           // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n UND_E

UND_E Eingang E

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

UND_E verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Eingangs mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) & (Eingang)`

Beispiel

```
LAD_E     E_TEST1              // Wenn E_TEST1 eingeschaltet
UND_E     E_TEST2              // und E_TEST2 eingeschaltet
SPRINGJ   TESTZYKLUS           // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n UND_IV

UND_IV Zeiger V, Variable V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

UND_IV verknüpft den Inhalt der durch den Zeiger bestimmten Variable und den Inhalt der angegebenen Variable mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.

Operation

`(VARERG) ç (Zeiger à Variable) & (Variable)`

Beispiel

```
// In diesem Beispiel haben wir ein Variablenfeld, welches z.B. vom PC geschrieben
// wird und maskieren aus diesem Variablenfeld die oberen 16 Bit aus (die geben
// andere Informationen an).

DEF_W     2000,TABELLE_START      // Anfang des Variablenfeldes
LAD_VA    V_ZEIGER,TABELLE_START  // Zeiger-Variable initialisieren
LAD_VA    V_MASKE,65535           // entspricht hexadezimal FFFF

LOOP:
UND_IV    V_ZEIGER,V_MASKE        // Verknüpfung durchführen
LAD_IV    V_ZEIGER,VARERG         // Ergebnis der Verknüpfung wieder in Tabelle
INC_V     V_ZEIGER,1              // Tabellenzeiger auf nächsten Eintrag
SPRING    LOOP                    // Zurück zur Schleife
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n UND_M

UND_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

UND_M verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Merkers mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.

Operation

`(Bitergebnis) ç (Bitergebnis) & (Merker)`

Beispiel

```
LAD_M     M_TEST1                 // Wenn M_TEST1
UND_M     M_TEST2                 // und M_TEST2 eingeschaltet sind,
SPRINGJ   TESTZYKLUS              // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)

n UND_VA

UND_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

UND_VA verknüpft den Inhalt der Variable und den angegebenen Wert mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.

Operation

`(VARERG) ç (Variable) & Wert`

Beispiel

```
// In diesem Beispiel verwenden wir den Befehl LAD_VM, um 32 Merker in einer
// Variable zu transportieren. Uns interessieren hier aber nur die ersten
// 16 Merker, deshalb maskieren wir die anderen mit UND_VA aus.

LAD_VM     V_DATEN,ERSTER_MERKER   // Übertragen von 32 Merkern in die Variable
UND_VA     V_DATEN,0xFFFF          // mit Hex FFFF die oberen 16 Bit ausmaskieren
LAD_VV     V_DATEN,VARERG          // Ergebnis wieder in Variable speichern
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n UND_VI

UND_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

UND_VI verknüpft die Inhalte der angegebenen Variable und der durch den Zeiger bestimmten Variable mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.

Operation

`(VARERG) ç (Variable) & (Zeiger à Variable)`

Beispiel

```
// In diesem Beispiel haben wir ein Variablenfeld, welches z.B. vom PC geschrieben
// wird, und maskieren aus diesem Variablenfeld die oberen 16 Bit aus (die geben
// andere Informationen an).

LAD_VA     V_ZEIGER,2000           // Zeiger-Variable initialisieren
LAD_VA     V_MASKE,0xFFFF          // Maske für die unteren 16 Bit

LOOP:
UND_VI     V_MASKE,V_ZEIGER        // Verknüpfung durchführen
LAD_IV     V_ZEIGER,VARERG         // Ergebnis der Verknüpfung wieder in Tabelle
INC_V      V_ZEIGER,1              // Tabellenzeiger auf nächsten Eintrag
SPRING     LOOP                    // Zurück zur Schleife
```

Hinweise

n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n UND_VV

UND_VA Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	**- Ja**	**n Nein**

UND_VV verknüpft die Inhalte der beiden angegebenen Variablen mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.

Operation

```
(VARERG)  ç  (Variable1) & (Variable2)
```

Beispiel

```
// In diesem Beispiel verwenden wir den Befehl LAD_VM, um 32 Merker in einer
// Variable zu transportieren. Uns interessieren hier aber nur die ersten
// 16 Merker, deshalb maskieren wir die anderen mit UND_VV aus.

LAD_VA     V_MASKE,65535          // hex FFFF zum Ausmaskieren der oberen Bits
LAD_VM     V_DATEN,ERSTER_MERKER  // Übertragen von 32 Merkern in die Variable
UND_VV     V_DATEN,V_MASKE        // Ausmaskieren der oberen 16 Bit (= Merker)
LAD_VV     V_DATEN,VARERG         // Ergebnis wieder in die Variable speichern
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Variablenbefehle (Seite 3)

n UPREND

UPREND

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**y Nein**	**þ Ja**

UPREND beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.

Operation

```
Programmadresse  ç  (Stackpointer)
Stackpointer     ç  Stackpointer + 2
(Bitergebnis)    ç  EIN
```

Beispiel

```
GEHUPR     UPROG                  // Ruft Unterprogramm UPROG auf
SPRING     LOOP

UPROG:                            // Unterprogramm UPROG
...                               // Programmcode
UPREND                            // Beendet das Unterprogramm
                                  // Der Bitergebnisspeicher ist danach immer EIN
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.
- Der Bitergebnisspeicher ist der Rückkehr aus dem Unterprogramm eingeschaltet.

Siehe auch

Programmablaufbefehle (Seite 3)

n UPRENDJ

UPRENDJ

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**þ Ja**

UPRENDJ beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.

Operation

```
Programmadresse  ç  (Stackpointer)
Stackpointer     ç  Stackpointer + 2
(Bitergebnis)    ç  EIN
```

Beispiel

```
GEHUPR     UPROG                  // Ruft Unterprogramm UPROG auf
SPRING     LOOP

UPROG:                            // Unterprogramm
...                               // Programmcode
UPRENDJ                           // Beendet Unterprogramm, wenn Bitergebnis ein
                                  // Der Bitergebnisspeicher ist danach immer ein
```

Hinweise

- Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.

Siehe auch

Programmablaufbefehle (Seite 3)

n UPRENDN

UPRENDN

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**þ Ja**

UPRENDN beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.

Operation

```
Programmadresse  ç  (Stackpointer)
Stackpointer     ç  Stackpointer + 2
(Bitergebnis)    ç  EIN
```

Beispiel

```
GEHUPR     UPROG                  // Ruft Unterprogramm UPROG auf
SPRING     LOOP

UPROG:                            // Unterprogramm
...
UPRENDN                           // Beendet Unterprogramm, wenn Bitergebnis aus
                                  // Der Bitergebnisspeicher ist danach immer EIN
```

Hinweise

- Dieser Befehl wird nur ausgeführt, wenn der Bitergebnisspeicher ausgeschaltet ist.
- Der Bitergebnisspeicher ist nach der Rückkehr aus dem Unterprogramm eingeschaltet.

