

# MC-1B

# Befehlsdokumentation

Befehlsreferenz für die Programmierung in MC-1B Sprache

# MC-1B Befehlsdokumentation

---

## Befehlsreferenz für die Programmierung in MC-1B Sprache

Jede Vervielfältigung dieses Dokumentes sowie der zugehörigen Software oder Firmware bedarf der vorherigen schriftlichen Zustimmung durch die Fa. MICRO DESIGN Industrieelektronik GmbH. Zuwiderhandlung wird strafrechtlich verfolgt. Alle Rechte an dieser Dokumentation sowie der zugeordneten Software, Hardware und/oder Firmware liegen bei MICRO DESIGN.

Im Text erwähnte Warenzeichen werden unter Berücksichtigung und Anerkennung der Inhaber der jeweiligen Warenzeichen verwendet. Ein getrennte Kennzeichnung verwendeter Warenzeichen erfolgt im Text ggf. nicht durchgängig. Die Nichterwähnung oder Nichtkennzeichnung eines Warenzeichens bedeutet nicht, daß das entsprechende Zeichen nicht anerkannt oder nicht eingetragen ist.

Insofern diesem Dokument eine System- und/oder Anwendungssoftware zugeordnet ist, sind Sie als rechtmäßiger Erwerber berechtigt, diese Software zusammen mit MICRO DESIGN Hardwarekomponenten an Ihre Endkunden lizenzfrei weiterzugeben, solange keine getrennte, hiervon abweichende Vereinbarung getroffen wurde. Beinhaltet die diesem Dokument zugeordnete Software Beispielprogramme und Beispielapplikationen, so dürfen Sie diese nicht unverändert an Ihren Endkunden weitergeben, sondern ausschließlich zum eigenen Gebrauch und zu Lernzwecken verwenden.

Einschränkung der Gewährleistung: Es wird keine Haftung für die Richtigkeit des Inhaltes dieses Dokumentes übernommen. Da sich Fehler, trotz aller Bemühungen und Kontrollen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Technische Änderungen an der diesem Dokument zugeordneten Software, Hardware und/oder Firmware behalten wir uns jederzeit – auch unangekündigt – vor.

Copyright © 1998-2003 MICRO DESIGN Industrieelektronik GmbH.

Waldweg 55, 88690 Uhldingen, Deutschland

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

E-Mail: [technik@microdesign.de](mailto:technik@microdesign.de)

<http://www.microdesign.de>

**We like to move it!™**

# Inhaltsverzeichnis

<b>Kapitel 1 Einführung .....</b>	<b>13</b>
n Das offene Konzept der MC200.....	13
n Grundlage dieser Beschreibung .....	13
n eMC200 – Wichtiger Hinweis! .....	13
1.1 Über diese Dokumentation .....	14
n Struktur und Nummerierung.....	15
n Formatierung in dieser Dokumentation .....	15
n Dokumentation der PC-Software.....	15
<b>Kapitel 2 Programmieren mit MC-1B .....</b>	<b>17</b>
n Alles in Einem.....	17
n Warum nicht IEC1131? .....	17
2.1 Grundsätzliche Vereinbarungen.....	18
n Sprachdefinition .....	18
n Kommentare .....	19
n Labels (Sprungmarken).....	20
n Makros.....	21
n Struktur des Quelltext-Projekts .....	21
n Modulare Programmierung .....	22
2.2 Datentypen .....	23
n Variablen .....	23
n Merker .....	23
n Ausgänge.....	23
n Eingänge.....	23
n Konstanten.....	24
n Labels.....	24
2.3 Zyklen, Abläufe und Tasks.....	25
n Wie läuft ein Programm in MC-1B ab?.....	25
n Verwenden von Tasks .....	26
n Zyklen programmieren .....	27
n Zyklen mit dynamischen Sprüngen.....	28
2.4 Unser Ergebnisspeicher: das Bitergebnis .....	29
n Wie wird das Bitergebnis beeinflusst?.....	29
n Wie wirkt sich das Bitergebnis aus? .....	29
n Bitergebnisschieberegister .....	30
<b>Kapitel 3 Befehlsübersicht .....</b>	<b>31</b>
n Kennzeichnung der Parameter .....	31
n Zusammenspiel mit dem Bitergebnisspeicher .....	31

n	Wichtiger Hinweis .....	31
3.1	Befehle nach Funktionsgruppen .....	32
n	Definitionsbefehle .....	32
n	Compilieranweisungen .....	32
n	Merkerbefehle .....	33
n	Ein-/Ausgangsbefehle .....	34
n	Analog I/O .....	35
n	Variablenbefehle .....	36
n	Variablenvergleichsbefehle .....	39
n	Programmablaufbefehle .....	40
n	Positionierbefehle .....	41
n	Display und Texte .....	43
n	Datenkonvertierungsbefehle .....	43
n	Serielle Anbindung .....	44
n	Systembefehle .....	45
n	Makros .....	45
3.2	Alphabetische Befehlsübersicht .....	47
n	ADD_II .....	47
n	ADD_IV .....	47
n	ADD_VA .....	48
n	ADD_VI .....	48
n	ADD_VV .....	49
n	AUS_A .....	49
n	AUS_AI .....	50
n	AUS_M .....	50
n	AUS_MI .....	51
n	CHGVEL .....	51
n	CHGVPO .....	52
n	CLRERR .....	53
n	CLRSER .....	53
n	DEC_V .....	54
n	DEC_VV .....	55
n	DEF_A .....	56
n	DEF_E .....	56
n	DEF_M .....	57
n	DEF_V .....	57
n	DEF_W .....	58
n	DIV_II .....	58
n	DIV_IV .....	59

n	DIV_VA.....	59
n	DIV_VI.....	60
n	DIV_VV.....	60
n	EIN_A.....	61
n	EIN_AI.....	61
n	EIN_M.....	62
n	EIN_MI.....	62
n	ENDM.....	63
n	EXITM.....	63
n	FLASH.....	64
n	GEHUPRI.....	67
n	GEHUPRJ.....	68
n	GEHUPRN.....	68
n	GEHUPRV.....	69
n	GETPAR.....	70
n	INC_V.....	70
n	INC_VV.....	71
n	LAD_A.....	71
n	LAD_AI.....	72
n	LAD_DT.....	72
n	LAD_DV.....	73
n	LAD_E.....	74
n	LAD_EI.....	74
n	LAD_II.....	75
n	LAD_IV.....	75
n	LAD_M.....	76
n	LAD_MI.....	76
n	LAD_MV.....	77
n	LAD_P1.....	78
n	LAD_P2.....	79
n	LAD_P3.....	80
n	LAD_P4.....	81
n	LAD_VA.....	82
n	LAD_VI.....	82
n	LAD_VL.....	83
n	LAD_VM.....	84
n	LAD_VV.....	85
n	MACRO.....	85
n	MOD_A.....	86

n	MOD_AI .....	87
n	MOD_M .....	87
n	MOD_MI .....	88
n	MUL_II .....	88
n	MUL_IV .....	89
n	MUL_VA .....	89
n	MUL_VI .....	90
n	MUL_VV .....	90
n	NLAD_A .....	91
n	NLAD_E .....	91
n	NLAD_M .....	92
n	NODER_A .....	92
n	NODER_E .....	93
n	NODER_M .....	93
n	NUND_A .....	94
n	NUND_E .....	94
n	NUND_M .....	95
n	ODER_A .....	95
n	ODER_E .....	96
n	ODER_M .....	96
n	ODER_II .....	97
n	ODER_IV .....	98
n	ODER_VA .....	98
n	ODER_VI .....	99
n	ODER_VV .....	99
n	PRLABS .....	100
n	PRLREL .....	101
n	PWRDRV .....	102
n	RCVSER .....	103
n	SETAIO .....	103
n	SETDSP .....	104
n	SETEDI .....	110
n	SETFUN .....	112
n	SETNUL .....	113
n	SETOVR .....	114
n	SETPAR .....	115
n	SETRMP .....	115
n	SETSER .....	116
n	SETVEL .....	119

n	SLL_V .....	119
n	SLL_VV .....	120
n	SNDSER .....	120
n	SPRING .....	121
n	SPRINGJ .....	121
n	SPRINGN .....	122
n	SRL_V .....	122
n	SRL_VV .....	123
n	STCONT .....	123
n	STHOME .....	124
n	STIABS .....	125
n	STICIR .....	126
n	STIREL .....	127
n	STOPDN .....	128
n	STOPEM .....	128
n	STPABS .....	129
n	STPREL .....	129
n	STPRLD .....	130
n	SUB_II .....	130
n	SUB_IV .....	131
n	SUB_VA .....	131
n	SUB_VI .....	132
n	SUB_VV .....	132
n	TEXT .....	133
n	UND_A .....	134
n	UND_E .....	134
n	UND_II .....	135
n	UND_IV .....	136
n	UND_M .....	136
n	UND_VA .....	137
n	UND_VI .....	137
n	UND_VV .....	138
n	UPREND .....	138
n	UPRENDJ .....	139
n	UPRENDN .....	139
n	VERG_II .....	140
n	VERG_IV .....	141
n	VERG_VA .....	142
n	VERG_VI .....	143

n	VERG_VV .....	144
n	WART_A...WART_E .....	145
n	XODER_A.....	146
n	XODER_E.....	146
n	XODER_M .....	147
n	XOR_II .....	147
n	XOR_IV.....	148
n	XOR_VA.....	148
n	XOR_VI.....	149
n	XOR_VV.....	149
n	#ELSE .....	150
n	#ENDIF .....	150
n	#IF.....	151
n	#INCLUDE .....	152
<b>Kapitel 4 Besondere Anwendungen .....</b>		<b>155</b>
4.1	Messfunktionen .....	156
n	Ablauf der Messung .....	157
n	Auswerten von mehreren Meßwerten.....	158
4.2	Display-Programmierung .....	160
n	SPS-Texte .....	160
n	Texte formatieren und darstellen .....	161
n	Zahlenwerte formatieren und darstellen .....	161
n	Zahlenwerte editieren .....	162
n	Tastaturbelegung .....	162
n	Die Format-Variablen V_ANZMSK/V_INPMSK.....	163
4.3	Analog Ein-/ Ausgänge .....	164
n	Schreiben von analogen Ausgängen .....	164
n	Lesen von analogen Eingängen .....	164
n	Mittelwertbildung.....	164
n	Werteaktualisierung .....	165
n	Umwandlung Analog/Digital .....	165
4.4	Timer .....	166
n	Die integrierten Systemtimer .....	166
n	Virtuelle Timer.....	168
n	Globaler Systemtimer.....	169
n	Blinkmerker.....	170
4.5	Serielltes Modul .....	171
n	Sende- und Empfangspuffer .....	171
n	Wenn es eng wird: Pufferüberläufe.....	172



n Beispiel: Barcode-Leser .....	172
4.6 EAU-T Emulation .....	174
n Funktionsweise .....	174
n Parametrierung .....	174
n Beispiel .....	174
<b>Kapitel 5 System- und Achsparameter .....</b>	<b>177</b>
5.1 Systemparameter .....	178
n Ändern der Parameter .....	178
n Übersicht Systemparameter .....	178
5.2 Achsparameter .....	179
n Ändern der Parameter .....	179
n Berechnung der Parameter-Nummer .....	179
n Übersicht Achsparameter für Servomotorcontroller .....	180
n Übersicht Achsparameter für Schrittmotorcontroller .....	186
n Einrichtung des Zählsystems .....	190
5.3 Parameter serielles Modul .....	191
n Berechnung der Parameter-Nummer .....	191
n Übersicht Parameter serielles Modul .....	191
<b>Kapitel 6 Systemdaten .....</b>	<b>193</b>
n Vom System reservierte Speicherbereiche .....	193
n Lese-/Schreibzugriff .....	193
6.1 Bitergebnisschieberegister .....	194
6.2 Ergebnismerker- und Variablen .....	195
n Variable VARERG – Generelle Ergebnisvariable .....	195
n Variable DIV_REST – Divisionsrest .....	195
n Variable MUL_REST – Multiplikationsüberlauf .....	196
n Vergleichsergebnisse .....	196
6.3 Interne Systemdaten .....	197
6.4 SPS-Tasks .....	198
6.5 RS485 Kommunikation .....	199
6.6 Angeschlossene Module .....	200
6.7 Achsenstatus .....	202
n Bitcodierte Achsstatus-Parameter .....	202
n Achse 1 .....	203
n Achse 2 .....	204
n Achse 3 .....	205
n Achse 4 .....	206
n Achse 5 .....	207
n Achse 6 .....	208

n Achse 7.....	209
n Achse 8.....	210
n Achse 9.....	211
n Achse 10.....	212
n Achse 11.....	213
n Achse 12.....	214
n Achse 13.....	215
n Achse 14.....	216
n Achse 15.....	217
n Achse 16.....	218
6.8 Meßfunktionsdaten .....	219
6.9 Systemtimer .....	220
6.10 Blinkmerker .....	222
6.11 Zählermodul .....	223
6.12 Analoge Ein- und Ausgänge.....	224
6.13 EAU-T Emulation.....	225
6.14 Displayprogrammierung .....	226
6.15 Tastaturen.....	227
n MC200BED (stehend, 20 Tasten) .....	227
n MC200BED (liegend, 36 Tasten) .....	228
n MC200HT.....	228
6.16 Elektrisches Handrad.....	230
6.17 Eingebaute serielle Schnittstelle .....	231
6.18 Serielles Erweiterungsmodul .....	232
6.19 Übersicht Speichermanagement .....	234
n MC200CPU (Typen B und C).....	234
n MC200PROFI/B .....	235
n Aufbau des Universal Data Block (UDB) .....	236
<b>Kapitel 7 Beispiele .....</b>	<b>239</b>
n Achspositionierung .....	239
n Lauflicht.....	239
n Display-Programmierung.....	239
n Mixed Mode (PC-Anbindung) .....	239
7.1 Achspositionierung.....	240
7.2 Lauflicht .....	242
7.3 Displayprogrammierung .....	246
n Hauptprogramm (Datei MAIN.MC) .....	246
n Textdefinitionen (Datei TEXT.MCT) .....	249
7.4 Mixed Mode (PC-Anbindung) .....	250

n	Dokumentationen .....	257
n	Installation der Software .....	257
n	Tabellen.....	259
n	Abbildungen .....	261
n	Zum Thema SPS-Programmierung .....	262
n	Zum Thema Achsen .....	262

## n Raum für Ihre Notizen

# Kapitel 1 Einführung

Wir gratulieren Ihnen zu Ihrer Entscheidung, Ihr Projekt mit einem MC200 System und der MC-1B Programmiersprache zu entwickeln! Denn das MC200 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. Gerade die konsequent durchgängige Modularisierung – für jede Funktion des Systems ein getrenntes Modul – ermöglicht auch Ihnen den Aufbau von skalierbaren, einfach an Sondermaschinen anpaßbaren Programmen.

## n Das offene Konzept der MC200

Durch ein offenes Konzept haben Sie natürlich auch verschiedene Möglichkeiten dieses System zu programmieren. Alle diese Varianten sind selbstverständlich frei miteinander kombinierbar:

- n MC200 als PC-Steuerung: Unsere schnellen Windows NT Treiber ermöglichen eine Programmierung des Systems nahezu unter Echtzeitbedingungen. Sie können ein Steuerungsprogramm schreiben ohne auch nur eine Zeile SPS-Programm zu tippen – einfach mit einer Standard PC-Programmiersprache wie z.B. Pascal oder Basic.
- n MC200 als Profibus-Slave: Durch unsere DP-Slave Anbindung können Sie den gesamten Ablauf des Systems über den Profibus steuern – natürlich auch wieder ohne eine Zeile SPS-Programm zu schreiben.
- n MC200 als SPS-System: Die am weitesten verbreitete Variante ist jedoch weiterhin die SPS-Programmierung. Auch hier bietet Ihnen das MC200 System eine enorme Flexibilität durch die einfach erlernbare und trotzdem sehr mächtige Programmiersprache MC-1B. Um dieses Thema geht es in dieser Dokumentation.

## n Grundlage dieser Beschreibung

Die Beschreibung des Funktionsumfangs, der Wirkungsweise der SPS-Befehle, der erhältlichen Erweiterungsmodule, der PC-Software sowie des MC200 Systems als Ganzem beruht auf dem Entwicklungsstand Januar 2003.

Wenn Sie mit älteren Versionen der Produkte arbeiten ist es möglich, daß einzelne Funktionen nicht so arbeiten wie in dieser Dokumentation beschrieben. Bitte aktualisieren Sie in diesem Fall den Versionsstand der Einzelkomponenten. Aktualisierte Versionen der VMC Workbench und der MC200 Betriebssysteme erhalten Sie stets im Internet unter <http://www.microdesign.de>.

## n eMC200 – Wichtiger Hinweis!

Die Beschreibung der spezifischen eMC200 Befehle und Erweiterungen ist noch nicht in diese Dokumentation eingeflossen. Bitte beachten Sie deshalb die entsprechenden Hinweise auf unserer Webseite, in den Definitions- und Hilfedateien der VMC Workbench X2 sowie den erhältlichen Kurzdokumentationen.

Wir werden die fehlenden Informationen raschestmöglich nachreichen!

# 1.1 Über diese Dokumentation

Die hier vorliegende Dokumentation ist logisch in folgende Kapitel gegliedert:

## **Kapitel 1 - Einführung (ab Seite 13)**

Dieses Kapitel lesen Sie gerade. Es gibt Ihnen einen kurzen Überblick über das MC200 System und den Aufbau 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-1B (ab Seite 17)**

Im zweiten Kapitel wenden wir uns der grundsätzlichen Idee der MC-1B Programmierung zu. Dieses Kapitel ist insbesondere dann für Sie interessant, wenn Sie bislang noch keine Programme in MC-1B 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 31)**

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

## **Kapitel 4 - Besondere Anwendung (ab Seite 155)**

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 MC200 Familie.

## **Kapitel 5 - System- und Achsparameter (ab Seite 177)**

In diesem Kapitel erfahren Sie alles, was Sie über die Parametrierung des MC200 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.

## **Kapitel 6 - Systemdaten (ab Seite 193)**

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.

## **Kapitel 7 - Beispiele (ab Seite 239)**

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-1B Sprache kommen.

## **Anhänge (ab Seite 255)**

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-1B 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-Hilfdateien auf der VMC Workbench CD. Eine gedruckte Fassung der VMC Workbench Dokumentation ist nicht verfügbar.

n Raum für Ihre Notizen



# Kapitel 2 Programmieren mit MC-1B

Wer schon einmal in SPS-Sprachen programmiert hat, dem wird die MC-1B 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:

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

Das alles trifft auch auf die MC-1B 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-1B Systems eingehen.