Siehe auch

Programmablaufbefehle (Seite 3)

n VERG_IV

VERG_IV Zeiger V, Variable V

Gruppe	Abhängig von BES	Verändert BES
Variablenvergleich	**y Nein**	**n Nein**

VERG_IV vergleicht den Inhalt der durch den Zeiger bestimmten Variable mit dem Inhalt der angegebenen Variable. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:

Merker	Wird gesetzt wenn...
M_GLEICH	...die Inhalte der durch den Zeiger bestimmten Variable und der angegebenen Variable gleich sind
M_KLEINER	... der Inhalt der durch den Zeiger bestimmten Variable kleiner ist als der Inhalt der angegebenen Variable
M_GROESSER	...der Inhalt der durch den Zeiger bestimmten Variable größer ist als der Inhalt der angegebenen Variable

n Tabelle 14 – Variablenvergleichsmerker bei VERG_IV

Operation

```
(M_GLEICH)    ç (Zeiger à Variable) = (Variable)
(M_KLEINER)   ç (Zeiger à Variable) < (Variable)
(M_GROESSER)  ç (Zeiger à Variable) > (Variable)
```

Beispiel

```
// In diesem Beispiel haben wir eine Variable, diese vergleichen wir mit einem
// Variablenfeld, welches z.B. vom PC geschrieben ist.

DEF_W      2000,TABELLE1          // Anfang des ersten Variablenfeldes
DEF_W      3000,TABELLE2          // Anfang des zweiten Variablenfeldes
LAD_VA     V_VERGLEICH,1234       // die Variable V_VERGLEICH hat den Inhalt 1234
LAD_VA     V_ZEIGER1,TABELLE1     // Zeiger-Variable1 initialisieren
LAD_VA     V_ZEIGER2,TABELLE2     // Zeiger-Variable2 initialisieren

LOOP:
VERG_VI    V_ZEIGER1,V_VERGLEICH  // Vergleich durchführen
LAD_M      M_GLEICH               // Übereinstimmender Eintrag gefunden?
SPRINGJ    FERTIG                 // Wenn ja, Springe zum Label "Fertig"
INC_V      V_ZEIGER1,1            // Tabellenzeiger1 auf nächsten Eintrag
INC_V      V_ZEIGER2,1            // Tabellenzeiger2 auf nächsten Eintrag
SPRING     LOOP                   // Zurück zur Schleife
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Variablenvergleichsbefehle (Seite 3)

n VERG_VA

VERG_VA Variable V, Wert K

Gruppe	Abhängig von BES	Verändert BES
Variablenvergleich	**y Nein**	**n Nein**

VERG_VA vergleicht den Inhalt der angegebenen Variable mit dem angegebenen Wert. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:

Merker	Wird gesetzt wenn...
M_GLEICH	...der Inhalt des angegebenen Variable dem Wert entspricht
M_KLEINER	...der Inhalt der angegebenen Variable kleiner ist als der Wert
M_GROESSER	...der Inhalt der angegebenen Variable größer ist als der Wert

n Tabelle 15 – Variablenvergleichsmerker bei VERG_VA

Operation

```
(M_GLEICH)    ç (Variable) = (Wert)
(M_KLEINER)   ç (Variable) < (Wert)
(M_GROESSER)  ç (Variable) > (Wert)
```

Beispiel

```
LAD_VA     V_ZEIGER,K_ANFANG_TAB   // Zeiger initialisieren
VERG_VA    V_ZEIGER,2000           // Vergleich
LAD_M      M_GLEICH                // Wenn der M_GLEICH ein ist, dann steht der
                                   // Wert 2000 in V_ZEIGER
SPRINGJ    FERTIG                  // Springe zum Label FERTIG
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Variablenvergleichsbefehle (Seite 3)

n VERG_VI

VERG_VI Variable V, Zeiger V

Gruppe	Abhängig von BES	Verändert BES
Variablenvergleich	**y Nein**	**n Nein**

VERG_VI vergleicht den Inhalt der angegebenen Variable mit dem Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:

Merker	Wird gesetzt wenn...
M_GLEICH	...die Inhalte der angegebenen Variable und der durch den Zeiger bestimmten Variable gleich sind
M_KLEINER	...der Inhalt der angegebenen Variable kleiner ist als der Inhalt der durch den Zeiger bestimmten Variable
M_GROESSER	...der Inhalt der angegebenen Variable größer ist als der Inhalt der durch den Zeiger bestimmten Variable

n Tabelle 16 – Variablenvergleichsmerker bei VERG_VI

Operation

```
(M_GLEICH)    ç (Variable) = (Zeiger à Variable)
(M_KLEINER)   ç (Variable) < (Zeiger à Variable)
(M_GROESSER)  ç (Variable) > (Zeiger à Variable)
```

Beispiel

```
// In diesem Beispiel haben wir ein Variablenfeld, welches z.B. vom PC geschrieben
// wird. Dieses Variablenfeld vergleichen wir mit einer Variable.

LAD_VA     V_ZEIGER1,TABELLE1      // Zeiger-Variable1 initialisieren
LAD_VA     V_ZEIGER2,TABELLE2      // Zeiger-Variable2 initialisieren
LAD_VA     V_VERGLEICH,1234        // die Variable V_VERGLEICH hat den Inhalt 1234
LOOP:
VERG_VI    V_VERGLEICH,V_ZEIGER1   // Vergleich durchführen
LAD_M      M_GLEICH                // Übereinstimmender Eintrag gefunden?
SPRINGJ    FERTIG                  // Wenn ja, Springe zum Label "Fertig"
INC_V      V_ZEIGER1,1             // Tabellenzeiger1 auf nächsten Eintrag
INC_V      V_ZEIGER2,1             // Tabellenzeiger2 auf nächsten Eintrag
SPRING     LOOP                    // Zurück zur Schleife
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Variablenvergleichsbefehle (Seite 3)

n VERG_VV

VERG_VV Variable1 V, Variable2 V

Gruppe	Abhängig von BES	Verändert BES
Variablenvergleich	**y Nein**	**n Nein**

VERG_VV vergleicht die Inhalte der beiden angegebenen Variablen. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:

Merker	Wird gesetzt wenn...
M_GLEICH	...die Inhalte beiden angegebenen Variablen gleich sind.
M_KLEINER	...der Inhalt der Variable1 kleiner ist als der Inhalt der Variable2
M_GROESSER	...der Inhalt der Variable1 groesser ist als der Inhalt der Variable2

n Tabelle 17 – Variablenvergleichsmerker bei VERG_VV

Operation

```
(M_GLEICH)   ç (Variable1) = (Variable2)
(M_KLEINER)  ç (Variable1) < (Variable2)
(M_GROESSER) ç (Variable1) > (Variable2)
```

Beispiel

```
VERG_VV    V_TEST1,V_TEST2        // Vergleicht die Variablen V_TEST1 und V_TEST2
LAD_M      M_KLEINER              // Vergleichsergebnis abfragen: ist V_TEST1
                                  // kleiner als V_TEST2?
SPRINGJ    Fehler                 // Darf nicht sein! Fehler ausgeben
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Variablenvergleichsbefehle (Seite 3)

n WART_A...WART_E

WART_A...WART_E

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	**- Ja**	**n Nein**

WART_A kann zur Programmierung einfacher Schleifen verwendet werden. Die Schleife wird so lange durchlaufen bis der Bitergebnisspeicher eingeschaltet ist.

Operation (bei Ausführung von WART_E)

(Programmadresse) **ç** (Adresse des letzten WARTE_A Befehls)

Beispiel

```
// Programmierung einer Schleife mit WART_A...WART_E

WART_A                          // Anfang der Schleife
...                             // Programmcode
WART_E                          // Wenn Bitergebnisspeicher ein, dann weiter mit
                                // nächster Zeile
                                // wenn Bitergebnisspeicher aus, dann zurück zu
                                // der Zeile, in der WART_A steht

// Obiges Beispiel einer Schleife mit WART_A...WARTE_E entspricht genau
// folgendem Beispiel einer "normalen" Schleife

LOOP:                           // Label definieren
...                             // Programmcode
SPRINGN LOOP                    // Wenn Bitergebnisspeicher ein, dann weiter mit
                                // nächster Zeile
                                // wenn Bitergebnisspeicher aus, dann zurück zum
                                // Label LOOP
```

Hinweise

n Der Programmcode zwischen WART_A und WART_E wird so lange ausgeführt, bis der Bitergebnisspeicher bei Ausführung des Befehls WART_E eingeschaltet ist.

Siehe auch

Programmablaufbefehle (Seite 3)

n XODER_A

XODER_A Ausgang A

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

XODER_A verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Ausgangs mit einem logischen Exclusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.

Operation

(Bitergebnis) **ç** (Bitergebnis) ^ (Ausgang)

Beispiel

```
LAD_A      A_TEST1                // Wenn A_TEST1 eingeschaltet
XODER_A    A_TEST2                // oder E_TEST2 eingeschaltet ist,
SPRINGJ    TESTZYKLUS             // dann springe zu Label TESTZYKLUS
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n XODER_E

XODER_E Eingang E

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	**y Nein**	**þ Ja**

XODER_E verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Eingangs mit einem logischen Exclusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.

Operation

(Bitergebnis) **ç** (Bitergebnis) ^ (Eingang)

Beispiel

```
LAD_E      E_TEST1                // Wenn E_TEST1 eingeschaltet
XODER_E    E_TEST2                // oder E_TEST2 eingeschaltet ist,
SPRINGJ    TESTZYKLUS             // dann springe zu Label TESTZYKLUS
```

Hinweise

- Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Ein-/Ausgangsbefehle (Seite 3)

n XODER_M

XODER_M Merker M

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	**y Nein**	**þ Ja**

XODER_M verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Merkers mit einem logischen Exclusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.

Operation

```
(Bitergebnis) ç (Bitergebnis) ^ (Merker)
```

Beispiel

```
LAD_M      M_TEST1                // Wenn M_TEST1
XODER_M    M_TEST2                // oder M_TEST2 eingeschaltet sind,
SPRINGJ    TESTZYKLUS             // dann springe zu Label TESTZYKLUS
```

Hinweise

n Dieser Befehl ist nicht abhängig vom Bitergebnisspeicher und wird immer ausgeführt.

Siehe auch

Merkerbefehle (Seite 3)
Fehler! Verweisquelle konnte nicht gefunden werden. (Seite **Fehler! Textmarke nicht definiert.**)

n #ELSE

#ELSE

Gruppe	Abhängig von BES	Verändert BES
Compilierung	**y Nein**	**n Nein**

#ELSE wird für Abfragen im Zusammenhang mit der bedingten Compilierung verwendet. Vor dem Befehl wird eine Bedingung mit dem Befehl EQ abgefragt. Ist die erste abhängige Anweisung nicht erfüllt, wird die zweite abhängige Anweisung durchgeführt.

Beispiel

```
DEF_W      3,Achsenanzahl         // Achsenanzahl ist gleich 3

#IF Achsenanzahl EQ 2             // Achsenanzahl ist in diesem Beispiel 2, also
...                               // wird der Quelltext nach diesem #IF bis zum
...                               // nächsten #ELSE nicht verwendet.
#ELSE                             // Weil die Bedingung beim #IF nicht erfüllt war,
...                               // wird der Quelltext unter #ELSE ausgeführt
...
#ENDIF                            // #ENDIF markiert das Ende dieser Abfrage. Ab
...                               // jetzt wird wieder der Quelltext unabhängig
...                               // von der Abfrage weiter verwendet.
```

Siehe auch

Compilieranweisungen (Seite 3)

n #ENDIF

#ENDIF

Gruppe	Abhängig von BES	Verändert BES
Compilierung	**y Nein**	**n Nein**

#ENDIF beendet eine bedingte Compilierung.

Beispiel

```
#IF Achsenanzahl EQ 2            // Wenn Achsenanzahl 2 ist, dann Code verwenden
...
...
#ENDIF                           // Ende bedingter Compilierung
```

Siehe auch

Compilieranweisungen (Seite 3)

n #IF

Gruppe	Abhängig von BES	Verändert BES
Compilierung	**y Nein**	**n Nein**

#IF Abfragebedingung K
#IF Wert1 K **EQ Wert2** K

#IF wird für Abfragen im Zusammenhang mit der bedingten Compilierung verwendet. Vor dem Befehl wird eine Bedingung mit dem Befehl EQ abgefragt. Wenn die erste abhängige Bedingung erfüllt ist, wird die erste Anweisung durchgeführt, ist die erste abhängige Anweisung nicht erfüllt, wird die zweite abhängige Anweisung (wenn eine vorhanden ist) durchgeführt.

Verfügbare Verknüpfungsbedingungen

Für die Verknüpfung der Definitionsbefehle stehen Ihnen folgende Abfragebedingungen zur Verfügung:

Befehl	Bedeutung
EQ	"Equal": Wenn die beiden Parameter in der Abfrage gleich sind, wird die bedingte Compilierung durchgeführt.
NEQ	"Not Equal": Wenn die beiden Parameter in der Abfrage ungleich sind, wird die bedingte Compilierung durchgeführt.
GT	"Greater than": Wenn der erste Parameter grösser ist als der zweite Parameter, wird die bedingte Compilierung durchgeführt.
LT	"Less than": Wenn der erste Parameter kleiner ist als der zweite Parameter, wird die bedingte Compilierung durchgeführt.

n Tabelle 18 – Verknüpfungen für die bedingte Compilierung

Beispiel

```
DEF_W      2,Achsenanzahl          // Achsenanzahl ist gleich 2

#IF Achsenanzahl EQ 2              // Achsenanzahl ist in diesem Beispiel 2, also
...                                // wird der Quelltext nach diesem #IF bis zum
...                                // nächsten #ENDIF oder #ELSE verwendet.
#ELSE                              // Weil die Bedingung beim #IF erfüllt war, wird
...                                // der Quelltext unter #ELSE nicht ausgeführt
...
#ENDIF                             // #ENDIF markiert das Ende dieser Abfrage. Ab
...                                // jetzt wird wieder der Quelltext unabhängig
...                                // von der Abfrage weiter verwendet.

// #IF...#ELSE...#ENDIF Abfragen können auch verschachtelt werden, so wie hier:

#IF Achsenanzahl EQ 2              // Wie oben beschrieben, ist Achsenanzahl 2, also
...                                // wird der Quelltext verwendet
#IF Greifer EQ 1                   // Wenn ein System mit Greifer verwendet wird,
...                                // dann diesen Quelltext verwenden
...
#ELSE                              // Wenn die vorherige Abfrage nach der Konstante
...                                // Greifer nicht erfüllt wurde, dann wird nun der
...                                // folgende Quelltext verwendet
#ENDIF                             // Ende der Abfrage nach Greifer
#ENDIF                             // Ende der Abfrage nach Achsenanzahl
...                                // jetzt wird wieder der Quelltext unabhängig
...                                // von der Abfrage weiter verwendet.
```

Siehe auch

Compilieranweisungen (Seite 3)

n #INCLUDE

#INCLUDE Quellcodedateiname K

Gruppe	Abhängig von BES	Verändert BES
Compilierung	**y Nein**	**n Nein**

#INCLUDE fügt eine weitere Quellcodedatei an der aktuellen Stelle ein.

Beispiel

```
#INCLUDE Achsen.mc                 // Die Datei Achsen.mc wird mit eingebunden
```

Siehe auch

Compilieranweisungen (Seite 3)

n Raum für Ihre Notizen

n Raum für Ihre Notizen

Kapitel 4 Achsprogrammierung

Im MC100 System erfolgt die Ansteuerung der Achsen vollständig über Systemvariablen- und Merker. Dies bedeutet:

- **n** Für jede Achse ist der Funktionsumfang durch bestimmte Systemvariablen- und Merker festgelegt.
- **n** Duch Beschreiben dieser Variablen und Merker wird eine Funktion vorbereitet.
- **n** Nachdem alle für diese Funktion notwendigen Daten in die Variablen und Merker geschrieben wurden, kann die Funktion durch setzen des Bestätigungsmerkers (M_SEND_xxx) ausgelöst werden.
- **n** Der Zustand der Achsen wird innerhalb des MC100 Systems wiederum über Systemvariablen- und Merker zurückgespiegelt.

Innerhalb eines SPS-Programms könnte das dann z.B. so aussehen:

```
LAD_VA     V_RAMPE_SMA1,80         // Rampe programmieren: 80 ms/KHz
LAD_VA     V_FMIN_SMA1,500         // Start-Stopp-Frequenz programmieren: 500 Hz
LAD_VA     V_FMAX_SMA1,5000        // Lauffrequenz programmieren: 5000 Hz
LAD_VA     V_FAKTOR_SMA1,256       // Keine Maßeinheit programmieren

LAD_VA     V_INDEX_SMA1,10000      // Fahrstrecke mit 10000 Schritten laden
EIN_M      M_START_SMA1            // Startmerker: Achse 1 soll gestartet werden
EIN_M      M_SEND_SMA1             // Programmierte Funktion auslösen: Starte
                                   // Positionierung der Achse 1 um 10000 Schritte
                                   // mit den oben angegebenen Geschwindigkeiten

// Bevor weitere Aktionen im Zusammenhang mit einer der Achsen ausgelöst werden
// oder der Status der Achse abgefragt werden kann, muss im SPS-Programm
// sichergestellt werden, dass der Fahrauftrag an die Achse übertragen wurde.
// Deshalb warten wir hier darauf, dass der Merker M_SEND_SMA1 zurückgesetzt
// wird – dies bedeutet dann: alle Daten sind übertragen, die Achse startet.