## n Alles in Einem

MC-1B integriert alle Funktionen die Sie im Umfeld einer MC200 Umgebung nutzen können, in eine einheitliche Programmiersprache. Insbesondere auf die einfache und effektive Programmierung der Positionierbewegungen wurde sehr großen Wert gelegt. Im Einzelnen können Sie folgende Geräte direkt und ohne Umwege aus der MC-1B Sprache ansprechen:

- n Servomotorachsen
- n Schrittmotorachsen
- n Digitale Aus- und Eingänge
- n Analoge Aus- und Eingänge
- n Eine Auswahl von Displays, Handbedienteilen und Tastaturen
- n Schnelle Zählermodule
- n Protokolldrucker mit serieller Schnittstelle

Außerdem ist eine sehr einfache Integration in ein komplexes Feldbussystem wie den Profibus möglich. Noch einfacher wird die Handhabung einer Dezentralisierung mit unseren MC200 Busmodulen: dann nämlich müssen Sie sich in der Programmierung um gar nichts kümmern, Sie können alle dezentralisierten Module direkt ansprechen.

## n Warum nicht IEC1131?

Wir haben lange darüber nachgedacht ob wir für das MC200 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-1B 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-1B 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-1B Sprache gearbeitet haben, dann sollten Sie sich unbedingt mit diesem Abschnitt beschäftigen.

### n Sprachdefinition

In der MC-1B 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 Ressourcen arbeiten, denn die MC-1B 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:

- n Der symbolische Name darf nicht länger als 23 Zeichen sein.
- n 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.
- n 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 //Kommentar

### 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 Schlüssel schalter 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-1B 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_Bli nkl i cht // Ausgang A_Bli nkl i cht umschalten

```

Eine **Übersicht aller Makro-Befehle** sowie weitere Erläuterungen zu diesem Thema finden Sie im Kapitel 3.1 - Befehle nach Funktionsgruppen unter "Makros" (Seite 45).

## n Struktur des Quelltext-Projekts

Ihr MC-1B 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-1B 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.

## 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-1B 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-Programms 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 32).

## 2.2 Datentypen

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

### n Variablen

SPS-Variablen sind im MC-1B System stets 32 Bit breit, dies bedeutet folgende Limitierungen:

- ⇒ Größter möglicher Wert: 2147483647
- ⇒ Kleinster möglicher Wert: -2147483648

Im MC-1B 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 sofort mit dem Faktor 10, bis eine ganze Zahl dabei herauskommt.

Variablen können mit MC-1B 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.

Bitte beachten Sie, dass konstante Werte im SPS-Programm nicht den vollen 32-Bit Wertebereich ausschöpfen können: Aufgrund des Objekt-Formats der SPS-Engine können bei konstanten Werten maximal 24 Bit adressiert werden. Sollen Sie in einem konkreten Fall konstante Werte größer als 24 Bit benötigen, verwenden Sie bitte die binären Schiebefunktionen SLL\_VA und SRL\_VA bzw. SLL\_VV und SRL\_VV.

## 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.

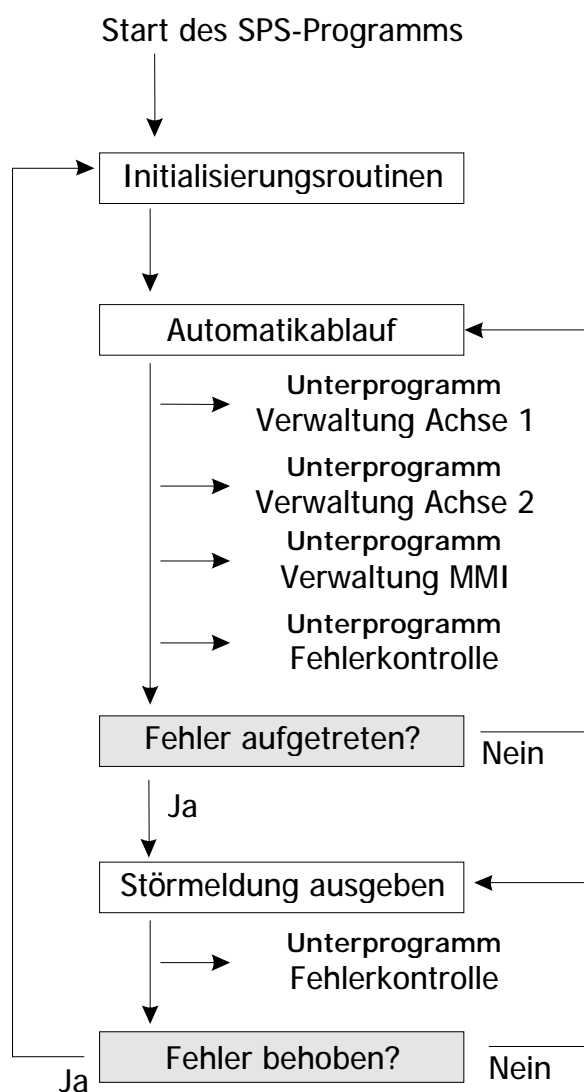


## 2.3 Zyklen, Abläufe und Tasks

Das MC200 System ist von Haus aus eine Mehrtask-Maschine, d.h. es können mehrere Abläufe voneinander unabhängig bearbeitet werden. Diese Möglichkeit wird von der MC-1B Programmiersprache in vollem Umfang unterstützt. Noch wichtiger ist hingegen zu verstehen, daß das MC200 System nicht als eine Zyklus-Maschine im eigentlichen Sinn arbeitet. Im Gegensatz zu einer Zyklusmaschine arbeitet die MC200 stets asynchron; sämtliche Betriebszustände – wie z.B. der Zustand von Ein- und Ausgängen – werden fortlaufend aktualisiert, auch die Programmausführung ist nicht an einen festen Zyklus gebunden.

### n Wie läuft ein Programm in MC-1B ab?

Die wichtigste Grundlage, die man sich bei der Programmierung eines Systems mit MC-1B verinnerlichen muß, ist: es wird nicht mit einem festen Zyklus-Ablauf gearbeitet! Mit MC-1B 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-1B

Wie man in diesem Beispiel sieht, gibt es mit MC-1B 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-1B Programm

Wenn wir obigen Ablauf einmal in der MC-1B 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 Verwaltung Achse 1
      GEHUPRI Achse2Verwalt               // Unterprogramm Verwaltung 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 FehlerMel dung              // 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-1B 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:

- n 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.
- n 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-1B 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-1B 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
VERG_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-Variablen auf nächsten Zyklus setzen

Zyklus2: // Label (Sprungmarke) für zweiten Zyklus
VERG_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-Variablen auf nächsten Zyklus setzen

```

Auch eine "klassische" Zyklusmaschine ist also mit der MC-1B 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-1B 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-1B 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-1B 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 Bitergebnis 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 Befehlsübersicht, 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 6.1 - Bitergebnisschieberegister (Seite 194)

# Kapitel 3 Befehlsübersicht

In diesem Kapitel finden Sie eine vollständige Übersicht aller MC-1B 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 32)

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

## Abschnitt 3.2 - Alphabetische Befehlsübersicht (Seite 47)

Hier finden Sie eine vollständige, alphabetische Referenz aller MC-1B 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  $\boxed{V}$

Diese Information definiert den Datentyp (vgl. Kapitel 2.2 - Datentypen ab Seite 23) , der von diesem Befehl erwartet wird. Folgende Datentypen sind hierbei zulässig:

n  $\boxed{V}$  - Variable

n  $\boxed{M}$  - Merker

n  $\boxed{E}$  - Eingang

n  $\boxed{A}$  - Ausgang

n  $\boxed{K}$  - 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 32.
- 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-1B Programmierung bereits vertraut sind. Beachten Sie ggf. die Erläuterungen in Kapitel 2 - Programmieren mit MC-1B ab Seite 17.

## 3.1 Befehle nach Funktionsgruppen

### n Definitionsbefehle

Befehl	Bedeutung	Seite
DEF_A Ausgangsnr. <input type="checkbox"/> , Symbolname <input type="checkbox"/>	DEF_A weist einem Ausgang einen symbolischen Namen zu.	56
DEF_E Eingangsnr. <input type="checkbox"/> , Symbolname <input type="checkbox"/>	DEF_E weist einem Eingang einen symbolischen Namen zu.	56
DEF_M Merckernr. <input type="checkbox"/> , Symbolname <input type="checkbox"/>	DEF_M weist einem Merker einen symbolischen Namen zu.	57
DEF_V Variablennr. <input type="checkbox"/> , Symbolname <input type="checkbox"/>	DEF_V weist einer Variablen einen symbolischen Namen zu.	57
DEF_W Wert <input type="checkbox"/> , Symbolname <input type="checkbox"/>	DEF_W weist dem Wert einer Konstanten einen symbolischen Namen zu.	58

n Tabelle 1 – Definitionsbefehle















### n Compileranweisungen

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.	150
#ENDIF	#ENDIF beendet eine bedingte Compilierung.	150
#IF Abfragebedingung <input type="checkbox"/> #IF Wert1 <input type="checkbox"/> EQ Wert2 <input type="checkbox"/>	#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.	151
#INCLUDE Quellcodedateiname <input type="checkbox"/>	#INCLUDE fügt eine weitere Quellcodedatei an der aktuellen Stelle ein.	152

n Tabelle 2 – Compileranweisungen







## n Merkerbefehle

Befehl	Bedeutung	Seite
AUS_M Merker 	Der Befehl AUS_M löscht den angegebenen Merker.	50
AUS_MI Zeiger 	Der Befehl AUS_MI löscht den Merker, der durch den Zeiger bestimmt wird.	51
EIN_M Merker 	EIN_M setzt den angegebenen Merker.	62
EIN_MI Zeiger 	EIN_MI setzt den durch den angegebenen Zeiger bestimmten Merker.	62
LAD_M Merker 	LAD_M lädt den Zustand des angegebenen Merkers in den Bitergebnisspeicher.	76
LAD_MI Zeiger 	LAD_MI überträgt den Zustand des durch den angegebenen Zeiger bestimmten Merkers in den Bitergebnisspeicher.	76
MOD_M Merker 	MOD_M überträgt den Zustand des Bitergebnisspeichers in den angegebenen Merker.	87
MOD_MI Zeiger 	MOD_MI überträgt den Zustand des Bitergebnisspeichers in den durch den angegebenen Zeiger bestimmten Merker.	88
NLAD_M Merker 	NLAD_M lädt den invertierten Zustand des Merkers in den Bitergebnisspeicher.	92
NODER_M Merker 	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.	93
NUND_M Merker 	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.	95
ODER_M Merker 	ODER_M verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Merkers mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	96
UND_M Merker 	UND_M verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Merkers mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.	136
XODER_M Merker 	XODER_M verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Merkers mit einem logischen Exklusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.	147

n Tabelle 3 – Merkerbefehle



## n Ein-/Ausgangsbefehle

Befehl	Bedeutung	Seite
AUS_A Ausgang <input type="checkbox"/>	Der Befehl AUS_A schaltet den angegebenen Ausgang aus.	49
AUS_AI Zeiger <input type="checkbox"/>	Der Befehl AUS_AI schaltet den Ausgang aus, der durch den Zeiger bestimmt wird.	50
EIN_A Ausgang <input type="checkbox"/>	EIN_A schaltet den angegebenen Ausgang ein.	61
EIN_AI Zeiger <input type="checkbox"/>	EIN_AI schaltet den durch den angegebenen Zeiger bestimmten Ausgang ein.	61
LAD_A Ausgang <input type="checkbox"/>	LAD_A lädt den Zustand des angegebenen Ausganges in den Bitergebnisspeicher.	71
LAD_AI Zeiger <input type="checkbox"/>	LAD_AI lädt den Zustand des durch den Zeiger bestimmten Ausganges in den Bitergebnisspeicher.	72
LAD_E Eingang <input type="checkbox"/>	LAD_E lädt den Zustand des angegebenen Eingangs in den Bitergebnisspeicher.	74
LAD_EI Zeiger <input type="checkbox"/>	LAD_EI lädt den Zustand des durch den Zeiger angegebenen Eingangs in den Bitergebnisspeicher.	74
MOD_A Ausgang <input type="checkbox"/>	MOD_A überträgt den Zustand des Bitergebnisspeichers auf den angegebenen Ausgang.	86
MOD_AI Zeiger <input type="checkbox"/>	MOD_AI überträgt den Zustand des Bitergebnisspeichers in den durch den angegebenen Zeiger bestimmten Ausgang.	87
NLAD_E Eingang <input type="checkbox"/>	NLAD_E lädt den invertierten Zustand eines Eingangs in den Bitergebnisspeicher.	91
NLAD_A Ausgang <input type="checkbox"/>	NLAD_A lädt den invertierten Zustand des Ausganges in den Bitergebnisspeicher.	91
NUND_A Ausgang <input type="checkbox"/>	NUND_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Ausganges mit einem logischen UND. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	94
NUND_E Eingang <input type="checkbox"/>	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.	94
UND_E Eingang <input type="checkbox"/>	UND_E verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Eingangs mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.	134
UND_A Ausgang <input type="checkbox"/>	UND_A verknüpft den aktuellen Zustand des Bitergebnisspeichers und den Zustand des angegebenen Ausganges mit einem logischen "UND". Das Ergebnis wird im Bitergebnisspeicher abgelegt.	134
NODER_A Ausgang <input type="checkbox"/>	NODER_A verknüpft den Zustand Bitergebnisspeichers und den invertierten Zustand des angegebenen Ausganges mit einem logischen ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	92
NODER_E Eingang <input type="checkbox"/>	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.	93

Befehl	Bedeutung	Seite
ODER_A Ausgang 	ODER_A verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Ausgangs mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	95
ODER_E Eingang 	ODER_E verknüpft die Zustände des Bitergebnisspeichers und des angegebenen Eingangs mit einem logischer ODER. Das Ergebnis dieser Verknüpfung wird wiederum im Bitergebnis abgelegt.	96
XODER_A Ausgang 	XODER_A verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Ausgangs mit einem logischen Exklusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.	146
XODER_E Eingang 	XODER_E verknüpft den Zustand des Bitergebnisspeichers und den Zustand des angegebenen Eingangs mit einem logischen Exklusiv-ODER (XOR). Das Ergebnis dieser Verknüpfung wird wieder im Bitergebnisspeicher abgelegt.	146

n Tabelle 4 – Ein-/ Ausgangsbefehle

## n Analog I/O





























Befehl	Bedeutung	Seite
SETAIO Analoger Ausgang  , Wert 	SETAIO übergibt den Inhalt der angegebenen Variable als Ausgangspegel an den angegebenen Analogausgang. Bei dem Sollwert handelt es sich um einen 12 Bit Wert, der also Werte zwischen 0 und 4095 enthalten kann.	103

n Tabelle 5 – Analog I/O

Zum Lesen von Analog-Eingangswerten können Sie einfach die entsprechende Systemvariable (siehe Kapitel 6.12 - Analoge Ein- und Ausgänge ab Seite 224 sowie Kapitel 4.3 - Analog Ein-/ Ausgänge ab Seite 164) auslesen.

## n Variablenbefehle

Befehl	Bedeutung	Seite
ADD_II Zeiger1 M, Zeiger2 M	Der Befehl ADD_II addiert den Inhalt der Variable, die durch Zeiger1 bestimmt wird, mit dem Inhalt der Variable, die durch Zeiger2 bestimmt wird. Das Ergebnis der Addition wird in VARERG gespeichert.	47
ADD_IV Zeiger M, Variable M	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	47
ADD_VA Variable M, 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.	48
ADD_VI Variable M, Zeiger M	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.	48
ADD_VV Variable1 M, Variable2 M	Der Befehl ADD_VV addiert die Inhalte der beiden angegebenen Variablen. Das Ergebnis der Addition wird in der Ergebnisvariable VARERG gespeichert.	49
DEC_V Variable M, Wert K	Der Befehl DEC_V verringert den Inhalt der Variable um den angegebenen Wert Wert. Der Ergebnisspeicher VARERG wird nicht beeinflusst.	54
DEC_VV Variable1 M, Variable2 M	Der Befehl DEC_VV verringert den Inhalt der Variable1 um den Inhalt der Variable2. Der Ergebnisspeicher VARERG wird nicht beeinflusst.	55
DIV_II Zeiger1 M, Zeiger2 M	DIV_II dividiert den Inhalt der durch Zeiger1 bestimmten Variable durch den Inhalt der durch Zeiger2 bestimmten Variable. Das Ergebnis der Division wird in VARERG und DIVREST gespeichert.	58
DIV_IV Zeiger M, Variable M	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.	59
DIV_VA Variable M, 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.	59
DIV_VI Variable M, Zeiger M	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.	60
DIV_VV Variable1 M, Variable2 M	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.	60
LAD_II Zeiger1 M, Zeiger2 M	LAD_II überträgt den Inhalt der durch Zeiger2 bestimmten Variablen in die durch Zeiger1 bestimmte Variable.	75
LAD_IV Zeiger M, Variable M	LAD_IV überträgt den Inhalt der angegebenen Variablen in die durch den Zeiger bestimmte Variable.	75
LAD_VA Variable M, Wert K	LAD_VA lädt den angegebenen Wert in die Variable.	82

Befehl	Bedeutung	Seite
LAD_VI Variable  , Zeiger 	LAD_VI überträgt den Inhalt der durch den Zeiger bestimmten Variable in die angegebene Variable.	82
LAD_VV Variable1  , Variable2 	LAD_VV lädt den Inhalt der Variable2 in die Variable1.	85
MUL_II Zeiger1  , Zeiger2 	MUL_II multipliziert die Inhalte der durch Zeiger1 und Zeiger2 bestimmten Variablen. Das Ergebnis wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL_REST gespeichert.	88
MUL_IV Zeiger  , Variable 	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.	89
MUL_VA Variable  , Wert 	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.	89
MUL_VI Variable  , Zeiger 	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.	90
MUL_VV Variable1  , Variable2 	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.	90
INC_V Variable  , Wert 	INC_V erhöht den Inhalt der Variable den angegebenen Wert. Der Ergebnisspeicher VARERG wird nicht beeinflusst.	70
INC_VV Variable1  , Variable2 	INC_VV erhöht den Inhalt der Variable1 um den in Variable2 gespeicherten Wert. Der Ergebnisspeicher VARERG wird nicht beeinflusst.	71
ODER_II Zeiger1  , Zeiger2 	ODER_II verknüpft die Inhalte der durch die angegebenen Zeiger1 und Zeiger2 bestimmten Variablen mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.	97
ODER_IV Zeiger  , Variable 	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.	98
ODER_VI Variable  , Zeiger 	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.	99
ODER_VA Variable  , Wert 	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.	98
ODER_VV Variable1  , Variable2 	ODER_VV der beiden angegebenen Variablen mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.	99