LOOP1:
LAD_M      M_SEND_SMA1             // Sendemerker aus = alle Daten übertragen?
SPRINGJ    LOOP1                   // Nein, wir warten noch ein wenig

// Jetzt läuft die Achse 1 und wir warten einfach, bis der In-Positionsmerker
// kommt, sprich: die Achse hat ihre Zielposition erreicht.

LOOP2:
LAD_M      M_FERTIG_SMA1           // Merker aus bedeutet: die Achse fährt noch
SPRINGN    LOOP2                   // und wir warten noch ein wenig
```

Weitere Beispiele zur Achsprogrammierung finden Sie in

n Systemvariablen- und Merker Schrittmotorachse 1

Merker	Symbol	Funktion
641	M_SEND_SMA1	Sendetelegramm wird an Achse 1 gesendet
865	M_ENDSPL_SMA1	Status Endschalter +
866	M_ENDSMI_SMA1	Status Endschalter -
867	M_UEBERW_SMA1	Status Eingang für Schrittüberwachung
868	M_MOPSW_SMA1	Status für den optionalen Referenzschalter
869	M_RGLBER_SMA1	Status für Reglerbereitschaft der Achse
870	M_FEHLER_SMA1	Merker wird ein, wenn bei der Schrittüberwachung ein Fehler aufgetreten ist.
871	M_POSOK_SMA1	Merker ein => Achse in Position Merker aus => Achse nicht in Position
872	M_FERTIG_SMA1	Merker ein => Achse steht Merker aus => Achse fährt
2493	M_START_SMA1	Startmerker für Achse 1, wird vom System zurückgesetzt
2494	M_HALT_SMA1	Achse stoppt mit programmierter Bremsrampe
2495	M_STOP_SMA1	Achse stoppt sofort, ohne Bremsrampe
2496	M_GETP_SMA1	Lesen der aktuellen Position der Achse 1
2500	M_INIT_SMA1	Initialisierungsmerker der Achse 1
2501	M_LAUFSCH_SMA1	Merker veranlasst die Achse eine Fahrt auf den Referenzschalter (Endschalter +) auszuführen
2502	M_ENDSCH_SMA1P	Endschalter – wird Referenzschalter
2503	M_OEFFNER_SMA1	Endschalter sind Öffner
2506	M_LAUFFMIN_SMA1	Achse fährt mit Start-Stopp-Frequenz
2507	M_HALTSCH_SMA1	Stoppe Achse bei erreichen des Endschalter +
2508	M_MINUS_SMA1	Drehrichtung der Achse für eine Fahrbewegung umdrehen
171	V_ABPOS_SMA1	Absolute Position der Achse (=Istposition in Maßeinheit)
172	V_INDEX_SMA1	Verfahrstrecke der Achse in Maßeinheit Positiver Wert = Drehrichtung + Negativer Wert = Drehrichtung -
173	V_FMAX_SMA1	Lauffrequenz der Achse (Hz) 100 – 40000 Hz
316	V_ABSCHR_SMA1	Istposition der Achse in Schritten
363	V_FAKTOR_SMA1	Umrechnungsfaktor der Achse
364	V_RAMPE_SMA1	Beschleunigungs- und Bremsrampe der Achse (ms/KHz) 4 – 256 ms/KHz
365	V_FMIN_SMA1	Start-Stopp-Frequenz der Achse 10 – 1000 Hz
366	V_FEHLSCHR_SMA1	Überwachungsschritte von Impuls zu Impuls des Überwachungssystems, z.B. Drehgeber 0 – 256 Schritte 0 = Schrittüberwachung ist ausgeschalter

n Systemvariablen- und Merker Schrittmotorachse 2

Merker	Symbol	Funktion
642	M_SEND_SMA2	Sendetelegramm wird an Achse 2 gesendet
873	M_ENDSPL_SMA2	Status Endschalter +
874	M_ENDSMI_SMA2	Status Endschalter -
875	M_UEBERW_SMA2	Status Eingang für Schrittüberwachung
876	M_MOPSW_SMA2	Status für den optionalen Referenzschalter
877	M_RGLBER_SMA2	Status für Reglerbereitschaft der Achse
878	M_FEHLER_SMA2	Merker wird ein, wenn bei der Schrittüberwachung ein Fehler aufgetreten ist.
879	M_POSOK_SMA2	Merker ein => Achse in Position Merker aus => Achse nicht in Position
880	M_FERTIG_SMA2	Merker ein => Achse steht Merker aus => Achse fährt
2509	M_START_SMA2	Startmerker für Achse 2, wird vom System zurückgesetzt
2510	M_HALT_SMA2	Achse stoppt mit programmierter Bremsrampe
2511	M_STOP_SMA2	Achse stoppt sofort, ohne Bremsrampe
2512	M_GETP_SMA2	Lesen der aktuellen Position der Achse 2
2516	M_INIT_SMA2	Initialisierungsmerker der Achse 2
2517	M_LAUFSCH_SMA2	Merker veranlasst die Achse eine Fahrt auf den Referenzschalter (Endschalter +) auszuführen
2518	M_ENDSCH_SMA2P	Endschalter – wird Referenzschalter
2519	M_OEFFNER_SMA2	Endschalter sind Öffner
2522	M_LAUFFMIN_SMA2	Achse fährt mit Start-Stopp-Frequenz
2523	M_HALTSCH_SMA2	Stoppe Achse bei erreichen des Endschalter +
2524	M_MINUS_SMA2	Drehrichtung der Achse für eine Fahrbewegung umdrehen
174	V_ABPOS_SMA2	Absolute Position der Achse (=Istposition in Maßeinheit)
175	V_INDEX_SMA2	Verfahrstrecke der Achse in Maßeinheit Positiver Wert = Drehrichtung + Negativer Wert = Drehrichtung -
176	V_FMAX_SMA2	Lauffrequenz der Achse (Hz) 100 – 40000 Hz
317	V_ABSCHR_SMA2	Istposition der Achse in Schritten
367	V_FAKTOR_SMA2	Umrechnungsfaktor der Achse
368	V_RAMPE_SMA2	Beschleunigungs- und Bremsrampe der Achse (ms/KHz) 4 – 256 ms/KHz
369	V_FMIN_SMA2	Start-Stopp-Frequenz der Achse 10 – 1000 Hz
370	V_FEHLSCHR_SMA2	Überwachungsschritte von Impuls zu Impuls des Überwachungssystems, z.B. Drehgeber 0 – 256 Schritte 0 = Schrittüberwachung ist ausgeschalter

n Systemvariablen- und Merker Schrittmotorachse 3

Merker	Symbol	Funktion
644	M_SEND_SMA3	Sendetelegramm wird an Achse 3 gesendet
881	M_ENDSPL_SMA3	Status Endschalter +
882	M_ENDSMI_SMA3	Status Endschalter -
883	M_UEBERW_SMA3	Status Eingang für Schrittüberwachung
884	M_MOPSW_SMA3	Status für den optionalen Referenzschalter
885	M_RGLBER_SMA3	Status für Reglerbereitschaft der Achse
886	M_FEHLER_SMA3	Merker wird ein, wenn bei der Schrittüberwachung ein Fehler aufgetreten ist.
887	M_POSOK_SMA3	Merker ein => Achse in Position Merker aus => Achse nicht in Position
888	M_FERTIG_SMA3	Merker ein => Achse steht Merker aus => Achse fährt
2525	M_START_SMA3	Startmerker für Achse 3, wird vom System zurückgesetzt
2526	M_HALT_SMA3	Achse stoppt mit programmierter Bremsrampe
2527	M_STOP_SMA3	Achse stoppt sofort, ohne Bremsrampe
2528	M_GETP_SMA3	Lesen der aktuellen Position der Achse 3
2532	M_INIT_SMA3	Initialisierungsmerker der Achse 3
2533	M_LAUFSCH_SMA3	Merker veranlasst die Achse eine Fahrt auf den Referenzschalter (Endschalter +) auszuführen
2534	M_ENDSCH_SMA3P	Endschalter – wird Referenzschalter
2535	M_OEFFNER_SMA3	Endschalter sind Öffner
2538	M_LAUFFMIN_SMA3	Achse fährt mit Start-Stopp-Frequenz
2539	M_HALTSCH_SMA3	Stoppe Achse bei erreichen des Endschalter +
2540	M_MINUS_SMA3	Drehrichtung der Achse für eine Fahrbewegung umdrehen
177	V_ABPOS_SMA3	Absolute Position der Achse (=Istposition in Maßeinheit)
178	V_INDEX_SMA3	Verfahrstrecke der Achse in Maßeinheit Positiver Wert = Drehrichtung + Negativer Wert = Drehrichtung -
179	V_FMAX_SMA3	Lauffrequenz der Achse (Hz) 100 – 40000 Hz
318	V_ABSCHR_SMA3	Istposition der Achse in Schritten
371	V_FAKTOR_SMA3	Umrechnungsfaktor der Achse
372	V_RAMPE_SMA3	Beschleunigungs- und Bremsrampe der Achse (ms/KHz) 4 – 256 ms/KHz
373	V_FMIN_SMA3	Start-Stopp-Frequenz der Achse 10 – 1000 Hz
374	V_FEHLSCHR_SMA3	Überwachungsschritte von Impuls zu Impuls des Überwachungssystems, z.B. Drehgeber 0 – 256 Schritte 0 = Schrittüberwachung ist ausgeschalter

n Systemvariablen- und Merker Schrittmotorachse 4

Merker	Symbol	Funktion
644	M_SEND_SMA4	Sendetelegramm wird an Achse 4 gesendet
889	M_ENDSPL_SMA4	Status Endschalter +
890	M_ENDSMI_SMA4	Status Endschalter -
891	M_UEBERW_SMA4	Status Eingang für Schrittüberwachung
892	M_MOPSW_SMA4	Status für den optionalen Referenzschalter
893	M_RGLBER_SMA4	Status für Reglerbereitschaft der Achse
894	M_FEHLER_SMA4	Merker wird ein, wenn bei der Schrittüberwachung ein Fehler aufgetreten ist.
895	M_POSOK_SMA4	Merker ein => Achse in Position Merker aus => Achse nicht in Position
896	M_FERTIG_SMA4	Merker ein => Achse steht Merker aus => Achse fährt
2541	M_START_SMA4	Startmerker für Achse 4, wird vom System zurückgesetzt
2542	M_HALT_SMA4	Achse stoppt mit programmierter Bremsrampe
2543	M_STOP_SMA4	Achse stoppt sofort, ohne Bremsrampe
2544	M_GETP_SMA4	Lesen der aktuellen Position der Achse 4
2548	M_INIT_SMA4	Initialisierungsmerker der Achse 4
2549	M_LAUFSCH_SMA4	Merker veranlasst die Achse eine Fahrt auf den Referenzschalter (Endschalter +) auszuführen
2550	M_ENDSCH_SMA4P	Endschalter – wird Referenzschalter
2551	M_OEFFNER_SMA4	Endschalter sind Öffner
2554	M_LAUFFMIN_SMA4	Achse fährt mit Start-Stopp-Frequenz
2555	M_HALTSCH_SMA4	Stoppe Achse bei erreichen des Endschalter +
2556	M_MINUS_SMA4	Drehrichtung der Achse für eine Fahrbewegung umdrehen
180	V_ABPOS_SMA4	Absolute Position der Achse (=Istposition in Maßeinheit)
181	V_INDEX_SMA4	Verfahrstrecke der Achse in Maßeinheit Positiver Wert = Drehrichtung + Negativer Wert = Drehrichtung -
182	V_FMAX_SMA4	Lauffrequenz der Achse (Hz) 100 – 40000 Hz
319	V_ABSCHR_SMA4	Istposition der Achse in Schritten
375	V_FAKTOR_SMA4	Umrechnungsfaktor der Achse
376	V_RAMPE_SMA4	Beschleunigungs- und Bremsrampe der Achse (ms/KHz) 4 – 256 ms/KHz
377	V_FMIN_SMA4	Start-Stopp-Frequenz der Achse 10 – 1000 Hz
378	V_FEHLSCHR_SMA4	Überwachungsschritte von Impuls zu Impuls des Überwachungssystems, z.B. Drehgeber 0 – 256 Schritte 0 = Schrittüberwachung ist ausgeschalter

4.1 Programmbeispiele

n Relative Positionierung einer Schrittmotorachse

```
LAD_VA     V_RAMPE_SMA1,80         // Rampe = 80 ms/KHz
LAD_VA     V_FMIN_SMA1,500         // Start-Stopp-Frequenz = 500 Hz
LAD_VA     V_FMAX_SMA1,5000        // Lauffrequenz = 5000 Hz
LAD_VA     V_FAKTOR_SMA1,256       // Keine Maßeinheit

LAD_VA     V_INDEX_SMA1,10000      // Fahrstrecke mit 10000 Schritten laden
EIN_M      M_START_SMA1            // Startmerker Achse 1 setzen
EIN_M      M_SEND_SMA1             // Sendetelegramm an Achse 1 sende, Achse 1 Start
LOOP1:
LAD_M      M_SEND_SMA1             // Sendemerker aus?
SPRINGJ    LOOP1
LOOP2:
LAD_M      M_FERTIG_SMA1           // Merker aus = Achse fährt!
SPRINGN    LOOP2
```

n Absolute Positionierung einer Schrittmotorachse

```
        LAD_VAV_RAMPE_SMA1,50           ;Rampe = 50 ms/KHz
        LAD_VAV_FMIN_SMA1,500                   ;Start-Stopp-Frequenz = 500 Hz
        LAD_VA      V_FMAX_SMA1,5000            ;Lauffrequenz = 5000 Hz
        LAD_VAV_FAKTOR_SMA1,5120        ;Istposition und Fahrstrecke in Maßeinheit
                                                ;Spindelsteigung = 5mm
                                                ;Schritte pro Umdrehung des Motors
= 1000
                                                ;Auflösung = 1/10 mm
        SUB_VV      V_SOLLPOS_A1,V_ABPOS_SMA1 ;Subtrahiere die Sollposition mit der
Istposition
        LAD_VV      V_INDEX_SMA1,VARERG         ;Ergebnis in Fahrstrecke laden
        EIN_M       M_START_SMA1                ;Startmerker Achse 1 setzen
        EIN_M       M_SEND_SMA1             ;Sendetelegramm an Achse 1 sende, Achse
                                                ;1 startet
LOOP1:
        LAD_M  M_SEND_SMA1              ;Sendemerker aus?
        SPRINGJ     LOOP1
LOOP2:
        LAD_M  M_FERTIG_SMA1                    ;Merker aus = Achse fährt!
        SPRINGN     LOOP2

        usw.
```

n Referenzfahrt einer Schrittmotorachse

```
        LAD_VAV_RAMPE_SMA1,50          ;Rampe = 50 ms/KHz
        LAD_VAV_FMIN_SMA1,500                  ;Start-Stopp-Frequenz = 500 Hz
        LAD_VA      V_FMAX_SMA1,5000           ;Lauffrequenz = 5000 Hz
        LAD_VAV_FAKTOR_SMA1,5120       ;Istposition und Fahrstrecke in Maßeinheit
                                                ;Spindelsteigung = 5mm
                                                ;Schritte pro Umdrehung des Motors
= 1000
                                                ;Auflösung = 1/10 mm