Befehl	Bedeutung	Seite
SRL_V Variable <input type="checkbox"/> , Wert <input type="checkbox"/>	SRL_V schiebt den Inhalt der Variable um den angegebenen Wert nach rechts. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.	122
SRL_VV Variable1 <input type="checkbox"/> , Variable2 <input type="checkbox"/>	SRL_VV schiebt den Inhalt der Variable1 um den Inhalt der Variable2 nach rechts. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.	123
SLL_V Variable <input type="checkbox"/> , Wert <input type="checkbox"/>	SLL_V 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.	119
SLL_VV Variable1 <input type="checkbox"/> , Variable2 <input type="checkbox"/>	SLL_VV schiebt den Inhalt der Variable1 um den Inhalt der Variable2 nach links. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.	120
SUB_VA Variable <input type="checkbox"/> , Wert <input type="checkbox"/>	SUB_VA subtrahiert vom Inhalt der Variable den angegebenen Wert. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	131
SUB_VV Variable1 <input type="checkbox"/> , Variable2 <input type="checkbox"/>	SUB_VV subtrahiert vom Inhalt der Variable1 den Inhalt der Variable2. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	132
SUB_VI Variable <input type="checkbox"/> , Zeiger <input type="checkbox"/>	SUB_VI subtrahiert vom Inhalt der angegebenen Variable den Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	132
SUB_IV Zeiger <input type="checkbox"/> , Variable <input type="checkbox"/>	SUB_IV subtrahiert vom Inhalt der durch den Zeiger bestimmten Variable den Inhalt der angegebenen Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	131
SUB_II Zeiger1 <input type="checkbox"/> , Zeiger2 <input type="checkbox"/>	SUB_II subtrahiert vom Inhalt der durch Zeiger 1 bestimmten Variable den Inhalt der durch Zeiger2 bestimmten Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.	130
UND_II Zeiger1 <input type="checkbox"/> , Zeiger2 <input type="checkbox"/>	UND_II verknüpft die Inhalte der durch Zeiger1 und Zeiger2 bestimmten Variablen mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.	135
UND_IV Zeiger <input type="checkbox"/> , Variable <input type="checkbox"/>	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.	136
UND_VA Variable <input type="checkbox"/> , Wert <input type="checkbox"/>	UND_VA verknüpft den Inhalt der Variable und den angegebenen Wert mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.	137
UND_VI Variable <input type="checkbox"/> , Zeiger <input type="checkbox"/>	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.	137
UND_VA Variable1 <input type="checkbox"/> , Variable2 <input type="checkbox"/>	UND_VV verknüpft die Inhalte der beiden angegebenen Variablen mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.	138
XOR_II Zeiger1 <input type="checkbox"/> , Zeiger2 <input type="checkbox"/>	XOR_II verknüpft den Inhalt der durch die Zeiger bestimmten Variablen mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.	147
XOR_IV Zeiger <input type="checkbox"/> , Variable <input type="checkbox"/>	XOR_IV verknüpft den Inhalt der durch die Zeiger bestimmten Variable und der angegebenen Variable mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im	148

Befehl	Bedeutung	Seite
	Ergebnisspeicher VARERG abgelegt.	
XOR_VA Variable <input type="checkbox"/> , Wert <input type="checkbox"/>	XOR_VA verknüpft den Inhalt der Variable und den angegebenen Wert mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.	148
XOR_VI Variable <input type="checkbox"/> , Zeiger <input type="checkbox"/>	XOR_VI verknüpft den Inhalt der angegebenen Variable und der durch den Zeiger bestimmten Variable mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.	149
XOR_VV Variable1 <input type="checkbox"/> , Variable2 <input type="checkbox"/>	XOR_VV verknüpft den Inhalt der beiden angegebenen Variablen mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.	149

n Tabelle 6 – Variablenbefehle

## n Variablenvergleichsbefehle

Befehl	Bedeutung	Seite
VERG_II Zeiger1 <input type="checkbox"/> , Zeiger2 <input type="checkbox"/>	VERG_II vergleicht die Inhalte der durch Zeiger1 und Zeiger2 bestimmten Variablen. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	140
VERG_IV Zeiger <input type="checkbox"/> , Variable <input type="checkbox"/>	VERG_IV vergleicht den Inhalt der durch den Zeiger bestimmten Variable mit dem Inhalt der angegebenen Variable. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	141
VERG_VA Variable <input type="checkbox"/> , Wert <input type="checkbox"/>	VERG_VA vergleicht den Inhalt der angegebenen Variable mit dem angegebenen Wert. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	142
VERG_VI Variable <input type="checkbox"/> , Zeiger <input type="checkbox"/>	VERG_VI vergleicht den Inhalt der angegebenen Variable mit dem Inhalt der durch den Zeiger bestimmten Variable. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	143
VERG_VV Variable1 <input type="checkbox"/> , Variable2 <input type="checkbox"/>	VERG_VV vergleicht die Inhalte der beiden angegebenen Variablen. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:	144

n Tabelle 7 – Variablenvergleichsbefehle

## n Programmablaufbefehle

Befehl	Bedeutung	Seite
GEHUPRI Label <input type="checkbox"/>	GEHUPRI ruft ein Unterprogramm ab dem angegebenen Label auf.	67
GEHUPRJ Label <input type="checkbox"/>	GEHUPRJ ruft ein Unterprogramm ab dem angegebenen Label auf. Der Aufruf wird nur ausgeführt, wenn der Bitergebnisspeicher zum Zeitpunkt der Befehlsausführung eingeschaltet ist.	68
GEHUPRN Label <input type="checkbox"/>	GEHUPRN ruft ein Unterprogramm ab dem angegebenen Label auf. Der Aufruf wird nur ausgeführt, wenn der Bitergebnisspeicher zum Zeitpunkt der Befehlsausführung ausgeschaltet ist.	68
GEHUPRV Zieladresse <input type="checkbox"/>	GEHUPRV ruft ein Unterprogramm auf, dessen Startadresse in der angegebenen Variable enthalten ist. Verwenden Sie diesen Befehl um eine dynamische Zyklusprogrammierung zu erstellen.	69
LAD_P1 Label <input type="checkbox"/>	LAD_P1 lädt den Programmzähler des ersten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	78
LAD_P2 Label <input type="checkbox"/>	LAD_P2 lädt den Programmzähler des zweiten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	79
LAD_P3 Label <input type="checkbox"/>	LAD_P3 lädt den Programmzähler des dritten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	80
LAD_P4 Label <input type="checkbox"/>	LAD_P4 lädt den Programmzähler des vierten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.	81
LAD_VL Variable <input type="checkbox"/> , Label <input type="checkbox"/>	LAD_VL überträgt die Programmadresse des Labels in die angegebene Variable. Der Befehl LAD_VL ist notwendig, um ein Unterprogramm mit GEHUPRV anzuspringen.	83
SPRING Label <input type="checkbox"/>	SPRING setzt den Programmablauf ab dem angegebenen Label fort.	121
SPRINGJ Label <input type="checkbox"/>	SPRINGJ setzt den Programmablauf ab dem angegebenen Label fort, wenn der Bitergebnisspeicher zu diesem Zeitpunkt eingeschaltet ist.	121
SPRINGN Label <input type="checkbox"/>	SPRINGN setzt den Programmablauf ab der angegebenen Label fort, wenn der Bitergebnisspeicher zu diesem Zeitpunkt ausgeschaltet ist.	122
UPREND	UPREND beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.	138
UPRENDJ	UPRENDJ beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.	139
UPRENDN	UPRENDN beendet ein Unterprogramm. Das Programm wird an der Stelle fortgesetzt, von der aus das Unterprogramm aufgerufen wurde.	139
WART_A...WART_E	WART_A kann zur Programmierung einfacher Schleifen verwendet werden. Die Schleife wird so lange durchlaufen bis der Bitergebnisspeicher eingeschaltet ist.	145

n Tabelle 8 – Programmablaufbefehle



## n Positionierbefehle

Befehl	Bedeutung	Seite
CHGVEL Achse <input type="checkbox"/> , Geschwindigkeit <input type="checkbox"/>	Mit dem Befehl CHGVEL wird die in der als zweiter Parameter angegebene Variable enthaltene Verfahrensgeschwindigkeit für die angegebene Achse gesetzt. Die neue Geschwindigkeit wird sofort auf den parametrisierten Geschwindigkeitswert geändert.	51
CHGVPO Achse <input type="checkbox"/> , Position <input type="checkbox"/>	Mit dem Befehl CHGVPO zusammen mit SETVEL, kann eine neue Geschwindigkeit in Abhängigkeit von der parametrisierten Position gefahren werden. Der Befehl kann nur verwendet werden, wenn gegenwärtig eine Positionierung aktiv ist, d.h. wenn die Achse bereits gestartet wurde. Die Beschleunigung wird automatisch so errechnet, daß bei der angegebenen Position die neue Geschwindigkeit erreicht ist.	52
CLRERR Achse <input type="checkbox"/>	Mit diesem Befehl wird eine Fehlermeldung bzw. Achsstörung zurückgesetzt. Sie müssen jede Achsstörung zurücksetzen, bevor die entsprechende Achse wieder positioniert werden kann.	53
PRLABS Achse <input type="checkbox"/> , Absolute Position <input type="checkbox"/>	PRLABS lädt eine absolute Zielposition in die angegebene Achse vor. Die Zielposition ist in dem als zweiten angegebenen Parameter enthalten. PRLABS löst noch keinen Achsenstart aus, deshalb kann er auch während einer laufenden Positionierung verwendet werden. Zum Starten der Achsen auf die mit PRLABS vorgeladene Position verwenden Sie bitte STPRLD.	100
PRLREL Achse <input type="checkbox"/> , Relative Position <input type="checkbox"/>	PRLREL lädt eine relative Zielposition in die angegebenen Achse vor. Die Zielposition ist in dem als zweiten angegebenen Parameter enthalten. PRLREL löst noch keinen Achsenstart aus, deshalb kann er auch während einer laufenden Positionierung verwendet werden. Zum Starten der Achsen auf die mit PRLABS vorgeladene Position verwenden Sie bitte STPRLD.	101
PWRDRV Achse <input type="checkbox"/> , Status <input type="checkbox"/>	PWRDRV schaltet das Leistungsteil der angegebenen Achse ein oder aus. Sie müssen immer das Leistungsteil einschalten bevor Sie eine Positionierung starten können. Dies gilt auch für Schrittmotorcontroller.	102
SETFUN Achse <input type="checkbox"/> , Funktion <input type="checkbox"/>	SETFUN aktiviert oder deaktiviert Sonderfunktionen der angegebenen Achse.	112
SETNUL Achse <input type="checkbox"/>	SETNUL setzt die aktuelle Position der angegebenen Achse zu Null.	113
SETOVR Achse <input type="checkbox"/> , Overridewert <input type="checkbox"/>	SETOVR verändert den Override-Wert der angegebenen Achse. Der neue Override ist in der angegebenen Variable enthalten. Mit dem Override kann die Verfahrensgeschwindigkeit der angegebenen Achse verändert werden: der Override gibt den prozentualen Anteil der tatsächlichen Geschwindigkeit im Verhältnis zur programmierten an.	114
SETRMP Achse <input type="checkbox"/> , Beschleunigung <input type="checkbox"/>	SETRMP setzt die Beschleunigungs- und Bremsrampe der angegebenen Achse. Die Rampe wird auf den in der Variable enthaltenen Wert gesetzt. Die Beschleunigungsrampe entspricht danach dem in der Variable enthaltenen Wert, die Bremsrampe wird automatisch gemäß Parametrierung angepasst.	115
SETVEL Achse <input type="checkbox"/> , Geschwindigkeit <input type="checkbox"/>	SETVEL setzt die Sollgeschwindigkeit der angegebenen Achse gesetzt auf den in der angegebenen Variable enthaltenen Wert. Die neue Geschwindigkeit ist beim nächsten Start der Achse aktiv. Die Geschwindigkeit der aktuellen Bewegung wird nicht beeinflusst.	119

Befehl	Bedeutung	Seite
STCONT Achse <input type="checkbox"/> , Richtung <input type="checkbox"/>	STCONT startet die angegebene Achse. Die Bewegung erfolgt kontinuierlich in die angegebene Richtung. Mögliche Richtungsangaben sind PLUS und MINUS.	123
STHOME Achse <input type="checkbox"/>	STHOME startet eine Referenzfahrt für die angegebene Achse. Das System führt hierbei standardmäßig die für Servomotorcontroller übliche Nullung durch.	124
STIABS Achse <input type="checkbox"/> , Absolute Position <input type="checkbox"/>	STIABS bereitet eine linear interpolierte Bewegung für die angegebene Achse vor. Die Zielposition dieser Bewegung wird durch den Wert der angegebenen Variable definiert. Bei STIABS wird – im Gegensatz zu STIREL – die Zielposition als absoluter Wert angegeben.	125
STICIR Achse1 <input type="checkbox"/> , Achse2 <input type="checkbox"/>	STICIR startet eine kreisförmig interpolierte Bewegung der angegebenen Achsen. Für die zirkulare Interpolation werden die Mittelpunkte, die beiden Endpunkte der Achsen auf dem Kreisbogen benötigt sowie die Kreisrichtung als zusätzliche Parameter benötigt. Aufgrund der flexiblen Parametrierung können Sie nicht nur Kreise, sondern auch Ausschnitte des Kreisbogens fahren.	126
STIREL Achse <input type="checkbox"/> , Relative Position <input type="checkbox"/>	STIREL bereitet eine linear interpolierte Bewegung für die angegebene Achse vor. Die Zielposition dieser Bewegung wird durch den Wert der angegebenen Variable definiert. Bei STIREL wird – im Gegensatz zu STIABS – die Zielposition als relativer Wert angegeben.	127
STOPDN Achse <input type="checkbox"/>	STOPDN stoppt die angegebene Achse mit parametrierter Bremsrampe.	128
STOPEM Achse <input type="checkbox"/>	STOPEM stoppt die angegebene Achse mit maximaler Rampe. Diese Funktion wird normalerweise bei einem Notstop verwendet.	128
STPABS Achse <input type="checkbox"/> , Absolute Position <input type="checkbox"/>	STIABS startet die angegebene Achse. Die absolute Zielposition ist in der angegebenen Variable enthalten.	129
STPREL Achse <input type="checkbox"/> , Relative Position <input type="checkbox"/>	STIREL startet die angegebene Achse. Die relative Zielposition ist in der angegebenen Variable enthalten.	129
STPRLD Achse <input type="checkbox"/>	STPRLD startet die angegebene Achse auf die mit PRLABS oder PRLREL vorgeladene Position.	130

n Tabelle 9 – Positionierbefehle

## n Display und Texte

Befehl	Bedeutung	Seite
LAD_DT Formatmaske $\overline{M}$ , Zeile $\overline{K}$ , Spalte $\overline{K}$ , Länge $\overline{K}$	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.	72
LAD_DV Formatmaske $\overline{M}$ , Zeile $\overline{K}$ , Spalte $\overline{K}$ , Länge $\overline{K}$ , Dez. $\overline{K}$ , Vorzeichen $\overline{K}$	LAD_DV legt das Format der folgenden Variableanzeige bzw. des folgenden Editorbefehls durch die Argumente Zeile, 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.	73
SETDSP Display $\overline{K}$ , Funktion $\overline{K}$	SETDSP löst eine Funktion am angegebenen Display aus. Dabei sind verschiedene Funktionen, wie Text- und Variablenanzeige mit verschiedenen Fonts, Anzeige von einzelnen Buchstaben und Umschalten der Anzeigeseite, Anzeige von Bitmaps, Tastaturklick und Signaltöne möglich. Die gewünschte Funktion wird als zweiter Parameter übergeben.	104
SETEDI Display $\overline{K}$ , Funktion $\overline{K}$	SETEDI steuert die Funktion des integrierten Variableneditors. Der Variableneditor ist eine Betriebssystemfunktion, die es erlaubt, über die Tastatur numerische Eingaben vorzunehmen. Mit der ersten Konstante wird das zu verwendende Display gewählt. Mit der zweiten Konstante wird die eigentliche Funktion ausgelöst.	110
TEXT	TEXT deklariert alle folgenden Informationen in der aktuellen Datei bis zum Dateiende als Textdefinition.	133

n Tabelle 10 – Display und Texte

Sie können im Display Texte und Variablen frei positionieren. Die Länge der Textausgabe bzw. das Datenformat der Variablenanzeige können Sie festlegen. Bitte beachten Sie auch die Hinweise im Kapitel 4.2 - Display-Programmierung (Seite 160) sowie in den Kapiteln 6.14 - Displayprogrammierung (Seite 226) und 6.15 - Tastaturen (Seite 227).

## n Datenkonvertierungsbefehle

Befehl	Bedeutung	Seite
LAD_MV Erster Merker $\overline{M}$ , Variable $\overline{M}$	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.	77
LAD_VM Variable $\overline{M}$ , Erster Merker $\overline{M}$	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.	84

n Tabelle 11 – Datenkonvertierungsbefehle

## n Serielle Anbindung

Befehl	Bedeutung	Seite
CLRSER Modulnummer <input type="checkbox"/>	Mit diesem Befehl wird das angegebene serielle Modul neu initialisiert. Alle Send- und Empfangspuffer werden gelöscht, etwaige Fehlermerker werden zurückgesetzt.	53
LAD_DT Formatmaske <input type="checkbox"/> , Zeile <input type="checkbox"/> , Spalte <input type="checkbox"/> , Länge <input type="checkbox"/>	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.	72
LAD_DV Formatmaske <input type="checkbox"/> , Zeile <input type="checkbox"/> , Spalte <input type="checkbox"/> , Länge <input type="checkbox"/> , Dez. <input type="checkbox"/> , Vorzeichen <input type="checkbox"/>	LAD_DV legt das Format der folgenden Variableanzeige bzw. des folgenden Editorbefehls durch die Argumente Zeile, 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.	73
RCVSER Modulnr. <input type="checkbox"/> , Pufferanfang <input type="checkbox"/>	RCVSER überträgt serielle Empfangsdaten von einem seriellen Erweiterungsmodul in ein Variablenfeld. Für jedes empfangene Byte wird eine Variable verwendet. Die erste hierfür zu verwendende Variable wird durch die Konstante angegeben.	103
SETSER Modul <input type="checkbox"/> , Funktion <input type="checkbox"/>	SETSER löst verschiedene Funktionen im Zusammenhang mit dem seriellen Erweiterungsmodul aus. Mit der ersten Konstante wird das zu verwendende Display gewählt. Mit der zweiten Konstante wird die eigentliche Funktion ausgelöst.	116
SNDSER Modul <input type="checkbox"/> , Pufferanfang <input type="checkbox"/>	SNDSER überträgt binäre Daten aus einem Variablenfeld zu einem seriellen Erweiterungsmodul. Für jedes zu sendende Byte wird eine Variable verwendet. Die erste hierfür zu verwendende Variable wird durch die Konstante angegeben. Das erste Byte des Variablenfelds enthält die Anzahl der zu übertragenden Daten in Byte, ab der zweiten Variable beginnen die eigentlichen Sendedaten.	120

n Tabelle 12 – Serielle Anbindung

## n Systembefehle

Einige Befehle gehören in keine der bisher aufgeführten Kategorien. Diese Befehle werden hier unter "Systembefehle" zusammengefasst, da sie einen direkten Einfluß auf das SPS-System und die Parametrierung als solche haben.