//Fahrt auf Endschalter+
        LAD_VA      V_INDEX_SMA1,10            ;Überlaufstrecke = 1 mm
        EIN_M       M_LAUFSCH_SMA1                 ;Referenzfahrt auf Endschalter +
        AUS_M M_OEFFNER_SMA1               ;Endschalter = Schließer
        EIN_M       M_START_SMA1                   ;Startmerker Achse 1 setzen
        EIN_M       M_SEND_SMA1            ;Sendetelegramm an Achse 1 sende, Achse
                                                        ;1 startet
REF1:
        LAD_M M_SEND_SMA1              ;Sendemerker aus?
        SPRINGJ     REF1
REF2:
        LAD_M M_FERTIG_SMA1                    ;Merker aus = Achse fährt!
        SPRINGN     REF2

//Endschalter + freifahren
        LAD_VA      V_INDEX_SMA1,10            ;Überlaufstrecke = 1 mm
        EIN_M       M_LAUFSCH_SMA1                 ;Referenzfahrt auf Endschalter +
        EIN_M       M_OEFFNER_SMA1                 ;Endschalter = Öffner
        EIN_M       M_MINUS_SMA1                   ;Drehrichtung für Fahrt ändern
      EIN_M         M_START_SMA1                   ;Startmerker Achse 1 setzen
        EIN_M       M_SEND_SMA1            ;Sendetelegramm an Achse 1 sende, Achse
                                                        ;1 startet
REF3:
        LAD_M M_SEND_SMA1              ;Sendemerker aus?
        SPRINGJ     REF3
REF4:
        LAD_M M_FERTIG_SMA1                    ;Merker aus = Achse fährt!
        SPRINGN     REF4

        usw.
```


0 Umrechnungsfaktor

$$\text{Umrechnungsfaktor} = \frac{\text{Schritte pro Umdrehung des Motors}}{\text{Weg pro Umdrehung des Motors} * \text{Auflösung}} * 256$$

Beispiel:

Spindelsteigung = 5 mm

Schritt / Umdrehung = 400 Schritte

Auflösung = 1/10 mm => 10

1/100 mm => 100

$$\text{Umrechnungsfaktor} = \frac{400}{5\ \text{mm} * 10} * 256 = 2048$$

Kapitel 5 Speicherbelegung

5.1 Merker

Die Steuerung verfügt über 3548 Merker, diese frei verfügbaren Merker sind spannungsausfallsicher gepuffert. Die vom System reservierten Merker dürfen nur für die vom System zugewiesenen Funktionen verwendet werden.

Merkerbereich	*Inhalt*
1-499	frei verfügbar, batteriegepuffert
500	Bitergebnisspeicher
501 - 507	Bitergebnisspeicherschieberegister
508	Variablenvergleichsergebnis gleich
509	Variablenvergleichsergebnis kleiner
510	Merker immer ein
511 - 542	Timer 1 bis 32
543	Reserviert für Systemerweiterung
550	Steuerungsreset
551 - 560	Reserviert für Systemerweiterung
561	Variableneingabe aktiv
562	Lampentest
563	Maschinentaste gedrückt
564	Variablen und Texte anzeigen
565	Anzeige löschen bei Eingabe
566	Variable aus Anzeige übernehmen
567 – 569	Reserviert für Systemerweiterungen
570	Stop Taskwechsel
571	Kommunikationsstörung
572 - 579	Reserviert für Systemerweiterungen
580	Freigabe elektrisches Handrad
581 - 589	Reserviert für Systemerweiterungen
590	Zwei NC-Programme gleichzeitig
591	Fehler NC-Programm 1
592	Fehler NC-Programm 2
593	Ende NC-Programm 1
594	Ende NC-Programm 2
595	Merkerfunktion des NC-Prgrammes unterdrücken
596 – 600	Reserviert für Systemerweiterungen
601 – 608	Sendemerker Servoachsen 1 – 8
609 – 616	Initialisierung Umrechnung Servoachsen 1 – 8
617 – 624	Initialisierung PID-Regler Servoachsen 1 – 8
625 – 632	Sendemerker Temperaturregler MC100-TR (Modul 1 – 8)
633 – 640	Initialisierung Temperaturregler MC100-TR (Modul 1 – 8)
641 – 648	Sendemerker Schrittmotorachsen 1 – 8
649 – 656	Sendemerker der analogen Ein- Ausgangskarten (Modul 1)
657 – 664	Sendemerker der analogen Ein- Ausgangskarten (Modul 2)
665 – 800	Reserviert für Systemerweiterungen
801 – 808	Statusmerker der Servoachse 1
809 – 816	Statusmerker der Servoachse 2
817 – 824	Statusmerker der Servoachse 3
225 – 832	Statusmerker der Servoachse 4
833 – 840	Statusmerker der Servoachse 5
841 – 848	Statusmerker der Servoachse 6
849 – 856	Statusmerker der Servoachse 7
857 – 864	Statusmerker der Servoachse 8
865 – 872	Statusmerker der Schrittmotorachse 1
873 – 880	Statusmerker der Schrittmotorachse 2
881 – 888	Statusmerker der Schrittmotorachse 3
889 – 896	Statusmerker der Schrittmotorachse 4
897 – 904	Statusmerker der Schrittmotorachse 5

905 – 912	Statusmerker der Schrittmotorachse 6
913 – 920	Statusmerker der Schrittmotorachse 7
921 – 928	Statusmerker der Schrittmotorachse 8
929 – 936	Fehlermerker MC100-CA (Modul 1)
937 – 944	Fehlermerker Mc100-CA (Modul 2)
953 – 1999	Reserviert für Systemerweiterungen
2000 – 2099	G-Funktionen für NC-Programm
2100 – 2199	M-Funktionen für NC-Programm
2200 – 2225	Funktionsmerker A-Z für NC-Programm
2226 – 2300	Reserviert für Systemerweiterung
2301 – 2324	Merker für Servoachse 1
2325 – 2348	Merker für Servoachse 2
2349 – 2372	Merker für Servoachse 3
2373 – 2396	Merker für Servoachse 4
2397 – 2420	Merker für Servoachse 5
2421 – 2444	Merker für Servoachse 6
2445 – 2468	Merker für Servoachse 7
2469 – 2492	Merker für Servoachse 8
2493 – 2508	Merker für Schrittmotorachse 1
2509 – 2524	Merker für Schrittmotorachse 2
2525 – 2540	Merker für Schrittmotorachse 3
2541 – 2556	Merker für Schrittmotorachse 4
2557 – 2572	Merker für Schrittmotorachse 5
2573 – 2588	Merker für Schrittmotorachse 6
2589 – 2604	Merker für Schrittmotorachse 7
2605 – 2620	Merker für Schrittmotorachse 8
2621 – 2628	Merker für MC100-CA (Modul 1)
2629 – 2636	Merker für MC100-CA (Modul 2)
2637 – 2652	Reserviert für Systemerweiterungen
2653 – 2700	Merker für Tastatur 6 x 8
2701 – 2732	Merker für Tastatur 4 x 8 (nur MC100-PF 4 x 6)
2733 – 2780	Merker für Tastatur 6 x 8 MC100-BED (Modul 1)
2781 – 2828	Merker für Tastatur 6 x 8 MC100-BED (Modul 2)
2829 – 2844	Statusmerker für Temperaturregler MC100-TR (Modul 1)
2845 – 2860	Statusmerker für Temperaturregler MC100-TR (Modul 2)
2861 – 2876	Statusmerker für Temperaturregler MC100-TR (Modul 3)
2877 – 2892	Statusmerker für Temperaturregler MC100-TR (Modul 4)
2893 – 2908	Statusmerker für Temperaturregler MC100-TR (Modul 5)
2909 – 2924	Statusmerker für Temperaturregler MC100-TR (Modul 6)
2925 – 2940	Sendemerker für Temperaturregler MC100-TR (Modul 1)
2941 – 2956	Sendemerker für Temperaturregler MC100-TR (Modul 2)
2957 – 2972	Sendemerker für Temperaturregler MC100-TR (Modul 3)
2973 – 2988	Sendemerker für Temperaturregler MC100-TR (Modul 4)
2989 – 3004	Sendemerker für Temperaturregler MC100-TR (Modul 5)
3005 – 3020	Sendemerker für Temperaturregler MC100-TR (Modul 6)
3021 – 3036	Sendemerker für Temperaturregler MC100-TR (Modul 7)
3037 – 3052	Sendemerker für Temperaturregler MC100-TR (Modul 8)
3053 – 3068	Statusmerker für Temperaturregler MC100-TR (Modul 7)
3067 – 3084	Statusmerker für Temperaturregler MC100-TR (Modul 8)
3085 – 3132	Merker für Leuchtdioden
3132 – 3199	Reserviert für Systemerweiterungen
3200 – 3548	Frei verfügbar

5.2 Variablen

Die Steuerung verfügt über bis zu 16000 spannungsausfallsicher gepufferte Variable. Die vom System reservierten Variablen dürfen nur für die vom System zugewiesenen Funktionen verwendet werden.

Variablenbereich	*Inhalt*
1	Variablenergebnisspeicher für arithmetische und logische Variablenbefehle
2 – 135	Frei verfügbar
136 - 146	Puffer für Initialisierung der Servoachsen
147	Istposition der Servoachse 1
148	Sollposition der Servoachse 1
149	Sollgeschwindigkeit der Servoachse 1
150	Istposition der Servoachse 2
151	Sollposition der Servoachse 2
152	Sollgeschwindigkeit der Servoachse 2
153	Istposition der Servoachse 3
154	Sollposition der Servoachse 3
155	Sollgeschwindigkeit der Servoachse 3
156	Istposition der Servoachse 4
157	Sollposition der Servoachse 4
158	Sollgeschwindigkeit der Servoachse 4
159	Istposition der Servoachse 5
160	Sollposition der Servoachse 5
161	Sollgeschwindigkeit der Servoachse 5
162	Istposition der Servoachse 6
163	Sollposition der Servoachse 6
164	Sollgeschwindigkeit der Servoachse 6
165	Istposition der Servoachse 7
166	Sollposition der Servoachse 7
167	Sollgeschwindigkeit der Servoachse 7
168	Istposition der Servoachse 8
169	Sollposition der Servoachse 8
170	Sollgeschwindigkeit der Servoachse 8
171	Istposition der Schrittmotorachse 1
172	Indexstrecke der Schrittmotorachse 1
173	Lauffrequenz der Schrittmotorachse 1
174	Istposition der Schrittmotorachse 2
175	Indexstrecke der Schrittmotorachse 2
176	Lauffrequenz der Schrittmotorachse 2
177	Istposition der Schrittmotorachse 3
178	Indexstrecke der Schrittmotorachse 3
179	Lauffrequenz der Schrittmotorachse 3
180	Istposition der Schrittmotorachse 4
181	Indexstrecke der Schrittmotorachse 4
182	Lauffrequenz der Schrittmotorachse 4
183	Istposition der Schrittmotorachse 5
184	Indexstrecke der Schrittmotorachse 5
185	Lauffrequenz der Schrittmotorachse 5
186	Istposition der Schrittmotorachse 6
187	Indexstrecke der Schrittmotorachse 6
188	Lauffrequenz der Schrittmotorachse 6
189	Istposition der Schrittmotorachse 7
190	Indexstrecke der Schrittmotorachse 7
191	Lauffrequenz der Schrittmotorachse 7
192	Istposition der Schrittmotorachse 8
193	Indexstrecke der Schrittmotorachse 8
194	Lauffrequenz der Schrittmotorachse 8

195	Istposition des schnellen Zählers 1 MC100-CA (Modul 1)
196	Istposition des schnellen Zählers 2 MC100-CA (Modul 1)
197	Istposition des schnellen Zählers 3 MC100-CA (Modul 1)
198	Istposition des schnellen Zählers 4 MC100-CA (Modul 2)
199	Istposition des schnellen Zählers 5 MC100-CA (Modul 2)
200	Istposition des schnellen Zählers 6 MC100-CA (Modul 2)
201 – 232	Timer 1 - 32
233	Divisionsrest