Befehl	Bedeutung	Seite
FLASH Funktion <input type="checkbox"/> , Dateinummer <input type="checkbox"/>	FLASH erlaubt den Zugriff auf die integrierte Flash-Diskette der MC200 Steuerung. Der erste Parameter gibt die auszuführende Funktion an, die als zweiter Parameter angegebene Variable enthält die zu bearbeitende Datei.	64
GETPAR Parameter <input type="checkbox"/> , Variable <input type="checkbox"/>	GETPAR überträgt den Wert des angegebenen Parameters in die Variable.	70
SETPAR Parameter <input type="checkbox"/> , Wert <input type="checkbox"/>	SETPAR überträgt den Inhalt der angegebenen Variable in den Parameter.	115

n Tabelle 13 – Systembefehle

## 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.	63
EXITM	EXITM beendet die Makrodefinition in Abhängigkeit einer bedingten Compilierung. Verwenden Sie EXITM, um ein Makro in Abhängigkeit von symbolischen Konstanten abubrechen.	63
Name <input type="checkbox"/> MACRO [Par 1 <input type="checkbox"/> , [Par 2 <input type="checkbox"/> , [...]]]	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.	85

n Tabelle 14 – Makrobefehle

## n Raum für Ihre Notizen

## 3.2 Alphabetische Befehlsübersicht

### n ADD\_II

ADD\_II Zeiger1  $\bar{M}$ , Zeiger2  $\bar{M}$

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

Der Befehl ADD\_II addiert den Inhalt der Variable, die durch Zeiger1 bestimmt wird, mit dem Inhalt der Variable, die durch Zeiger2 bestimmt wird. Das Ergebnis der Addition wird in VARERG gespeichert.

#### Operation

(VARERG)  $\zeta$  (Zeiger1  $\bar{a}$  Variable) + (Zeiger 2  $\bar{a}$  Variable)

#### Beispiel

```
LAD_VA    V_ZEIGER1, 200    // V_ZEIGER1 zeigt auf Variable 200
LAD_VA    V_ZEIGER2, 400    // V_ZEIGER2 zeigt auf Variable 400
ADD_II    V_ZEIGER1, V_ZEIGER2 // Addiert den Inhalt der Variable 200 zu dem
// Inhalt der Variable 400 und speichert das
// Ergebnis in der Variable VARERG
```

#### Hinweise

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

#### Siehe auch

Variablenbefehle (Seite 36)

### n ADD\_IV

ADD\_IV Zeiger  $\bar{M}$ , Variable  $\bar{M}$

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)  $\zeta$  (Zeiger  $\bar{a}$  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 36)

**n ADD\_VA**ADD\_VA Variable  $\mathbb{M}$ , Wert  $\mathbb{K}$ 

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)  $\Leftarrow$  (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 36)

**n ADD\_VI**ADD\_VI Variable  $\mathbb{M}$ , Zeiger  $\mathbb{M}$ 

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)  $\Leftarrow$  (Variable) + (Zeiger  $\rightarrow$  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 36)



**n ADD\_VV**ADD\_VV Variable1 , Variable2 

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)  $\Leftarrow$  (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 36)

**n AUS\_A**AUS\_A Ausgang 

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

Der Befehl AUS\_A schaltet den angegebenen Ausgang aus.

**Operation**(Ausgang)  $\Leftarrow$  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 34)

**n AUS\_AI**AUS\_AI Zeiger 

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 → Ausgang)  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_V     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 34)

**n AUS\_M**AUS\_M Merker 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	- Ja	n Nein

Der Befehl AUS\_M löscht den angegebenen Merker.

**Operation**(Merker)  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 33)

## n AUS\_MI

AUS\_MI Zeiger

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 → Merker)  $\bar{C}$  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 33)

## n CHGVEL

CHGVEL Achse , Geschwindigkeit

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

Mit dem Befehl CHGVEL wird die in der als zweiter Parameter angegebenen Variable enthaltene Verfahrgeschwindigkeit für die angegebene Achse gesetzt. Die neue Geschwindigkeit wird sofort auf den parametrisierten Geschwindigkeitswert geändert.

### Beispiel

```
// Die Achse 1 soll auf die neue Geschwindigkeit von 5000 beschleunigen, wenn der
// Eingang 1 (E_START) aktiv wird.

LAD_VA    V_GESW_POS, 2000      // Laden der Geschwindigkeit von 2000
// in Variable GESW_POS
LAD_VA    V_GESW_A1, 5000       // Laden der Geschwindigkeit von 5000
// in Variable GESW_A1
STPABS    1, V_SOLLP_A1         // Starte die absolute Positionierung der Achse 1
// die in V_SOLLP_A1 enthaltenen Position

LOOP:
LAD_E     E_START               // Abfrage: Eingang E-START eingeschaltet?
CHGVEL    1, V_GESCHW           // Geschwindigkeitswechsel auf die in der
// V_GESCHW enthaltene Geschwindigkeit

SPRINGN   LOOP
```

### Hinweise

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

n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)  
 SETVEL (Seite 119)  
 CHGVPO (Seite 52)  
 SETOVR (Seite 114)  
 PWRDRV (Seite 102)

## n CHGVPO

CHGVPO Achse , Position

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

Mit dem Befehl CHGVPO zusammen mit SETVEL, kann eine neue Geschwindigkeit in Abhängigkeit von der parametrisierten Position gefahren werden. Der Befehl kann nur verwendet werden, wenn gegenwärtig eine Positionierung aktiv ist, d.h. wenn die Achse bereits gestartet wurde. Die Beschleunigung wird automatisch so errechnet, daß bei der angegebenen Position die neue Geschwindigkeit erreicht ist.

### Beispiel

```
// Die Achse 1 soll bei der Position 250 einen Geschwindigkeitswechsel auf die neue
// Geschwindigkeit von 5000 durchführen.

LAD_VA      V_GESW_A1, 3000      // Laden der Startgeschwindigkeit
SETVEL      1, V_GESW_A1        // Setzen der Startgeschwindigkeit
LAD_VA      V_SOLLP_A1, 500      // Laden der Sollposition für die Positionierung
STPABS      1, V_SOLLP_A1        // Starte die absolute Positionierung der Achse 1
// mit dem Wert, der in der Variablen SOLLP_A1
// enthalten ist

LAD_VA      V_GESW_POS, 5000     // Laden der neuen Geschwindigkeit
SETVEL      1, V_GESW_POS        // Programmieren neue Geschwindigkeit für Achse 1
// Normalerweise die neue Geschwindigkeit erst
// bei der nächsten Positionierung wirksam

LAD_VA      V_POS_A1, 250        // Laden der Position, an der die Geschwindigkeit
// umgeschaltet werden soll

CHGVPO      1, V_POS_A1          // Programmieren des Geschwindigkeitswechsels
// Jetzt wird der Geschwindigkeitswechsel von
// 3000 auf 5000 automatisch durchgeführt, sodaß
// die neue Geschwindigkeit an der übergebenen
// Position 250 erreicht ist.

LOOP:
LAD_M       M_INP_A1             // Abfrage: Achse 1 in Position?
SPRINGN     LOOP                // Warten, bis Achse 1 in Position ist
```

### Hinweise

- n Dieser Befehl wird nur in der MOC implementiert und nicht in der SM2, bitte beachten!
- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)  
 SETVEL (Seite 119)  
 CHGVEL (Seite 51)  
 PWRDRV (Seite 102)  
 SETOVR (Seite 114)

## n CLRERR

CLRERR Achse 

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

Mit diesem Befehl wird eine Fehlermeldung bzw. Achsstörung zurückgesetzt. Sie müssen jede Achsstörung zurückzusetzen, bevor die entsprechende Achse wieder positioniert werden kann.

### Beispiel

//Die Fehlermeldung der Achse 1 soll zurückgesetzt werden.

```
CLRERR      1          // Fehlermeldungen der Achse 1 zurück setzen
PWRDRV     1, ON      // Leistungsteil wieder freischalten
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)

## n CLRSER

CLRSER Modulnummer 

Gruppe	Abhängig von BES	Verändert BES
Seriellmodul	- Ja	n Nein

Mit diesem Befehl wird das angegebene serielle Modul neu initialisiert. Alle Sende- und Empfangspuffer werden gelöscht, etwaige Fehlermerker werden zurückgesetzt.

### Beispiel

```
CLRSER     SER1      // Serielles Erweiterungsmodul 1 zurücksetzen
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Sämtliche Systemmerker und –variablen werden bei Ausführung des Befehls CLRSER auf den Ursprungsstand nach dem Einschalten der Steuerung gesetzt.

### Siehe auch

Serielle Anbindung (Seite 44)  
 Serielles Modul (Seite 171)

**n DEC\_V**DEC\_V Variable  $\boxed{V}$ , Wert  $\boxed{K}$ 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

Der Befehl DEC\_V verringert den Inhalt der Variable um den angegebenen 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:

n M\_GLEICH wird gesetzt, wenn nach der Operation die Variable1 den Wert "0" enthält

n M\_GROESSER wird gesetzt, wenn nach der Operation die Variable1 größer als "0" ist

n M\_KLEINER wird gesetzt, wenn nach der Operation die Variable1 kleiner als "0" ist.

**Operation**

(Variable1)  $\ominus$  (Variable) - Wert

(M\_GLEICH)  $\ominus$  (Ergebnis = 0)

(M\_GROESSER)  $\ominus$  (Ergebnis > 0)

(M\_KLEINER)  $\ominus$  (Ergebnis < 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**

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

**Siehe auch**

Variablenbefehle (Seite 36)

## n DEC\_VV

DEC\_VV Variable1 , Variable2

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

Der Befehl DEC\_VV verringert den Inhalt der Variable1 um den Inhalt der Variable2. 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:

n M\_GLEICH wird gesetzt, wenn nach der Operation die Variable1 den Wert "0" enthält

n M\_GROESSER wird gesetzt, wenn nach der Operation die Variable1 größer als "0" ist

n M\_KLEINER wird gesetzt, wenn nach der Operation die Variable1 kleiner als "0" ist.

### Operation

(Variable1)  $\ominus$  (Variable1) - (Variable2)

(M\_GLEICH)  $\ominus$  (Ergebnis = 0)

(M\_GROESSER)  $\ominus$  (Ergebnis > 0)

(M\_KLEINER)  $\ominus$  (Ergebnis < 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_VV, 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
LAD_VA     V_STEP, 1 // Zahl der Inkremente
```

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

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)

**n DEF\_A**DEF\_A Ausgangsnr. , Symbolname 

Gruppe	Abhängig von BES	Verändert BES
Definitionen	<b>y</b> Nein	<b>n</b> 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-1B 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 32)

**n DEF\_E**DEF\_E Eingangsnr. , Symbolname 

Gruppe	Abhängig von BES	Verändert BES
Definitionen	<b>y</b> Nein	<b>n</b> 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-1B 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 32)



## n DEF\_M

DEF\_M Merckernr. , Symbolname

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-1B 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 32)

## n DEF\_V

DEF\_V Variablennr. , Symbolname

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-1B 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 32)

**n DEF\_W**DEF\_W Wert  $\mathbb{K}$ , Symbolname  $\mathbb{K}$ 

Gruppe	Abhängig von BES	Verändert BES
Definitionen	<b>y</b> Nein	<b>n</b> 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-1B Quelltextes vornehmen. Bitte beachten Sie aber, daß Definitionen innerhalb des Quelltextes nur für die entsprechende Quelltextdatei, nicht aber für das gesamte Projekt gelten.

**Siehe auch**

Definitionsbefehle (Seite 32)

**n DIV\_II**DIV\_II Zeiger1  $\mathbb{V}$ , Zeiger2  $\mathbb{V}$ 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	<b>n</b> Nein

DIV\_II dividiert den Inhalt der durch Zeiger1 bestimmten Variable durch den Inhalt der durch Zeiger2 bestimmten Variable. Das Ergebnis der Division wird in VARERG und DIVREST gespeichert.

**Operation**

```
(VARERG)   $\mathbb{C}$  (Zeiger1  $\mathbb{a}$  Variable) / (Zeiger 2  $\mathbb{a}$  Variable)
(DIVREST)  $\mathbb{C}$  (Zeiger1  $\mathbb{a}$  Variable) % (Zeiger2  $\mathbb{a}$  Variable)
```

**Beispiel**

```
{ DIV_II    V_ZEIGER1, V_ZEIGER2 // Dividiert den Inhalt der Variable auf die
// V_ZEIGER1 zeigt, durch den Inhalt der
// Variable, auf die V_ZEIGER2 zeigt.
// Das Ergebnis wird in VARERG
// und DIVREST gespeichert
```

**Hinweise**

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

**Siehe auch**

Variablenbefehle (Seite 36)

## n DIV\_IV

DIV\_IV Zeiger  $\overline{M}$ , Variable  $\overline{M}$

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)  $\zeta$  (Zeiger  $\tilde{a}$  Variable) / (Variable)

(DIVREST)  $\zeta$  (Zeiger  $\tilde{a}$  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 36)

## n DIV\_VA

DIV\_VA Variable  $\overline{M}$ , Wert  $\overline{K}$

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)  $\zeta$  (Variable) / Wert

(DIVREST)  $\zeta$  (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 36)

## n DIV\_VI

DIV\_VI Variable , Zeiger 

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)  $\zeta$  (Variable) / (Zeiger  $\hat{a}$  Variable)  
 (DIVREST)  $\zeta$  (Variable) % (Zeiger  $\hat{a}$  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 36)

## n DIV\_VV

DIV\_VV Variable1 , Variable2 

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)  $\zeta$  (Variable1) / (Variable2)  
 (DIVREST)  $\zeta$  (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 36)

## n EIN\_A

EIN\_A Ausgang  $\bar{A}$

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

EIN\_A schaltet den angegebenen Ausgang ein.

### Operation

(Ausgang)  $\bar{C}$  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 34)

## n EIN\_AI

EIN\_AI Zeiger  $\bar{V}$

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  $\rightarrow$  Ausgang)  $\bar{C}$  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

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

n Verwenden Sie EIN\_AI, um einen Ausgang zu setzen, dessen Nummer bei der Programmerstellung noch nicht feststeht.

### Siehe auch


Ein-/Ausgangsbefehle (Seite 34)

**n EIN\_M****EIN\_M Merker** 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	- Ja	<b>n</b> Nein

EIN\_M setzt den angegebenen Merker.

**Operation**

(Merker)  EIN

**Beispiel**

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

**Hinweise**

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

**Siehe auch**

Merkerbefehle (Seite 33)

**n EIN\_MI****EIN\_MI Zeiger** 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	- Ja	<b>n</b> Nein

EIN\_MI setzt den durch den angegebenen Zeiger bestimmten Merker.

**Operation**

(Zeiger  Merker)  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 33)

**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 keine Parameter benötigt.
```

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

**Siehe auch**

Makros (Seite 45)

**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 eine Referenzfahrt für alle im System vorhandenen
// Achsen starten. Die Anzahl der Achsen wurde zuvor mit der Konstante
// Achsenzahl festgelegt.
```

```
DEF_W     2, Achsenzahl        // Hier gehen wir davon aus: 2 Achsen im System

REFDRIVE  MACRO                // Makro REFDRIVE definieren
STHOME    1, 0                // Referenzfahrt für Achse 1 starten
#IF Achsenzahl EQ 1          // Wenn nur eine Achse vorhanden ist...
EXITM      // Makro abbrechen
#ENDIF

STHOME    2, 0                // Referenzfahrt für Achse 2 starten
#IF Achsenzahl EQ 2          // Wenn zwei Achsen vorhanden sind...
#ENDIF

// Ansonsten gehen wir davon aus, daß wir
// drei Achsen haben
STHOME    3, 0                // Referenzfahrt für Achse 3 starten
ENDM      // Ende der Makrodefinition
```

**Siehe auch**

Makros (Seite 45)

## n FLASH

FLASH Funktion , Dateinummer

Gruppe	Abhängig von BES	Verändert BES
Systembefehle	- Ja	n Nein

FLASH erlaubt den Zugriff auf die integrierte Flash-Diskette der MC200 Steuerung. Der erste Parameter gibt die auszuführende Funktion an, die als zweiter Parameter angegebene Variable enthält die zu bearbeitende Datei.

Sie können mit dem Befehl Variablenbereiche innerhalb des ausfallsicheren Flash speichern, sie zurückladen oder löschen. Weiterhin ist es mit dem Befehl FLASH möglich, das aktuelle SPS-Programm und die aktuellen System- und Achsparameter zu speichern.

### Mögliche Werte für den ersten Parameter (Funktionsnummer):

Name	Wert	Bedeutung
LIST	1	Aufrufen von Informationen zu einer Datei im Flash.
READ	2	Übertragen einer Datei aus dem Flash in den Variablenspeicher.
WRITE	3	Übertragen eines Ausschnitts des Variablenspeichers ins Flash.
DELETE	4	Löschen einer Datei im Flash.
DESTROY	5	Löschen aller Dateien im Flash.
PLC_SAVE	14	Speichern des SPS-Programmes im Flash.
PAR_SAVE	15	Speichern der System- und Achsparameter im Flash.

n Tabelle 15 – Funktionen für FLASH

### Mögliche Werte für den zweiten Parameter (Dateinummer)

Der zweite Parameter enthält als Variablenwert die zu verwendende Dateinummer. Gegenwärtig unterstützt das MC200 System 16 voneinander unabhängige Dateien. Diese Dateien können hierbei unterschiedlicher Länge sein.

Einige Funktionen verwenden die Dateinummer in abgeänderter Form oder ignorieren diese Angabe sogar. Bitte beachten Sie hierzu die Ausführungen auf den nächsten Seiten. Grundsätzlich gilt: Wird als Dateinummer ein Wert größer als 16 angegeben, so wird die Funktion nicht ausgeführt.

### Systemvariablen für die Flash-Diskette

Im Zusammenhang mit der Funktion FLASH werden einige Systemvariablen verwendet:

Name	Nummer	Bedeutung
V_FLASH_FREE	57	Enthält den freien Speicher der Flash-Diskette in "Anzahl Variablen"
V_FLASH_COUNT	58	Enthält die Anzahl der zu übertragenden Variablen
V_FLASH_START	59	Enthält die Nummer der ersten zu übertragende Variable
V_FLASH_NAME	60	Enthält eine beliebige Zahl als Dateiname

n Tabelle 16 – Systemvariablen für FLASH

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Als zweiter Parameter dürfen nur Variablen bis zur Variablennummer 255 verwendet werden.

### Siehe auch

Systembefehle (Seite 45)



## n FLASH LIST

Mit der Funktion LIST ermitteln Sie Informationen über die in der Flash-Diskette gespeicherten. Als zweiten Parameter ist hierbei die gewünschte Dateinummer anzugeben.

Die Informationen zur jeweiligen Datei werden in den Systemvariablen zur Flash-Diskette zurückgegeben:

n V\_FLASH\_START enthält die ursprüngliche Variablennummer der ersten Variable aus dieser Datei

n V\_FLASH\_COUNT enthält die Anzahl der Variablen in dieser Datei (0 = leere Datei)

n V\_FLASH\_NAME enthält den Dateinamen

### Beispiel

```
// In diesem Beispiel durchsuchen wir die Flash-Diskette nach einer Datei mit dem
// Namen "991004".
```

```
LAD_VA    V_DATEI, 1           // Beginn mit Datei #1

Loop:
FLASH     LIST, V_DATEI       // Informationen zur Datei holen
VERG_VA   V_FLASH_COUNT, 0    // Leere Datei?
NLAD_M    M_GLEICH           // Vergleichsergebnis prüfen
SPRINGN   Weiter            // Ja, überspringen

VERG_VA   V_FLASH_NAME, 991004 // Gesuchte Datei?
NLAD_M    M_GLEICH           // Vergleichsergebnis prüfen
SPRINGN   Weiter            // Nein, überspringen

LAD_VV    VARERG, V_DATEI     // Dateinummer in Ergebnisvariable speichern
UPREND

Weiter:
INC_V     V_DATEI, 1         // Nächste Datei prüfen
VERG_VA   V_DATEI, 16        // Alle Dateien geprüft?
LAD_M     M_GROESSER         // Vergleichsergebnis prüfen
SPRINGN   Loop              // Nein, weitersuchen

LAD_VA    VARERG, 0          // 0 in Ergebnisvariable: Datei nicht gefunden
UPREND
```

## n FLASH READ

Mit FLASH READ wird eine bestimmte Datei aus der Flash-Diskette zurück in den Variablenspeicher gelesen. Als zweiter Parameter ist hierbei die zu lesende Dateinummer anzugeben.

Es ist nicht zwingend notwendig, die Datei in die gleichen Variablen zu lesen, aus denen Sie gespeichert wurde. Auch die Anzahl der zu lesenden Variablen kann angegeben werden. Die Parameter werden über die Systemvariablen zur Flash-Diskette übergeben:

n V\_FLASH\_START enthält die erste Variable, die mit der Datei überschrieben werden soll

n V\_FLASH\_COUNT enthält die Anzahl der zu lesenden Variablen

### Beispiel

```
LAD_VA    V_DATEI, 10        // Datei Nummer 10 lesen
LAD_VA    V_FLASH_START, 4320 // in Variablen ab 4320 speichern
LAD_VA    V_FLASH_COUNT, 17  // 17 Variablen lesen
FLASH     READ, V_DATEI      // ausführen
```

## n FLASH WRITE

Mit FLASH WRITE werden Variablen als Datei auf der Flash-Diskette gespeichert. Als zweiter Parameter wird hierbei die zu schreibende Dateinummer angegeben. Die weiteren Parameter werden über die Systemvariablen zur Flash-Diskette übergeben:

- n V\_FLASH\_START enthält die erste zu schreibende Variable
- n V\_FLASH\_COUNT enthält die Anzahl der zu schreibenden Variablen
- n V\_FLASH\_NAME enthält den Dateinamen der neuen Datei

### Beispiel

```
LAD_VA    V_DATEI, 10           // Datei Nummer 10 schreiben
LAD_VA    V_FLASH_START, 1900  // Variablen ab 1900 auf Flash-Diskette speichern
LAD_VA    V_FLASH_COUNT, 170   // 170 Variablen schreiben
```

```
// Achtung! Es sollte vor jeder Schreiboperation geprüft werden, ob auf der Flash-
// Diskette genügend Platz für die zu schreibenden Daten ist! Falls nicht genügend
// Speicher vorhanden ist, bricht die Funktion FLASH WRITE ohne Fehlermeldung ab.
```

```
VERG_VV   V_FLASH_COUNT, V_FLASH_FREE
NLAD_M    M_GROESSER           // genügend freier Speicher vorhanden?
FLASH     WRITE, V_DATEI       // dann ausführen
```

### Hinweise

- n Sie können mit der Funktion FLASH WRITE entweder neue Dateien anlegen oder vorhandene Dateien überschreiben. Falls die zu schreibende Datei bereits vorhanden ist, wird sie vom System automatisch gelöscht und statt dessen die neue Datei aufgezeichnet.
- n Bitte beachten Sie, daß der Hersteller der Flash-Speichermedien eine Lebensdauer von 100.000 Schreibzyklen garantiert. Dies bedeutet, daß die Haltbarkeit des Speichers bei normaler Anwendung nahezu unbegrenzt ist. Vermeiden Sie dennoch unnötige, sich stets wiederholende Schreiboperationen!

## n FLASH DELETE

Mit der Funktion FLASH DELETE wird eine Datei von der Flash-Diskette gelöscht. Als zweiter Parameter wird hierbei die zu schreibende Dateinummer angegeben. Sämtliche Daten dieser Datei gehen bei der Ausführung des Befehls verloren. Weitere Parameter werden nicht benötigt.

### Beispiel

```
LAD_VA    V_DATEI, 13           // Datei Nummer 13 löschen
FLASH     DELETE, V_DATEI       // ausführen
```

## n FLASH DESTROY

Mit der Funktion FLASH DESTROY werden sämtliche Dateien auf der Flash-Diskette gelöscht. Die als zweiter Parameter angegebene Dateinummer wird ignoriert. Sonstige im Flash gespeicherten Daten, wie z.B. das SPS-Programm oder die Systemparameter, werden von dem Befehl nicht beeinflusst.

### Beispiel

```
FLASH     DESTROY, VARERG       // Alle Dateien der Flash-Diskette löschen
```

## n FLASH PLCSAVE

Mit der Funktion FLASH PLCSAVE wird das aktuelle SPS-Programm aus dem RAM in die Flash-Diskette übertragen. Es wird damit zu dem Standardprogramm, welches bei einer Umlöschung mit dem Resettaster in den Speicher geladen wird. Die als zweiter Parameter angegebene Dateinummer wird ignoriert. Weitere Parameter werden nicht benötigt.

### Beispiel

```
FLASH      PLCSAVE, VARERG      // Aktuelles SPS-Programm im Flash speichern
```

### Hinweise

- n Nach Ausführung dieser Funktion wird automatisch ein Steuerungsneustart (Reset) durchgeführt.
- n Diese Funktion entspricht dem Speichern des SPS-Programms im Flash mit Hilfe der VMC Workbench Entwicklungsoberfläche.

## n FLASH PARSAVE

Mit der Funktion FLASH PARSAVE speichern Sie den aktuellen Satz von System- und Achsparametern im Flash. Das bedeutet, daß in Zukunft diese Parameter nach einem Steuerungsneustart (Reset) geladen werden. Die als zweiter Parameter angegebene Dateinummer wird ignoriert. Weitere Parameter werden nicht benötigt.

### Beispiel

```
FLASH      PARSAVE, VARERG      // Aktuelles Parameter im Flash speichern
```

### Hinweise

- n Diese Funktion entspricht dem Speichern der Parameter im Flash mit Hilfe der VMC Workbench Entwicklungsoberfläche.





## n GEHUPRI

GEHUPRI Label 

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	y Nein	Siehe Text

GEHUPRI ruft ein Unterprogramm ab dem angegebenen Label auf.

### Operation

(Stackpointer)  Programmadresse  
 Stackpointer  Stackpointer - 2  
 Programmadresse  Label - Adresse  
 (Bitergebnis)  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 40)

**n GEHUPRJ**GEHUPRJ Label 

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

**Beispiel**

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

**Hinweise**

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

**Siehe auch**





Programmablaufbefehle (Seite 40)

**n GEHUPRN**GEHUPRN Label 

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	- Ja	Siehe Text

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

**Operation**

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

**Beispiel**

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

**Hinweise**

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

**Siehe auch**

Programmablaufbefehle (Seite 40)

## n GEHUPRV

GEHUPRV Zieladresse

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)  $\zeta$  Programmadresse  
 Stackpointer  $\zeta$  Stackpointer - 2  
 Programmadresse  $\zeta$  (Zieladresse)  
 (Bitergebnis)  $\zeta$  EIN

### Beispiel

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

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

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

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

### Hinweise

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

### Siehe auch

Programmablaufbefehle (Seite 40)

## n GETPAR

GETPAR Parameter  $\square$ , Variable  $\square$

Gruppe	Abhängig von BES	Verändert BES
Systembefehle	- Ja	n Nein

GETPAR überträgt den Wert des angegebenen Parameters in die Variable.

### Operation

(Variable)  $\Leftarrow$  (Parameter)

### Beispiel

```
// Über das SPS-Programm soll aus der Variablenresultatenspeicher ein Parameter
// gelesen.
```

```
GETPAR 113, VARERG // aktuelle Null-Punktverschiebung lesen
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Für Achsparameter errechnet sich die Parameternummer aus der Basisparameternummer und der jeweils betroffenen Achse. Die entsprechende Berechnungsformel lautet:  
 $\Rightarrow$  (Nummer der Achse \* 100) + Basisparameternummer

### Siehe auch

Systembefehle (Seite 45)  
 SETPAR (Seite 115)  
 System- und Achsparameter (Seite 177)

## n INC\_V

INC\_V Variable  $\square$ , Wert  $\square$

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)  $\Leftarrow$  (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 // Zeigervariablen 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 36)

## n INC\_VV

INC\_VV Variable1  $\bar{V}$ , Variable2  $\bar{V}$

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

INC\_VV erhöht den Inhalt der Variable1 um den in Variable2 gespeicherten Wert. Der Ergebnisspeicher VARERG wird nicht beeinflusst.

### Operation

(Variable1)  $\oplus$  (Variable1) + (Variable2)

### Beispiel

```
// Hier bearbeiten wir ein Variablenfeld und setzen in jeder Variable die unteren
// 16 Bit. Mit INC_VV 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
LAD_VA    V_STEP, 1                // Zahl der Inkremente
LOOP:
ODER_VI   V_MASKE, V_ZEIGER        // Verknüpfung durchführen
LAD_IV    V_ZEIGER, VARERG         // Ergebnis der Verknüpfung wieder in Tabelle
INC_VV    V_ZEIGER, V_STEP         // 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 36)

## n LAD\_A

LAD\_A Ausgang  $\bar{A}$

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

LAD\_A lädt den Zustand des angegebenen Ausgangs in den Bitergebnisspeicher.

### Operation

(Bitergebnis)  $\oplus$  (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 34)

**n LAD\_AI**LAD\_AI Zeiger  $\bar{V}$ 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>b</b> Ja

LAD\_AI lädt den Zustand des durch den Zeiger bestimmten Ausgangs in den Bitergebnisspeicher.

**Operation**

(Bitergebnis)  $\bar{C}$  (Zeiger  $\bar{a}$  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 34)

**n LAD\_DT**

LAD\_DT Formatmaske  $\bar{V}$ , Zeile  $\bar{K}$ ,  
Spalte  $\bar{K}$ , Länge  $\bar{K}$

Gruppe	Abhängig von BES	Verändert BES
Display/Tastatur	- Ja	<b>n</b> 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 17 – 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**

Display und Texte (Seite 43)  
LAD\_DV (Seite 73)  
SETDSP (Seite 104)  
SETEDI (Seite 110)



## n LAD\_DV

LAD\_DV Formatmaske , Zeile ,  
Spalte , Länge , Dez. , Vorzeichen

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

LAD\_DV legt das Format der folgenden Variableanzeige bzw. des folgenden Editorbefehls durch die Argumente Zeile, 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 18 – 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


Display und Texte (Seite 43)  
LAD\_DT (Seite 72)  
SETDSP (Seite 104)  
SETEDI (Seite 110)

**n LAD\_E**LAD\_E Eingang 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> Ja

LAD\_E lädt den Zustand des angegebenen Eingangs in den Bitergebnisspeicher.

**Operation**

(Bitergebnis)  (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 34)

**n LAD\_EI**LAD\_EI Zeiger 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> Ja

LAD\_EI lädt den Zustand des durch den Zeiger angegebenen Eingangs in den Bitergebnisspeicher.

**Operation**

(Bitergebnis)  (Zeiger → 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 34)

## n LAD\_II

LAD\_II Zeiger1  $\overline{M}$ , Zeiger2  $\overline{M}$

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

LAD\_II überträgt den Inhalt der durch Zeiger2 bestimmten Variablen in die durch Zeiger1 bestimmte Variable.

### Operation

(Zeiger1  $\rightarrow$  Variable)  $\Leftarrow$  (Zeiger2  $\rightarrow$  Variable)

### Beispiel

```
// In diesem Beispiel kopieren wir 99 Einträge einer Tabelle, die bei
// Variable 200 beginnt, in eine Tabelle, die bei Variable 400 beginnt.

LAD_VA    V_ZEIGER1, 200          // V_ZEIGER1 mit Originaltabelle laden
LAD_VA    V_ZEIGER2, 400         // V_ZEIGER2 mit Zieltabelle laden
LAD_VA    V_ZAEHLER, 99          // Wir wollen 99 Variablen kopieren

LOOP:
LAD_II    V_ZEIGER2, V_ZEIGER1   // Variable in zweite Tabelle kopieren
INC_V    V_ZEIGER1, 1            // Zeiger auf Originaltabelle erhöhen
INC_V    V_ZEIGER2, 1            // Zeiger auf Zieltabelle erhöhen
DEC_V    V_ZAEHLER, 1            // Zähler für Kopieren verringern
LAD_M    M_GLEICH                 // Bereits alles kopiert?
SPRNGN   LOOP                     // Noch nicht alles kopiert, weiter
```

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)

## n LAD\_IV

LAD\_IV Zeiger  $\overline{M}$ , Variable  $\overline{M}$

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  $\rightarrow$  Variable)  $\Leftarrow$  (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 36)

**n LAD\_M**LAD\_M Merker 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	<b>y</b> Nein	<b>p</b> Ja

LAD\_M lädt den Zustand des angegebenen Merkers in den Bitergebnisspeicher.

**Operation**

(Bitergebnis)  (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 33)

**n LAD\_MI**LAD\_MI Zeiger 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	<b>y</b> Nein	<b>p</b> Ja

LAD\_MI überträgt den Zustand des durch den angegebenen Zeiger bestimmten Merkers in den Bitergebnisspeicher.

**Operation**

(Bitergebnis)  (Zeiger → 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 33)

## n LAD\_MV

LAD\_MV Erster Merker  $\overline{M}$ , Variable  $\overline{V}$ 

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 19 – LAD\_MV Übersicht

## Operation

```
(Erster Merker + 0)   $\zeta$  (Variable & 0x00000001)
(Erster Merker + 1)   $\zeta$  (Variable & 0x00000002)
(Erster Merker + 2)   $\zeta$  (Variable & 0x00000004)
...
(Erster Merker + 31)  $\zeta$  (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  // überträgt 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 43)

LAD\_VM (Seite 84)

## n LAD\_P1

Task starten: LAD\_P1 Label 

Task beenden: LAD\_P1 TASK\_STOP 

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	- Ja	Siehe Text

LAD\_P1 lädt den Programmzähler des ersten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.

### Beispiel

```
LAD_VA    V_TASK_1_P, 2           // Weist Task1 die Taskpriorität 2 zu
LAD_P1    LABEL                  // Startet die Task1
LAD_P1    TASK_STOP              // Beendet die Task1
```

### Task beenden

Mit LAD\_P1 TASK\_STOP beenden Sie die Task 1. War die Task des entsprechenden Parallelprogramms bereits aktiv, wird der Unterprogrammstart dieses Parallelprogramms gelöscht, d.h. alle Unterprogramme werden beendet und das laufende Programm wird auf die angegebene Adresse verzweigt. Verwenden Sie den Befehl LAD\_P1 TASK\_STOP mit äußerster Vorsicht: Falls Sie mit LAD\_P1 TASK\_STOP das aktuelle Parallelprogramm beenden und keine weiteren Parallelprogramme aktiv sind, dann bleibt das System stehen.

### Prioritäten festlegen

Mit Hilfe der Systemvariablen V\_TASK\_1\_P weisen Sie der Task 1 die gewünschte Priorität zu. Sie können den Tasks unterschiedliche Prioritäten zuweisen. Beim Steuerungsreset werden die Prioritäten der Tasks beim Systemstart auf 1 gesetzt, d.h. alle Tasks laufen mit der gleichen Priorität.

Je höher die Prioritätszahl für die Task ist, desto langsamer läuft die entsprechende Task. Ein Wert von 2 bedeutet, daß die CPU zwei Befehle aus den anderen Tasks in der gleichen Zeit ausführt wie einen Befehl aus der entsprechend verlangsamten Task. Der Wertebereich für die Taskpriorität liegt zwischen 1 und 255.

### Bitergebnisspeicher (BES)

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



### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Sie können auch innerhalb der Task mit dem LAD\_P1-Befehl die Ausführungsposition verändern.
- n 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 40)  
 LAD\_P2 (Seite 79)  
 LAD\_P3 (Seite 80)  
 LAD\_P4 (Seite 81)

## n LAD\_P2

Task starten: LAD\_P2 LABEL   
 Task beenden: LAD\_P2 TASK\_STOP 

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	- Ja	Siehe Text

LAD\_P2 lädt den Programmzähler des zweiten Parallelprogramms mit dem angegebenen Label. Das Parallelprogramm wird sofort ab dem angegebenen Label gestartet.

### Beispiel

```
LAD_VA    V_TASK_2_P, 2           // Weist Task2 die Taskpriorität 2 zu
LAD_P2    LABEL                  // Startet die Task2
LAD_P2    TASK_STOP              // Beendet die Task2
```

### Task beenden

Mit LAD\_P2 TASK\_STOP beenden Sie die Task 2. War die Task des entsprechenden Parallelprogramms bereits aktiv, wird der Unterprogrammstart dieses Parallelprogramms gelöscht, d.h. alle Unterprogramme werden beendet und das laufende Programm wird auf die angegebene Adresse verzweigt. Verwenden Sie den Befehl LAD\_P2 TASK\_STOP mit äußerster Vorsicht: Falls Sie mit LAD\_P2 TASK\_STOP das aktuelle Parallelprogramm beenden und keine weiteren Parallelprogramme aktiv sind, dann bleibt das System stehen.

### Prioritäten festlegen

Mit Hilfe der Systemvariablen V\_TASK\_2\_P weisen Sie der Task 2 die gewünschte Priorität zu. Sie können den Tasks unterschiedliche Prioritäten zuweisen. Nach einem Steuerungsreset oder wenn Sie den Tasks keinen Wert zuweisen, werden die Prioritäten der Tasks beim Systemstart auf 1 gesetzt, d.h. alle Tasks laufen mit der gleichen Priorität.

Je höher die Prioritätszahl für die Task ist, desto langsamer läuft die entsprechende Task. Ein Wert von 2 bedeutet, daß die CPU zwei Befehle aus den anderen Tasks in der gleichen Zeit ausführt wie einen Befehl aus der entsprechend verlangsamten Task. Der Wertebereich für die Taskpriorität liegt zwischen 1 und 255.

### Bitergebnisspeicher (BES)

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

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Sie können auch innerhalb der Task mit dem LAD\_P2-Befehl die Ausführungsposition verändern.