234	Multiplikationsüberlauf
235	Zähler elektrisches Handrad
236	Zähler elektrisches Handrad MC100-BED (Modul 1)
237	Zähler elektrisches Handrad MC100-BED (Modul 2)
238	Variableneingabe: Variablennummer
239	Anzeige: Nummer Text/ Variable
240	Variableneingabe: Formatbeschreibung
241	Anzeige: Formatbeschreibung
242 – 254	Reserviert für Systemerweiterungen
255	Zähler fehlerhafter Datenübertragung
256	Reserviert für Systemerweiterungen
257	Typ Modul 0 (SPS-CPU)
258 – 289	Typ Module 1 – 32
290 – 299	Reserviert für Systemerweiterungen
300	Istgeschwindigkeit Servoachse 1
301	Istschleppabstand Servoachse 1
302	Istgeschwindigkeit Servoachse 2
303	Istschleppabstand Servoachse 2
304	Istgeschwindigkeit Servoachse 3
305	Istschleppabstand Servoachse 3
306	Istgeschwindigkeit Servoachse 4
307	Istschleppabstand Servoachse 4
308	Istgeschwindigkeit Servoachse 5
309	Istschleppabstand Servoachse 5
310	Istgeschwindigkeit Servoachse 6
311	Istschleppabstand Servoachse 6
312	Istgeschwindigkeit Servoachse 7
313	Istschleppabstand Servoachse 7
314	Istgeschwindigkeit Servoachse 8
315	Istschleppabstand Servoachse 8
316	Istposition in Schritten der Schrittmotorachse 1
317	Istposition in Schritten der Schrittmotorachse 2
318	Istposition in Schritten der Schrittmotorachse 3
319	Istposition in Schritten der Schrittmotorachse 4
320	Istposition in Schritten der Schrittmotorachse 5
321	Istposition in Schritten der Schrittmotorachse 6
322	Istposition in Schritten der Schrittmotorachse 7
323	Istposition in Schritten der Schrittmotorachse 8
324	Istwert A/D-Kanal 1 MC100-AC (Modul 1)
325	Istwert A/D-Kanal 2 MC100-AC (Modul 1)
326	Istwert A/D-Kanal 3 MC100-AC (Modul 1)
327	Istwert A/D-Kanal 4 MC100-AC (Modul 1)
328	Istwert A/D-Kanal 5 MC100-AC (Modul 2)
329	Istwert A/D-Kanal 6 MC100-AC (Modul 2)
330	Istwert A/D-Kanal 7 MC100-AC (Modul 2)
331	Istwert A/D-Kanal 8 MC100-AC (Modul 2)
332 – 346	Reserviert für Systemerweiterung
347	Beschleunigung der Servoachse 1
348	Maximaler Schleppabstand der Servoachse 1
349	Beschleunigung der Servoachse 2

350	Maximaler Schleppabstand der Servoachse 2
351	Beschleunigung der Servoachse 3
352	Maximaler Schleppabstand der Servoachse 3
353	Beschleunigung der Servoachse 4
354	Maximaler Schleppabstand der Servoachse 4
355	Beschleunigung der Servoachse 5
356	Maximaler Schleppabstand der Servoachse 5
357	Beschleunigung der Servoachse 6
358	Maximaler Schleppabstand der Servoachse 6
359	Beschleunigung der Servoachse 7
360	Maximaler Schleppabstand der Servoachse 7
361	Beschleunigung der Servoachse 8
362	Maximaler Schleppabstand der Servoachse 8
363	Umrechnungsfaktor der Schrittmotorachse 1
364	Rampe der Schrittmotorachse 1
365	Start- Stopfrequenz der Schrittmotorachse 1
366	Schrittüberwachung der Schrittmotorachse 1
367	Umrechnungsfaktor der Schrittmotorachse 2
368	Rampe der Schrittmotorachse 2
369	Start- Stopfrequenz der Schrittmotorachse 2
370	Schrittüberwachung der Schrittmotorachse 2
371	Umrechnungsfaktor der Schrittmotorachse 3
372	Rampe der Schrittmotorachse 3
373	Start- Stopfrequenz der Schrittmotorachse 3
374	Schrittüberwachung der Schrittmotorachse 3
375	Umrechnungsfaktor der Schrittmotorachse 4
376	Rampe der Schrittmotorachse 4
377	Start- Stopfrequenz der Schrittmotorachse 4
378	Schrittüberwachung der Schrittmotorachse 4
379	Umrechnungsfaktor der Schrittmotorachse 5
380	Rampe der Schrittmotorachse 5
381	Start- Stopfrequenz der Schrittmotorachse 5
382	Schrittüberwachung der Schrittmotorachse 5
383	Umrechnungsfaktor der Schrittmotorachse 6
384	Rampe der Schrittmotorachse 6
385	Start- Stopfrequenz der Schrittmotorachse 6
386	Schrittüberwachung der Schrittmotorachse 6
387	Umrechnungsfaktor der Schrittmotorachse 7
388	Rampe der Schrittmotorachse 7
389	Start- Stopfrequenz der Schrittmotorachse 7
390	Schrittüberwachung der Schrittmotorachse 7
391	Umrechnungsfaktor der Schrittmotorachse 8
392	Rampe der Schrittmotorachse 8
393	Start- Stopfrequenz der Schrittmotorachse 8
394	Schrittüberwachung der Schrittmotorachse 8
395	Sollwert D/A-Kanal 1 MC100-CA (Modul 1)
396	Bezugsgröße D/A-Kanal 1 MC100-CA (Modul 1)
397	Sollwert D/A-Kanal 2 MC100-CA (Modul 1)
398	Bezugsgröße D/A-Kanal 2 MC100-CA (Modul 1)
399	Sollwert D/A-Kanal 3 MC100-CA (Modul 1)
400	Bezugsgröße D/A-Kanal 3 MC100-CA (Modul 1)
401	Sollwert D/A-Kanal 4 MC100-CA (Modul 1)
402	Bezugsgröße D/A-Kanal 4 MC100-CA (Modul 1)
403	Sollwert D/A-Kanal 1 MC100-CA (Modul 2)
404	Bezugsgröße D/A-Kanal 1 MC100-CA (Modul 2)
405	Sollwert D/A-Kanal 2 MC100-CA (Modul 2)
406	Bezugsgröße D/A-Kanal 2 MC100-CA (Modul 2)
407	Sollwert D/A-Kanal 3 MC100-CA (Modul 2)

408	Bezugsgröße D/A-Kanal 3 MC100-CA (Modul 2)
409	Sollwert D/A-Kanal 4 MC100-CA (Modul 2)
410	Bezugsgröße D/A-Kanal 4 MC100-CA (Modul 2)
411 – 430	Reserviert für Systemerweiterungen
431	Programmnummer NC-Programm 1
432	Zeilennummer NC-Prgramm 1
433	Zeilensprung NC-Programm 1
434	Fehlernummer NC-Programm 1
435	Programmnummer NC-Programm 2
436	Zeilennummer NC-Programm 2
437	Zeilensprung NC-Programm 2
438	Fehlernummer NC-Programm 2
439	Reserviert für Systemerweiterung
440 – 465	Formatvariable NC-Funktionen A – Z
466 – 469	Reserviert für Systemerweiterungen
470 – 495	Daten NC-Funktionen A – Z
496 – 543	Reserviert für Systemerweiterungen
544	Statusvariable MC100-TR Modul 1
545	Statusvariable MC100-TR Modul 2
546	Statusvariable MC100-TR Modul 3
547	Statusvariable MC100-TR Modul 4
548	Statusvariable MC100-TR Modul 5
549	Statusvariable MC100-TR Modul 6
550	Kurzschluß Ausgang 1 – 16
551	Übertemperatur Ausgang 1 – 16
552	Leitungsbruch Ausgang 1 – 16 (Fehler)
553	Leitungsbruch Ausgang 1 – 16 (Status)
554	Kurzschluß Ausgang 17 – 32
555	Übertemperatur Ausgang 17 – 32
556	Leitungsbruch Ausgang 17 – 32 (Fehler)
557	Leitungsbruch Ausgang 17– 32 (Status)
558	Kurzschluß Ausgang 33 – 48
559	Übertemperatur Ausgang 33 – 48
560	Leitungsbruch Ausgang 33 – 48 (Fehler)
561	Leitungsbruch Ausgang 33 – 48 (Status)
562	Kurzschluß Ausgang 49 – 64
563	Übertemperatur Ausgang 49 – 64
564	Leitungsbruch Ausgang 49 – 64 (Fehler)
565	Leitungsbruch Ausgang 49 – 64 (Status)
566	Kurzschluß Ausgang 65 – 80
567	Übertemperatur Ausgang 65 –80
568	Leitungsbruch Ausgang 65 – 80 (Fehler)
569	Leitungsbruch Ausgang 65 – 80 (Status)
570	Kurzschluß Ausgang 81 – 96
571	Übertemperatur Ausgang 81 – 96
572	Leitungsbruch Ausgang 81 – 96 (Fehler)
573	Leitungsbruch Ausgang 81 – 96 (Status)
574	Kurzschluß Ausgang 97 – 112
575	Übertemperatur Ausgang 97 – 112
576	Leitungsbruch Ausgang 97 – 112 (Fehler)
577	Leitungsbruch Ausgang 97 – 112 (Status)
578	Kurzschluß Ausgang 113 – 128
579	Übertemperatur Ausgang 113 – 128
580	Leitungsbruch Ausgang 113 – 128 (Fehler)
581	Leitungsbruch Ausgang 113 – 128 (Status)
582 – 587	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 1)
588 – 593	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 2)
594 – 599	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 3)

600 – 605	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 4)
606 – 611	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 5)
612 – 617	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 6)
618 – 623	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 1)
624 – 629	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 2)
630 – 635	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 3)
636 – 641	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 4)
642 – 647	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 5)
648 – 653	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 6)
654 – 659	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 7)
660 – 665	Istwert Temperatur Kanal 1 – 6 MC100-TR (Modul 8)
666 – 671	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 7)
672 – 677	Sollwert Temperatur Kanal 1 – 6 MC100-TR (Modul 7)
678	Statusvariable MC100-TR (Modul 7)
679	Statusvariable MC100-TR (Modul 8)
680 – 699	Reserviert für Systemerweiterungen
700 – 16000	Frei verfügbar