### Siehe auch

Programmablaufbefehle (Seite 40)  
 LAD\_P1 (Seite 78)  
 LAD\_P3 (Seite 80)  
 LAD\_P4 (Seite 81)

## n LAD\_P3

Task starten: LAD\_P3 Label 

Task beenden: LAD\_P3 TASK\_STOP 

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_VA    V_TASK_3_P, 2           // Weist Task3 die Taskpriorität 2 zu
LAD_P3    LABEL                   // Startet die Task3
LAD_P3    TASK_STOP               // Beendet die Task3
```

### Task beenden

Mit LAD\_P3 TASK\_STOP beenden Sie die Task 3. War die Task des entsprechenden Parallelprogramms bereits aktiv, wird der Unterprogrammstart dieses Parallelprogramms gelöscht, d.h. alle Unterprogramme werden beendet und das laufende Programm wird auf die angegebene Adresse verzweigt. Verwenden Sie den Befehl LAD\_P3 TASK\_STOP mit äußerster Vorsicht: Falls Sie mit LAD\_P3 TASK\_STOP das aktuelle Parallelprogramm beenden und keine weiteren Parallelprogramme aktiv sind, dann bleibt das System stehen.

### Prioritäten festlegen

Mit Hilfe der Systemvariablen V\_TASK\_3\_P weisen Sie der Task 3 die gewünschte Priorität zu. Sie können den Tasks unterschiedliche Prioritäten zuweisen. Beim Steuerungsreset werden die Prioritäten der Tasks beim Systemstart auf 1 gesetzt, d.h. alle Tasks laufen mit der gleichen Priorität.

Je höher die Prioritätszahl für die Task ist, desto langsamer läuft die entsprechende Task. Ein Wert von 2 bedeutet, daß die CPU zwei Befehle aus den anderen Tasks in der gleichen Zeit ausführt wie einen Befehl aus der entsprechend verlangsamten Task. Der Wertebereich für die Taskpriorität liegt zwischen 1 und 255.

### Bitergebnisspeicher (BES)

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

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Sie können auch innerhalb der Task mit dem LAD\_P3-Befehl die Ausführungsposition verändern.

### Siehe auch

Programmablaufbefehle (Seite 40)  
 LAD\_P1 (Seite 78)  
 LAD\_P2 (Seite 79)  
 LAD\_P4 (Seite 81)



## n LAD\_P4

Task starten: LAD\_P4 Label   
 Task beenden: LAD\_P4 TASK\_STOP

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_VA    V_TASK_4_P, 2           // Weist Task4 die Taskpriorität 2 zu
LAD_P4    LABEL                   // Startet die Task4
LAD_P4    TASK_STOP               // Beendet die Task4
```

### Task beenden

Mit LAD\_P4 TASK\_STOP beenden Sie die Task 4. War die Task des entsprechenden Parallelprogramms bereits aktiv, wird der Unterprogrammstart dieses Parallelprogramms gelöscht, d.h. alle Unterprogramme werden beendet und das laufende Programm wird auf die angegebene Adresse verzweigt. Verwenden Sie den Befehl LAD\_P4 TASK\_STOP mit äußerster Vorsicht: Falls Sie mit LAD\_P4 TASK\_STOP das aktuelle Parallelprogramm beenden und keine weiteren Parallelprogramme aktiv sind, dann bleibt das System stehen.

### Prioritäten festlegen

Mit Hilfe der Systemvariablen V\_TASK\_4\_P weisen Sie der Task 4 die gewünschte Priorität zu. Sie können den Tasks unterschiedliche Prioritäten zuweisen. Beim Steuerungsreset werden die Prioritäten der Tasks beim Systemstart auf 4 gesetzt, d.h. alle Tasks laufen mit der gleichen Priorität.

Je höher die Prioritätszahl für die Task ist, desto langsamer läuft die entsprechende Task. Ein Wert von 2 bedeutet, daß die CPU zwei Befehle aus den anderen Tasks in der gleichen Zeit ausführt wie einen Befehl aus der entsprechend verlangsamten Task. Der Wertebereich für die Taskpriorität liegt zwischen 1 und 255.

### Bitergebnisspeicher (BES)

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

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Sie können auch innerhalb der Task mit dem LAD\_P4-Befehl die Ausführungsposition verändern.

### Siehe auch

Programmablaufbefehle (Seite 40)  
 LAD\_P1 (Seite 78)  
 LAD\_P2 (Seite 79)  
 LAD\_P3 (Seite 80)

**n LAD\_VA**LAD\_VA Variable  $\mathbb{V}$ , Wert  $\mathbb{K}$ 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

LAD\_VA lädt den angegebenen Wert in die Variable.

**Operation**(Variable)  $\Leftarrow$  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 36)

**n LAD\_VI**LAD\_VI Variable  $\mathbb{V}$ , Zeiger  $\mathbb{V}$ 

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)  $\Leftarrow$  (Zeiger  $\rightarrow$  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 36)

## n LAD\_VL

LAD\_VL Variable , Label

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)  $\Leftarrow$  (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:
GEHUPRV   V_ZYKLUS              // Hauptprogramm Schleife
GEHURPI   Control              // Überwachungsfunktionen immer aufrufen
SPRING    Haupt                // Schleife

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

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

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

### Hinweise

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

### Siehe auch

Programmablaufbefehle (Seite 40)

GEHUPRV (Seite 69)

**n LAD\_VM**LAD\_VM Variable  $\mathbb{M}$ , Erster Merker  $\mathbb{M}$ 

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 20 – LAD\_VM Übersicht

**Operation**

(Bit 0 der Variable)  $\mathbb{C}$  (Erster Merker + 0)  
 (Bit 1 der Variable)  $\mathbb{C}$  (Erster Merker + 1)  
 (Bit 2 der Variable)  $\mathbb{C}$  (Erster Merker + 2)  
 ...  
 (Bit 31 der Variable)  $\mathbb{C}$  (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, VARERG        // Ergebnis wieder in die Variable speichern
```

**Hinweise**

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n 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 43)  
 LAD\_MV (Seite 77)

**n LAD\_VV**LAD\_VV Variable1 , Variable2 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

LAD\_VV lädt den Inhalt der Variable2 in die Variable1.

**Operation**(Variable1)  $\oplus$  (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 36)

**n MACRO**Name  MACRO [Par 1 , [Par 2 , [...]]]

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

- n 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 kompiliert. Ein zwanzig Zeilen langes Makro belegt bei jedem Aufruf exakt diese zwanzig Zeilen SPS-Programmspeicher.
- n 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 45)  
 ENDM (Seite 63)  
 EXITM (Seite 63)


## n MOD\_A

MOD\_A Ausgang 

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)  (Bitergebnisspeicher)

### Beispiel

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

### 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

Ein-/Ausgangsbefehle (Seite 34)

## n MOD\_AI

MOD\_AI Zeiger  $\overline{M}$

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  $\rightarrow$  Ausgang)  $\Leftarrow$  (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 34)

## n MOD\_M

MOD\_M Merker  $\overline{M}$

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	y Nein	n Nein

MOD\_M überträgt den Zustand des Bitergebnisspeichers in den angegebenen Merker.

### Operation

(Merker)  $\Leftarrow$  (Bitergebnisspeicher)

### Beispiel

```
MOD_M    M_TEST              // überträgt 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 33)

## n MOD\_MI

MOD\_MI Zeiger 

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  $\rightarrow$  Merker)  $\Leftarrow$  (Bitergebnisspeicher)

### Beispiel

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

### 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\_MI um den Inhalt des Bitergebnisspeichers auf einen Merker zu übertragen, dessen Nummer bei der Programmerstellung noch nicht feststeht.

### Siehe auch

Merkerbefehle (Seite 33)

## n MUL\_II

MUL\_II Zeiger1 , Zeiger2 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

MUL\_II multipliziert die Inhalte der durch Zeiger1 und Zeiger2 bestimmten Variablen. Das Ergebnis wird im Ergebnisspeicher VARERG abgelegt, ein etwaiger Multiplikationsüberlauf (> 32 Bit) wird im Ergebnisspeicher MUL\_REST gespeichert.

### Operation

(VARERG)  $\Leftarrow$  (Zeiger1  $\rightarrow$  Variable) \* (Zeiger2  $\rightarrow$  Variable)  
 (MUL\_REST)  $\Leftarrow$  ((Zeiger1  $\rightarrow$  Variable) \* (Zeiger2  $\rightarrow$  Variable)) >> 32

### Beispiel

```
MUL_II    V_ZEIGER1, V_ZEIGER2 // Multipliziert den Inhalt der Variable, auf
                               // die V_ZEIGER1 zeigt, mit dem Inhalt der
                               // Variable, auf die V_ZEIGER2 zeigt.
                               // Das Ergebnis wird in VARERG gespeichert
```

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)



## n MUL\_IV

MUL\_IV Zeiger  $\overline{V}$ , Variable  $\overline{V}$

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)  $\zeta$  (Zeiger  $\hat{a}$  Variable) \* (Variable)  
 (MUL\_REST)  $\zeta$  ((Zeiger  $\hat{a}$  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

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

### Siehe auch

Variablenbefehle (Seite 36)

## n MUL\_VA

MUL\_VA Variable  $\overline{V}$ , Wert  $\overline{K}$

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)  $\zeta$  (Variable) \* Wert  
 (MUL\_REST)  $\zeta$  ((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 36)

**n MUL\_VI****MUL\_VI** Variable , Zeiger 

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)  ☒ (Variable) * (Zeiger à Variable)
(MUL_REST) ☒ ((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 36)

**n MUL\_VV****MUL\_VV** Variable1 , Variable2 

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)  ☒ (Variable1) * (Variable2)
(MUL_REST) ☒ ((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 36)

## n NLAD\_A

NLAD\_A Ausgang  $\bar{A}$

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> Ja

NLAD\_A lädt den invertierten Zustand des Ausgangs in den Bitergebnisspeicher.

### Operation

(Bitergebnis)  $\zeta$   $\sim$ (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 34)

## n NLAD\_E

NLAD\_E Eingang  $\bar{E}$

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> Ja

NLAD\_E lädt den invertierten Zustand eines Eingangs in den Bitergebnisspeicher.

### Operation

(Bitergebnis)  $\zeta$   $\sim$ (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 34)

## n NLAD\_M

NLAD\_M Merker  $\overline{M}$

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	<b>y</b> Nein	<b>p</b> Ja

NLAD\_M lädt den invertierten Zustand des Merkers in den Bitergebnisspeicher.

### Operation

(Bitergebnis)  $\overline{C}$   $\sim$ (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 33)

## n NODER\_A

NODER\_A Ausgang  $\overline{A}$

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\overline{C}$  (Bitergebnis) |  $\sim$ (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 34)

Wahrheitstabelle (Seite 255)

## n NODER\_E

NODER\_E Eingang 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\leftarrow$  (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 34)

Wahrheitstabelle (Seite 255)

## n NODER\_M

NODER\_M Merker 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	<b>y</b> Nein	<b>p</b> 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)  $\leftarrow$  (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

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

### Siehe auch

Merkerbefehle (Seite 33)

Wahrheitstabelle (Seite 255)

**n NUND\_A****NUND\_A Ausgang** 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\leftarrow$  (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**

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

**Siehe auch**

Ein-/Ausgangsbefehle (Seite 34)

Wahrheitstabelle (Seite 255)

**n NUND\_E****NUND\_E Eingang** 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\leftarrow$  (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**

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

**Siehe auch**

Ein-/Ausgangsbefehle (Seite 34)

Wahrheitstabelle (Seite 255)

## n NUND\_M

NUND\_M Merker 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	<b>y</b> Nein	<b>p</b> 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)  $\&$  (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

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

### Siehe auch

Merkerbefehle (Seite 33)

Wahrheitstabelle (Seite 255)

## n ODER\_A

ODER\_A Ausgang 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\&$  (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 34)

Wahrheitstabelle (Seite 255)

**n ODER\_E****ODER\_E Eingang** 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\zeta$  (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 34)

Wahrheitstabelle (Seite 255)

**n ODER\_M****ODER\_M Merker** 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	<b>y</b> Nein	<b>p</b> 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)  $\zeta$  (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 33)

Wahrheitstabelle (Seite 255)



## n ODER\_II

ODER\_II Zeiger1 , Zeiger2 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

ODER\_II verknüpft die Inhalte der durch die angegebenen Zeiger1 und Zeiger2 bestimmten Variablen mit einem binären ODER. Das Ergebnis der Verknüpfung wird in der Ergebnisvariablen VARERG gespeichert.

### Operation

(VARERG)  $\Leftarrow$  (Zeiger1  $\rightarrow$  Variable) | (Zeiger2  $\rightarrow$  Variable)

### Beispiel

```
// In diesem Beispiel verknüpfen wir den Inhalt der ersten Tabelle, die bei
// Variable 200 beginnt, mit dem Inhalt der zweiten Tabelle, die bei Variable 400
// beginnt. Das Verknüpfungsergebnis wird wieder in der ersten Tabelle gespeichert.
// Es werden insgesamt 99 Variablen miteinander verknüpft.
```

```
DEF_W      200, TABELLE1      // Anfang der ersten Tabelle
DEF_W      400, TABELLE2      // Anfang der zweiten Tabelle
DEF_V      200, ANZ_GLEICH     // Vergleichszähler definieren

LAD_VA     V_ZEIGER1, TABELLE1 // V_ZEIGER1 mit Originaltabelle laden
LAD_VA     V_ZEIGER2, TABELLE2 // V_ZEIGER2 mit Zieltabelle laden
LAD_VA     V_ZAEHLER, 99      // Wir wollen 99 Variablen vergleichen

LOOP:
ODER_II    V_ZEIGER2, V_ZEIGER1 // Variable der ersten Tabelle mit Variable der
// zweiten Tabelle verknüpft
LAD_IV     V_ZEIGER1, VARERG   // Ergebnis der Verknüpfung in Tabelle schreiben

INC_V      V_ZEIGER1, 1       // Zeiger auf TABELLE1 erhöhen
INC_V      V_ZEIGER2, 1       // Zeiger auf TABELLE2 erhöhen
DEC_V      V_ZAEHLER, 1       // Zähler für Vergleichen verringern
VERG_VA    V_ZAEHLER, 0       // Bereits alles verglichen?
LAD_M      M_GLEICH           // Vergleichsergebnis abfragen
SPRINGN    LOOP               // Noch nicht alles kopiert, weiter
```

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)

**n ODER\_IV**ODER\_IV Zeiger  $\mathbb{M}$ , Variable  $\mathbb{M}$ 

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)  $\mathcal{C}$  (Zeiger  $\mathbb{M}$  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-Variablen 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 36)

**n ODER\_VA**ODER\_VA Variable  $\mathbb{M}$ , Wert  $\mathbb{K}$ 

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)  $\mathcal{C}$  (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 36)

## n ODER\_VI

ODER\_VI Variable  $\bar{V}$ , Zeiger  $\bar{V}$

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)  $\zeta$  (Variable) | (Zeiger  $\bar{a}$  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 36)

## n ODER\_VV

ODER\_VV Variable1  $\bar{V}$ , Variable2  $\bar{V}$

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)  $\zeta$  (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 36)

## n PRLABS

PRLABS Achse , Absolute Position

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

PRLABS lädt eine absolute Zielposition in die angegebene Achse vor. Die Zielposition ist in dem als zweiten angegebenen Parameter enthalten. PRLABS löst noch keinen Achsenstart aus, deshalb kann er auch während einer laufenden Positionierung verwendet werden. Zum Starten der Achsen auf die mit PRLABS vorgeladene Position verwenden Sie bitte STPRLD.

### Beispiel

```
// Die Achsen 1 und 2 sollen auf die Position 325 positioniert werden.

LAD_VA    V_SOLLP_A1, 3250    // Lade Sollposition in die Variable SOLLP_A1
LAD_VA    V_GESW_A1, 5000    // Lade Verfahrgeschwindigkeit in Variable
// GESW_A1
SETVEL    1, V_GESW_A1      // Laden der Verfahrgeschwindigkeit für Achse1
SETVEL    2, V_GESW_A1      // Laden der Verfahrgeschwindigkeit für Achse 2
PRLABS    1, V_SOLLP_A1      // Lade die absolute Position der Achse mit dem
// Wert, der in der Variablen SOLLP_A1 enthalten
// ist vor
PRLABS    2, V_SOLLP_A1      // Lade die absolute Position der Achse 2 mit dem
// Wert, der in der Variablen SOLLP_A1 enthalten
// ist vor
STPRLD    1                  // Starte Achse 1 mit den vorgeladenen Daten
STPRLD    2                  // Starte Achse 2 mit den vorgeladenen Daten
LOOP:
LAD_M     M_INP_A1           // Der Merker M_INP_A1 zeigt den Status der
// Achse 1 an
UND_M     M_INP_A2           // Der Merker M_INP_A2 zeigt den Status der
// Achse 2 an
SPRINGN   LOOP              // Merker = 1, Achse ist in Position
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen.
- n Sie können diesen Befehl auch verwenden, um einen schnellen Start dieser Achse über die AUTOSTART-Sonderfunktion des Achskontollers vorzubereiten. Entsprechende Informationen finden Sie bei der Beschreibung des Befehls SETFUN.

### Siehe auch

Positionierbefehle (Seite 41)  
 PRLREL (Seite 101)  
 STPRLD (Seite 130)  
 PWRDRV (Seite 102)

## n PRLREL

PRLREL Achse , Relative Position

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

PRLREL lädt eine relative Zielposition in die angegebenen Achse vor. Die Zielposition ist in dem als zweiten angegebenen Parameter enthalten. PRLREL löst noch keinen Achsenstart aus, deshalb kann er auch während einer laufenden Positionierung verwendet werden. Zum Starten der Achsen auf die mit PRLABS vorgeladene Position verwenden Sie bitte STPRLD.

### Beispiel

// Die Achsen 1 und 2 sollen um die Strecke 150 verfahren werden.

```
LAD_VA    V_STRE_A1, 1500    // Lade Strecke in die Variable STRECKE_A1
LAD_VA    V_GESW_A1, 5000   // Lade Verfahrgeschwindigkeit in die
                               // Variable GESCHW_A1
SETVEL    1, V_GESW_A1      // Laden der Verfahrgeschwindigkeit für die
                               // Achse 1
SETVEL    2, V_GESW_A1      // Laden der Verfahrgeschwindigkeit für die
                               // Achse 2
PRLREL    1, V_STRE_A1      // Lade die Strecke der Achse 1 mit dem Wert, der
                               // in der Variablen STRECKE_A1 enthalten ist vor
PRLREL    2, V_STRE_A1      // Lade die Strecke der Achse 2 mit dem Wert, der
                               // in der Variablen STRECKE_A1 enthalten ist vor
STPRLD    1                  // Starte Achse 1, mit den vorgeladenen Daten
STPRLD    2                  // Starte Achse 2, mit den vorgeladenen Daten
LOOP:
LAD_M     M_INP_A1           // Der Merker M_INP_A1 zeigt den Status
                               // der Achse 1
UND_M     M_INP_A2           // Der Merker M_INP_A2 zeigt den Status
                               // der Achse 2
SPRINGN   LOOP              // Merker = 1, Achse ist in Position
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen.
- n Sie können diesen Befehl auch verwenden, um einen schnellen Start dieser Achse über die AUTOSTART-Sonderfunktion des Achscontollers vorzubereiten. Entsprechende Informationen finden Sie bei der Beschreibung des Befehls SETFUN.

### Siehe auch

Positionierbefehle (Seite 41)  
 PRLREL (Seite 101)  
 STPRLD (Seite 130)  
 PWRDRV (Seite 102)  
 SETFUN (Seite 112)

## n PWRDRV

PWRDRV Achse , Status

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

PWRDRV schaltet das Leistungsteil der angegebenen Achse ein oder aus. Sie müssen immer das Leistungsteil einschalten bevor Sie eine Positionierung starten können. Dies gilt auch für Schrittmotorcontroller.

Mögliche Werte für die Konstante sind

n OFF oder AUS (0) = Leistungsteil aus

n ON oder EIN (1) = Leistungsteil ein

### Beispiel

Einschalten:

```
PWRDRV      1, ON           // Leistungsteil Achse 1 einschalten
```

EinschaltenW:

```
LAD_M       M_RGLB_A1      // Meldet der Regler bereit?
UND_M       M_RFGON_A1     // und ist die Freigabe durchgeschaltet?
SPRINGN     EinschaltenW   // Nein, dann noch warten
```

```
STHOME      1, NORMAL      // Referenzfahrt durchführen
UPREND      // Unterprogramm beenden
```

Ausschalten:

```
PWRDRV      1, OFF        // Leistungsteil Achse 1 ausschalten
```

AusschaltenW:

```
LAD_M       M_RFGON_A1     // Freigabe weggenommen?
SPRINGJ     AusschaltenW   // Nein, dann noch warten
UPREND      // Unterprogramm Ende
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Wenn Sie das Leistungsteil einschalten, sollten Sie anschließend auf die Rückmeldung "Leistungsteil eingeschaltet" im entsprechenden Systemmerker warten, bevor Sie eine Positionierung starten.
- n Bitte beachten Sie, daß Sie das Leistungsteil einer Achse nicht nochmals einschalten sollten, wenn es bereits mit PWRDRV eingeschaltet wurde. Dies kann zu nicht vorhersehbaren Ergebnissen führen.

### Siehe auch

Positionierbefehle (Seite 41)

## n RCVSER

RCVSER Modulnr. , Pufferanfang

Gruppe	Abhängig von BES	Verändert BES
Seriellmodul	- Ja	n Nein

RCVSER überträgt serielle Empfangsdaten von einem seriellen Erweiterungsmodul in ein Variablenfeld. Für jedes empfangene Byte wird eine Variable verwendet. Die erste hierfür zu verwendende Variable wird durch die Konstante angegeben.

Das erste Byte des Variablenfelds enthält die Anzahl der in das Feld übertragenen Daten, ab der zweiten Variable beginnen die eigentlichen Empfangsdaten.

### Beispiel

Warten:

```
LAD_M      M_SERI NP_1      // Neue Daten vom seriellen Modul 1?
SPRINGN    Warten         // Nein, wir warten
VERG_VA    V_SERI NP_1, 12 // Mindestens 12 Byte empfangen?
NLAD_M     M_KLEI NER     // Vergleichsergebnis holen
SPRINGN    Warten         // Nein, weiter warten
RCVSER     1, 1000        // Empfangsdaten ab Variable 1000
```

```
// In V1000 steht nun die Anzahl in das Feld übertragener Byte, ab V1001 finden
// sich die Empfangsdaten. Empfangende Texte sind als ASCII codiert.
```

### Hinweise

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

### Siehe auch

Serielle Anbindung (Seite 44)  
 Serielles Modul (Seite 171)  
 SND SER (Seite 120)

## n SETAIO

SETAIO Analoger Ausgang , Wert

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

SETAIO übergibt den Inhalt der angegebenen Variable als Ausgangspegel an den angegebenen Analogausgang. Bei dem Sollwert handelt es sich um einen 12 Bit Wert, der also Werte zwischen 0 und 4095 enthalten kann.

### Beispiel

```
// Im folgenden Beispiel wird der analoge Ausgang 1 auf den Wert 100 gesetzt.
```

```
LAD_VA     V_ANALOG, 100    // Die Variable V_ANALOG wird mit 100 geladen
SETAIO     1, V_ANALOG     // Der analoge Ausgang 1 wird auf den Wert der
                          // Variable V_ANALOG gesetzt.
```

### Hinweise

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

n Welcher Analogwert tatsächlich ausgegeben wird, ist abhängig von der Konfiguration des Moduls.

n Wird ein Wert größer als 4095 oder kleiner als 0 angegeben, so wird stets der maximale Analogwert (4095) ausgegeben.

### Siehe auch

Analog I/O (Seite 35)  
 Analog Ein-/ Ausgänge (Seite 164)

**n SETDSP**SETDSP Display , Funktion 

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

SETDSP löst eine Funktion am angegebenen Display aus. Dabei sind verschiedene Funktionen, wie Text- und Variablenanzeige mit verschiedenen Fonts, Anzeige von einzelnen Buchstaben und Umschalten der Anzeigeseite, Anzeige von Bitmaps, Tastaturklick und Signaltöne möglich. Die gewünschte Funktion wird als zweiter Parameter übergeben.

**Mögliche Werte für den ersten Parameter (Displayauswahl):**

Name	Wert	Bedeutung
DSP1	1	Display Nr.1
DSP2	2	Display Nr.2
RS232	3	Drucken über die serielle Schnittstelle

n Tabelle 21 – Gültige Werte für die Display-Auswahl mit SETDSP

**Mögliche Werte für den zweiten Parameter (Funktionsnummer):**

Name	Wert	Bedeutung
DSP	1	Darstellen mit normaler Schriftart
DSP_BIG	2	Darstellen mit großer Schriftart
DSP_ASCII	3	Einzelnes ASCII-Zeichen ausgeben
DSP_FROM_VAR	4	Variablenfeld als Text ausgeben
DSP_TO_VAR	5	Text in ein Variablenfeld kopieren
DSP_BEEPER	10	Lautsprecher ein-/ausschalten
DSP_BITMAP	11	Gespeichertes Bitmap linksbündig anzeigen
DSP_BITMAP_RIGHT	12	Gespeichertes Bitmap rechtsbündig anzeigen
DSP_PAGE	13	Textseite umschalten
DSP_OFFSET	14	Bitmap-Offset für Sprachumschaltung angeben
DSP_HOTKEY	21	Hotkey-Bitmap linksbündig anzeigen
DSP_HOTKEY_RIGHT	22	Hotkey-Bitmap rechtsbündig anzeigen
DSP_LED_ON	30	LED Testfunktion deaktivieren
DSP_LED_OFF	31	LED-Testfunktion aktivieren

n Tabelle 22 – Funktionen für SETDSP

**Hinweise**

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

**Siehe auch**

Display und Texte (Seite 43)

Display-Programmierung (Seite 160)

LAD\_DT (Seite 72)

LAD\_DV (Seite 73)

SETEDI (Seite 110)



## n SETDSP DSP und DSP\_BIG

Mit LAD\_DT wird das Format für die Ausgabe festgelegt und in die Variable V\_ANZNR die Nummer des entsprechenden Textes oder der Variable geschrieben. SETDSP stellt diesen Text dann auf der Anzeige dar.

### Beispiel

```
// Im folgenden Beispiel soll der Text Nr. 1 in der Zeile 1, ab der
// Position 1, mit der Länge 20 ausgegeben werden.

LAD_DT      V_ANZMSK1, ZEILE1, 1, 20 // Format für die Anzeige festlegen
LAD_VA      V_ANZNR1, 1              // Text auswählen
SETDSP      DSP1, DSP_BIG            // Text anzeigen mit großer Schrift
```

## n SETDSP DSP\_ASCII

Diese Funktion ist nur zusammen mit dem Drucken auf die eingebaute serielle Schnittstelle sinnvoll: hier können Sie ein beliebiges ASCII-Zeichen – auch nicht druckbare – ausgeben. Das auszugebende ASCII-Zeichen wird in V\_ANZNR1 geladen und dann die SETDSP-Funktion ausgeführt.

### Beispiel

```
// In folgendem Beispiel soll ein CR/LF (Zeilenvorschub) ausgegeben werden.

LAD_VA      V_ANZNR3, 13              // ASCII 13 = CR
SETDSP      RS232, DSP_ASCII         // ausgeben

Warte_S1:                                       // Warten bis Zeichen gedruckt
NLAD_M      M_RS232                  // Läuft Ausgabe noch?
SPRINGN     Warte_S1                // Ja, warten

LAD_VA      V_ANZNR3, 10              // ASCII 10 = LF
SETDSP      RS232, DSP_ASCII         // ausgeben

Warte_S2:                                       // Warten bis Zeichen gedruckt
NLAD_M      M_RS232                  // Läuft Ausgabe noch?
SPRINGN     Warte_S2                // Ja, warten
```

## n SETDSP DSP\_FROM\_VAR

Mit dieser Funktion können Sie einen Text in ein Variablenfeld kopieren. Das ist dann hilfreich, wenn Sie Teile einer Displayausgabe während der Laufzeit des SPS-Programms verändern wollen, bevor Sie den Text ausgeben. Alternativ können Sie über diese Funktion auch neue Texte zur Laufzeit erzeugen. So können auch Sonderzeichen ausgegeben werden, die zwar vom Display unterstützt werden (siehe Anhang B - Display-Zeichentabelle auf Seite 256), aber nicht als Text im Editor eingegeben werden können.

Tragen Sie hierzu in V\_ANZNRx die Nummer der ersten Variable des Variablenfelds ein, daß Sie ausgeben möchten. Die gewünschte Position und Formatierung geben Sie wie gewohnt mit LAD\_DT an. Bitte beachten Sie, daß jede Variable genau einem Zeichen entspricht.

### Beispiel

```
LAD_VA      200, 24                    // Sonderzeichen "Pfeil nach oben"
LAD_VA      201, 25                    // Sonderzeichen "Pfeil nach unten"
LAD_VA      202, 32                    // Leerzeichen
LAD_VA      203, 33                    // Ausrufezeichen
LAD_VA      V_ANZNR1, 200              // Variablenfeld ab Variable 200
LAD_DT      V_ANZMSK1, 0, 4, 4        // Zeile 1, Spalte 5, 4 Zeichen ausgeben
SETDSP      DSP1, DSP_FROM_VAR        // ausgeben
```

## n SETDSP DSP\_TO\_VAR

Als Gegenstück zur Funktion 4 des Befehls SETDSP kopiert die Funktion 5 einen definierten SPS-Text in ein Variablenfeld. Dies ermöglicht Ihnen, einen definierten SPS-Text zu verändern oder zu ergänzen, bevor Sie ihn auf dem Display ausgeben.

Tragen Sie in V\_ANZNRx die Textnummer ein. Schreiben Sie in die Variable V\_ANZMSKx die Nummer der ersten Variable, in die der Text kopiert werden soll. Nach Ausführung des Befehls enthält dann die erste angegebene Variable das erste Zeichen des SPS-Textes, die nächste Variable das zweite usw.

### Beispiel

```
// In diesem Beispiel gehen wir davon aus, im Text 20 "Weiter mit Taste >" steht.
// Die Textlänge wurde vom Compiler automatisch auf 20 Zeichen erweitert.
// Wir kopieren nun den Text in ein Variablenfeld, beginnend mit der Variable 1000.

LAD_VA    V_ANZNR1, 20           // Text 20
LAD_VA    V_ANZMSK1, 1000       // in Feld ab Variable 1000 kopieren
SETDSP    DSP1, DSP_TO_VAR      // Kopieren auslösen

// Die Variable 1000 enthält nun den Wert 87 (ASCII-Code für "W"), die Variable
// 1001 den Wert 102 ("e") usw. Wir wollen nun das Zeichen ">" (Textstelle 18, also
// in Variable 1017) mit dem Sonderzeichen "Pfeil links" ersetzen und ausgeben.

LAD_VA    1017, 24              // Sonderzeichen "Pfeil links"
LAD_VA    V_ANZNR1, 1000       // Variablenfeld ab 1000 ausgeben
LAD_DT    V_ANZMSK1, 3, 0, 20  // 20 Zeichen, vierte Zeile, erste Spalte
SETDSP    DSP1, DSP_FROM_VAR    // Text aus Variablenfeld ausgeben
```

## n SETDSP DSP\_BEEP

Manche Bedienteile der MC200 Familie verfügen über einen eingebauten Lautsprecher. Um Signale über diesen Lautsprecher auszugeben steht die Funktion 10 zur Verfügung. In V\_ANZNR wird die Lautsprecherfunktion geladen, und mit SETDSP ausgelöst. Mögliche Werte für V\_ANZNR sind hierbei:

Name	Wert	Bedeutung
DSP_CLICK_ON	1	Tastatur-Klick einschalten
DSP_CLICK_OFF	2	Tastatur-Klick ausschalten
DSP_BEEP_ON	3	Lautsprecher dauerhaft einschalten
DSP_BEEP_125	4	Lautsprecher für 125ms einschalten
DSP_BEEP_250	5	Lautsprecher für 250ms einschalten
DSP_BEEP_500	6	Lautsprecher für 500ms einschalten
DSP_BEEP_1000	7	Lautsprecher für 1s einschalten
DSP_BEEP_2000	8	Lautsprecher für 2s einschalten
DSP_BEEP_OFF	9	Lautsprecher ausschalten

n Tabelle 23 - Lautsprecherfunktionen

### Beispiel

```
// In folgendem Beispiel wird ein 250ms langer Ton auf dem Lautsprecher ausgegeben:

LAD_VA    V_ANZNR1, DSP_BEEP_250 // Länge des Tons auswählen
SETDSP    DSP1, DSP_BEEP         // Funktion auslösen
```

## n SETDSP DSP\_BMP

Bei einigen Displays der MC200 Familie haben Sie die Möglichkeit, Bitmaps im Display zu hinterlegen und vom SPS-Programm aus aufzurufen. Hierzu laden Sie die Nummer des Bitmaps in V\_ANZNR und stellen das Bitmap mit SETDSP dar. Zuvor dargestellter Text wird hierbei gelöscht, später auf das Display geschriebener Text wird über das Bitmap überlagert.

Um festzustellen, ob das von Ihnen verwendete Display Bitmaps unterstützt, beachten Sie bitte die Hinweise in Ihrer Hardware-Dokumentation.

### Beispiel

// In folgendem Beispiel wird das Bitmap 22 angezeigt:

```
LAD_VA    V_ANZNR1, 22           // Bitmap 22 auswählen
SETDSP    DSP1, DSP_BMP         // Bitmap darstellen
```

## n SETDSP DSP\_BMP\_RIGHT

Bei einigen Displays der MC200 Familie haben Sie die Möglichkeit, Bitmaps im Display zu hinterlegen und vom SPS-Programm aus aufzurufen. Hierzu laden Sie die Nummer des Bitmaps in V\_ANZNR und stellen das Bitmap mit SETDSP dar. Zuvor dargestellter Text wird hierbei gelöscht, später auf das Display geschriebener Text wird über das Bitmap überlagert.

Die rechtsbündige Darstellung eines Bitmaps wird nur von Displays der MC200 Familie unterstützt, die über eine Anzeigefläche von mehr als 128 Pixel verfügen. Um festzustellen, ob das von Ihnen verwendete Display diese Funktion unterstützt, beachten Sie bitte die Hinweise in Ihrer Hardware-Dokumentation.

### Beispiel

// In folgendem Beispiel wird das Bitmap 23 rechtsbündig angezeigt:

```
LAD_VA    V_ANZNR1, 23           // Bitmap 23 auswählen
SETDSP    DSP1, DSP_BMP_RIGHT    // Bitmap darstellen
```

## n SETDSP DSP\_PAGE

Bei einigen Displays der MC200 Familie haben Sie die Möglichkeit, zwischen unterschiedlichen Textseiten umzuschalten. Dies ist nützlich, um z.B. einen Hilfetext darzustellen, ohne den vorherigen Display-Inhalt zu löschen. Es stehen Ihnen 8 Textseiten (von 1 bis 8) zur Verfügung. Schreiben Sie hierzu die gewünschte Textseite in die Variable V\_ANZNR und wählen Sie die Textseite mit SETDSP. Bitte beachten Sie, daß diese Funktion nur für das MC200BED zur Verfügung steht.

Ob das von Ihnen verwendete Display die Textseiten-Umschaltung unterstützt entnehmen Sie bitte ihrer Hardware-Dokumentation.

### Beispiel

```
LAD_DT    V_ANZMSK1, ZEILE1, 1, 20 // Format für die Anzeige festlegen
LAD_VA    V_ANZNR1, 1              // Text auswählen
SETDSP    DSP1, DSP                // Text anzeigen
LAD_VA    V_ANZNR1, 2              // Textseite 2 auswählen
SETDSP    DSP1, DSP_PAGE           // Umschalten auf Seite 2. Im Display erscheint
// jetzt der vorherige Inhalt der zweiten Seite
LAD_DT    V_ANZMSK1, ZEILE4, 4, 6 // Format für die Anzeige festlegen
LAD_VA    V_ANZNR1, 2              // Text auswählen
SETDSP    DSP1, DSP                // Text auf der zweiten Seite anzeigen
LAD_VA    V_ANZNR1, 1              // Textseite 1 auswählen
SETDSP    DSP1, DSP_PAGE           // Umschalten auf Seite 1. Jetzt erscheint wieder
// der Display-Inhalt vor der letzten Umschaltung
```

## n SETDSP DSP\_OFFSET (Sprachumschaltung)

Um mehrsprachige Anwendungen zu realisieren, können Sie innerhalb der MC200CPU bzw. des MC200PROFI Texte für alle Zielsprachen definieren. Diese können Sie hinterher über die Systemvariable V\_TXT\_OFFSET (für Texte) und 14 (für Bitmaps) umschalten. Wichtig ist hierbei, daß Sie folgendes beachten:

- n Für jede Sprache innerhalb des SPS-Programms müssen die gleiche Anzahl Texte und die gleiche Anzahl Bitmaps definiert sein.
- n Die gleichen Texte müssen in den verschiedenen Sprachen immer auf der gleichen Textnummer – abzüglich des Textoffsets – liegen. Wenn Sie z.B. den Text "Bitte drücken Sie F1" auf die Textnummer 22 legen und Ihre Englischen Text bei 200 anfangen, dann müßte "Please press F1" auf der Textnummer 222 liegen und die ggf. französische Version dann auf 422.
- n Wenn ein Textoffset aktiv ist, dann wird immer die bei SETDSP angegebene Textnummer zuzüglich des Textoffsets angezeigt, bei einem Offset von 200 und dem Text 22 dann eben der Text 222.

Laden Sie für die Sprachumschaltung den Textoffset (also die Nummer des ersten Texts in der jeweiligen Sprache, abzüglich "1") in die Variable V\_TXT\_OFFSET. Laden Sie – falls gewünscht – den Bitmap-Offset (also die Nummer des ersten Bitmaps in der jeweiligen Sprache, abzüglich "1") in die Variable V\_ANZNRx. Die Sprache wird sofort nach Ausführung von SETDSP umgeschaltet.

### Beispiel

```

TEXT10    "Bitte drücken Sie F1" // deutscher Text
...
TEXT210   "Please press F1      " // englischer Text
...
TEXT410   "Touche F1 S. V. P.  " // französischer Text

VERG_VA   V_SPRACHE, 0          // Sprache "Deutsch"?
LAD_M     M_GLEICH              // Ergebnis prüfen
LAD_VA    V_TXT_OFFSET, 0       // deutscher Textoffset ist 0
LAD_VA    V_ANZNR1, 0          // deutscher Bitmapoffset ist 0
SETDSP    DSP1, DSP_OFFSET      // Sprachoffset setzen
VERG_VA   V_SPRACHE, 1         // Sprache "Englisch"?
LAD_M     M_GLEICH              // Ergebnis prüfen
LAD_VA    V_TXT_OFFSET, 200     // englischer Textoffset ist 200
LAD_VA    V_ANZNR1, 20         // englischer Bitmapoffset ist 20
SETDSP    DSP1, DSP_OFFSET      // Sprachoffset setzen
VERG_VA   V_SPRACHE, 2         // Sprache "Französi sch"?
LAD_M     M_GLEICH              // Ergebnis prüfen
LAD_VA    V_TXT_OFFSET, 400     // französischer Textoffset ist 400
LAD_VA    V_ANZNR1, 40         // französischer Bitmapoffset ist 40
SETDSP    DSP1, DSP_OFFSET      // Sprachoffset setzen

```

## n SETDSP DSP\_HOTKEY

Bei einigen Displays der MC200 Familie haben Sie die Möglichkeit, Bitmaps im Display zu hinterlegen und vom SPS-Programm aus aufzurufen. Eine spezielle Variante dieser Bitmaps sind die "Hotkey-Bitmaps", die eine Beschriftung von Softkeys zulassen. Hierzu laden Sie die Nummer des Bitmaps in V\_ANZNR und stellen das Bitmap mit SETDSP dar. Zuvor dargestellter Text wird hierbei gelöscht, später auf das Display geschriebener Text wird über das Bitmap überlagert.

Um festzustellen, ob das von Ihnen verwendete Display Hotkey-Bitmaps unterstützt, beachten Sie bitte die Hinweise in Ihrer Hardware-Dokumentation. Grundsätzlich gilt, daß diese Funktion nur bei Displays mit mehr als 128 Pixel Anzeighöhe zur Verfügung steht.

### Beispiel

// In folgendem Beispiel wird das Bitmap 22 als Hotkey linksbündig angezeigt:

```
LAD_VA      V_ANZNR1, 22           // Bitmap 22 auswählen
SETDSP      DSP1, DSP_HOTKEY     // Als Hotkey-Bitmap darstellen
```

## n SETDSP DSP\_HOTKEY\_RIGHT

Bei einigen Displays der MC200 Familie haben Sie die Möglichkeit, Bitmaps im Display zu hinterlegen und vom SPS-Programm aus aufzurufen. Eine spezielle Variante dieser Bitmaps sind die "Hotkey-Bitmaps", die eine Beschriftung von Softkeys zulassen. Hierzu laden Sie die Nummer des Bitmaps in V\_ANZNR und stellen das Bitmap mit SETDSP dar. Zuvor dargestellter Text wird hierbei gelöscht, später auf das Display geschriebener Text wird über das Bitmap überlagert.

Um festzustellen, ob das von Ihnen verwendete Display Hotkey-Bitmaps unterstützt, beachten Sie bitte die Hinweise in Ihrer Hardware-Dokumentation. Grundsätzlich gilt, daß diese Funktion nur bei Displays mit mehr als 128 Pixel Anzeighöhe zur Verfügung steht.

### Beispiel

// In folgendem Beispiel wird das Bitmap 23 als Hotkey rechtsbündig angezeigt:

```
LAD_VA      V_ANZNR1, 23           // Bitmap 23 auswählen
SETDSP      DSP1, DSP_HOTKEY_RIGHT // Als Hotkey-Bitmap darstellen
```

## n SETDSP DSP\_LED\_ON und DSP\_LED\_OFF

Um zu Testen, ob alle Tastatur-LEDs korrekt funktionieren und nicht defekt sind, können Sie die Funktionen 30 und 31 verwenden. Schreiben Sie in V\_ANZNR den Wert 31 um den LED-Test zu starten und 30 um den Test zu beenden. Rufen Sie dann SETDSP auf um die Funktion auszuführen.

### Beispiel

// Hier wird der Test gestartet, ein Timer aktiviert, nach Ablauf des  
// Timers der Test wieder beendet.


```
SETDSP      DSP1, DSP_LED_ON       // LED-Test auslösen
LAD_VA      V_TIMER_1, 500        // Timer: 5s
EIN_M       M_TIMER_1            // Timer starten
```

Warten:

```
NLAD_M      M_TIMER_1            // Timer schon abgelaufen?
SPRINGN     Warten              // Nein, noch warten

SETDSP      DSP1, DSP_LED_OFF     // LED-Test beenden
```

## n SETEDI

SETEDI Display , Funktion 

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

SETEDI steuert die Funktion des integrierten Variableneditors. Der Variableneditor ist eine Betriebssystemfunktion, die es erlaubt, über die Tastatur numerische Eingaben vorzunehmen. Mit der ersten Konstante wird das zu verwendende Display gewählt. Mit der zweiten Konstante wird die eigentliche Funktion ausgelöst.

**Mögliche Werte für den ersten Parameter (Displayauswahl):**

Name	Wert	Bedeutung
DSP1	1	Display Nr. 1
DSP2	2	Display Nr. 2

n Tabelle 24 – Gültige Werte für die Display-Auswahl bei SETEDI

**Mögliche Werte für den zweiten Parameter (Funktionsnummer):**

Name	Wert	Bedeutung
EDI_OFF	0	Editor beenden
EDI_ON	1	Editieren mit normaler Schriftart
EDI_ON_BIG	2	Editieren mit großer Schriftart
EDI_KEYBOARD	3	Auswahl der Tastaturbelegung

n Tabelle 25 – Funktionen für SETEDI

Während der Editor aktiv ist (d.h. mit den Funktionen EDI\_ON oder EDI\_ON\_BIG eingeschaltet wurde), können die numerischen Tasten sowie die Taste +/- auf der Tastatur nicht vom SPS-Programm aus abgefragt werden. Der Status dieser Tasten wird erst nach Abschalten des Editors mit der Funktion EDI\_OFF wieder übermittelt.

### Hinweise

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

### Siehe auch

Display und Texte (Seite 43)  
 Display-Programmierung (Seite 160)  
 LAD\_DV (Seite 73)  
 SETDSP (Seite 104)

## n SETEDI EDI\_ON, EDI\_ON\_BIG und EDI\_OFF

### Beispiel

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

DEF_V      214, V_TEST          // Variable die editiert werden soll

LAD_DV     V_INPMSK1, ZEILE1, 4, 5, 2 // Format für Bearbeitung festlegen
LAD_VV     V_INPDAT1, V_TEST      // Wert für Editor festlegen
SETEDI     DSP1, EDI            // Variable editieren
LAD_VV     V_TEST, V_INPDAT1     // editierten Wert wieder in Variable speichern

WARTE:
LAD_M      M_KEY_ENTER          // Hat der Benutzer die Enter-Taste gedrückt?
ODER_M     M_KEY_ESCAPE         // oder die Escape-Taste?
SPRINGN    Warte                // Nein, Editierung läuft noch

SETEDI     DSP1, EDI_OFF        // Editor beenden
LAD_M      M_KEY_ENTER          // Eingabe mit Enter-Taste bestätigt?
LAD_VV     V_TEST, V_INPDAT1    // dann editierten Wert übernehmen
```

## n SETEDI EDI\_KEYBOARD

Mit der Funktion 3 können Sie die Tastaturbelegung umschalten. Die Standardbelegung ist beim Systemstart immer die Tastaturbelegung für das MC200BED. Wenn Sie nun ein anderes Display verwenden, dann müssen Sie die entsprechende Tastaturbelegung auswählen. Hierzu wird die Nummer der Tastatur in V\_ANZNR geladen und mit SETDSP aktiviert. Folgende Werte sind gültig:

Name	Wert	Bedeutung
EDI_MC200BED	1	MC200BED
EDI_MC200BDL	2	MC200BDL oder MC200BED-ABA
EDI_MC200HT	3	MC200HT

n Tabelle 26 – Gültige Tastaturbelegungen

### Beispiel

```
// Folgendes Beispiel wählt die Tastaturbelegung für das MC200HT als Display 1 aus:

LAD_VA     V_INPDAT1, EDI_MC200HT // Auswahl der Tastaturbelegung
SETEDI     DSP1, EDI_KEYBOARD     // Tastaturbelegung wählen
```

## n SETFUN

SETFUN Achse , Funktion

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

SETFUN aktiviert oder deaktiviert Sonderfunktionen der angegebenen Achse.

### Gültige Werte für den Parameter 2 (Funktionsnummer)

Name	Wert	Funktion
AFUN_FOLLOW_ON	1	Schaltet die fortlaufende Übertragung der Schleppfehlerwerte vom Achskontroller zur CPU ein.
AFUN_FOLLOW_OFF	2	Schaltet die fortlaufende Übertragung der Schleppfehlerwerte vom Achskontroller zur CPU aus (Standard).
AFUN_VEL_ON	3	Schaltet die fortlaufende Übertragung der Ist-Geschwindigkeit vom Achskontroller zur CPU ein.
AFUN_VEL_OFF	4	Schaltet die fortlaufende Übertragung der Ist-Geschwindigkeit vom Achskontroller zur CPU aus (Standard).
AFUN_MSR_ON	5	Schaltet die Meßfunktion über den Referenz-Eingang des Achskontrollers ein. Nähere Informationen finden Sie im Kapitel 4.1 - Messfunktionen (Seite 156)
AFUN_MSR_OFF	6	Schaltet die Meßfunktion über den Referenz-Eingang des Achskontrollers aus (Standard).
AFUN_AUTOSTART_ON	7	Schaltet den automatischen Start der Achse bei Auslösen des Referenz-Eingangs auf dem Achsmodul ein.
AFUN_AUTOSTART_OFF	8	Schaltet den automatischen Start der Achse bei Auslösen des Referenz-Eingangs auf dem Achsmodul aus (Standard).
AFUN_SMSYNC_ON	11	Aktiviert den Synchronstart der beiden auf einem MC200SM2-Modul liegenden Schrittmotorachsen.
AFUN_SMSYNC_OFF	12	Deaktiviert den Synchronstart der beiden auf einem MC200SM2-Modul liegenden Schrittmotorachsen.
AFUN_MOCPOS_ON	11	Aktiviert das Senden der tatsächlichen Ist-Position der Achse. Normalerweise wird als Ist-Position die theoretische Ist-Position gemäß Rampengenerator ausgegeben.
AFUN_MOCPOS_OFF	12	Deaktiviert das Senden der tatsächlichen Ist-Position der Achse. Im Anschluß wird wieder die theoretische Ist-Position gemäß Rampengenerator ausgegeben.
AFUN_OVR_ON	13	Aktiviert die Auswertung des Override-Wertes (Standard).
AFUN_OVR_OFF	14	Deaktiviert die Auswertung des Override-Wertes.
AFUN_AUTOSTOP_ON	15	Schaltet den automatischen Achsenstart bei Auslösen des Referenz-Eingangs auf dem Achsmodul ein.
AFUN_AUTOSTOP_OFF	16	Schaltet den automatischen Achsenstop bei Auslösen des Referenz-Eingangs auf dem Achsmodul aus (Standard).
AFUN_OUTPUT_ON	17	Schaltet die Ausgangsfunktionen des Achscontrollers ein.
AFUN_OUTPUT_OFF	18	Schaltet die Ausgangsfunktionen des Achscontrollers aus (Standard).

n Tabelle 27 – Sonderfunktionen für SETFUN



**Beispiel**

```

SETFUN    1, AFUN_MSR_ON           // Aktiviert die Meßfunktion für Achse 1
SETFUN    2, AFUN_OVR_OFF         // Aktiviert die Override-Auswertung für Achse 2

```

**Hinweise**

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Alle Sonderfunktionen der Achscontroller lassen sich ausser mit SETFUN auch mit SETPAR beeinflussen. Dies bedeutet: Sie können nicht nur Sonderfunktionen mit SETPAR aktivieren oder deaktivieren, sondern auch versehentlich eine aktivierte Sonderfunktion mit SETPAR wieder überschreiben.

Nach Möglichkeit verwenden Sie deshalb in Ihrem SPS-Programm nur eine der beiden Konfigurationsmöglichkeiten – entweder SETPAR oder SETFUN. Sollten Sie trotzdem beide Möglichkeiten der Parametrierung benutzen, lesen Sie bitte aufmerksam die Liste der Achsparameter durch (Kapitel 5.2 - Achsparameter ab Seite 179).

- n Bitte verwenden Sie die Sonderfunktionen mit **äußerster** Vorsicht, da hierdurch das Verhalten der Achscontroller teilweise gravierend verändert wird! Bitte beachten Sie auch, daß die Einstellung einiger Sonderfunktionen durch die Achsparameter verändert werden kann. Vergleichen Sie hierzu Kapitel 5.2 - Achsparameter (ab Seite 179).

**Siehe auch**

Positionierbefehle (Seite 41)  
 SETPAR (Seite 115)  
 GETPAR (Seite 70)  
 Achsparameter (Seite 179)

**n SETNUL**

SETNUL Achse

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

SETNUL setzt die aktuelle Position der angegebenen Achse zu Null.

**Beispiel**

```
// Ist-Position der Achse 1 auf Null setzen
```

```
SETNUL 1 // Setzt momentane Position der Achse 1 zu Null
```

**Hinweise**

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, dieser Befehl löscht auch den Nullpunkt-Offset (Achsparameter 13).

**Siehe auch**

Positionierbefehle (Seite 41)  
 Achsparameter (Seite 179)

## n SETOVR

SETOVR Achse  $\square$ , Overridewert  $\square$

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

SETOVR verändert den Override-Wert der angegebenen Achse. Der neue Override ist in der angegebenen Variable enthalten. Mit dem Override kann die Verfahrgeschwindigkeit der angegebenen Achse verändert werden: der Override gibt den prozentualen Anteil der tatsächlichen Geschwindigkeit im Verhältnis zur programmierten an.

### Beispiel

```
// Bei der Achse 2 soll während der Positionierung auf die Sollposition 1000
// die Geschwindigkeit in Prozent zur parametrisierten Geschwindigkeit verändert
// werden können.

LAD_VA      V_SOLLP_A2, 1000      // Laden der Sollposition von 10000
LAD_VA      V_GESW_A2, 5000      // laden der Geschwindigkeit von 5000
// in Variable GESCHW_A2
SETVEL      2, V_GESW_A2         // lade Geschwindigkeit für Achse 2
STPABS      2, V_SOLLP_A2        // starte die absolute Positionierung der Achse 1
// mit dem Wert, der in der Variablen SOLLP_A2
// enthalten ist.

LOOP:
SETOVR      2, V_GESW_OVR        // Die Achse 2 beschleunigt auf die neue
// Geschwindigkeit, die in Prozent zur
// parametrisierten Geschwindigkeit in
LAD_M       M_INP_A2             // Variable GESCHW_OVR enthalten ist Abfrage:
// Achse 2 in Position

SPRINGN     LOOP
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.
- n Bitte beachten Sie, daß der Override in den Achsparametern aktiviert sein muß.

### Siehe auch

Positionierbefehle (Seite 41)  
 SETVEL (Seite 119)  
 CHGVEL (Seite 51)  
 CHGVPO (Seite 52)  
 PWRDRV (Seite 102)

## n SETPAR

SETPAR Parameter  $\square$ , Wert  $\square$

Gruppe	Abhängig von BES	Verändert BES
Systembefehle	- Ja	n Nein

SETPAR überträgt den Inhalt der angegebenen Variable in den Parameter.

Wenn Sie Achsparameter beeinflussen möchten, errechnet sich die jeweilige Parameternummer aus der Basisparameternummer - siehe auch Kapitel 5.2 - Achsparameter (ab Seite 179) - und der jeweils betroffenen Achse.

Die entsprechende Berechnungsformel lautet:

$$\Rightarrow (\text{Nummer der Achse} * 100) + \text{Basisparameternummer}$$

### Beispiel

```
// Über das SPS-Programm soll der Parameter (n14) Software-Limit-Plus für
// die Achse 1 geändert werden.
```

```
LAD_VA    V_LIM_A1, 5000    // Laden der Software-Limit-Plus mit 5000
SETPAR    114, V_LIM_A1    // Lade neuen Software-Limit-Plus in den
                          // Servomotorcontroller Achs-Parameternummer
```

### Hinweise

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

### Siehe auch

Systembefehle (Seite 45)

GETPAR (Seite 70)

SETFUN (Seite 112)

System- und Achsparameter (Seite 177)

## n SETRMP

SETRMP Achse  $\square$ , Beschleunigung  $\square$

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

SETRMP setzt die Beschleunigungs- und Bremsrampe der angegebenen Achse. Die Rampe wird auf den in der Variable enthaltenen Wert gesetzt. Die Beschleunigungsrampe entspricht danach dem in der Variable enthaltenen Wert, die Bremsrampe wird automatisch gemäß Parametrierung angepasst.

### Beispiel

```
// Die Achse 2 soll mit einer Beschleunigung von 2000 beschleunigt werden.
```

```
LAD_VA    V_RAMPE_A2, 2000    // Laden der Beschleunigung 2000 in V_RAMPE_A2
SETRMP    2, V_RAMPE_A2    // Lade Beschleunigungsrampe der Achse 2 mit
                          // dem Wert, der in der Variable VRAMPE_A2
                          // enthalten ist
```

### Hinweise

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

n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)

PWRDRV (Seite 102)

## n SETSER

SETSER Modul , Funktion 

Gruppe	Abhängig von BES	Verändert BES
Seriellmodul	- Ja	n Nein

SETSER löst verschiedene Funktionen im Zusammenhang mit dem seriellen Erweiterungsmodul aus. Mit der ersten Konstante wird das zu verwendende Display gewählt. Mit der zweiten Konstante wird die eigentliche Funktion ausgelöst.

Name	Wert	Bedeutung
SEND	1	Text oder Variable ausgeben
SEND_CRLF	2	CR/LF (Zeilenumbruch) ausgeben
SEND_ASCII	3	Einzelnes ASCII-Zeichen ausgeben
SEND_FROM_VAR	4	Text/Zeichenkette aus Variablenfeld ausgeben
SEND_TO_VAR	5	SPS-Text in Variablenfeld kopieren

n Tabelle 28 – Funktionen für SETSER

Reicht der Sendepuffer für dieses serielle Modul nicht aus, um die gewünschten Daten zu speichern, wird automatisch der Systemmerker M\_SEROUT\_OV\_x gesetzt, wobei "x" für die Nummer des seriellen Moduls steht. Der Sendeauftrag wird in diesem Fall vollständig nicht verarbeitet. Bitte beachten Sie, daß dieser Systemmerker von Ihnen im Programm vor dem nächsten Sendeauftrag wieder zurückgesetzt werden muss! Wiederholen Sie das Senden an das serielle Modul solange, bis das System den Fehlermerker M\_SEROUT\_OV\_x nicht mehr setzt.

### Hinweise

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

### Siehe auch

Serielle Anbindung (Seite 44)

Seriellmodul (Seite 171)

LAD\_DT (Seite 72)

LAD\_DV (Seite 73)

## n SETSER SEND

Mit LAD\_DT wird das Format für die Ausgabe festgelegt und in die Variable V\_SERNR die Nummer des entsprechenden Textes oder der Variableninhalt geschrieben. SETDSP ruft das Senden des Textes auf.

### Beispiel

```
LAD_DT    V_SERMSK, 0, 0, 20    // Format festlegen (20 Zeichen Länge)
LAD_VA    V_SERNR, 1           // Text auswählen
SETSER    SER1, SEND           // Text ausgeben
LAD_DV    V_SERMSK, 0, 0, 4    // Format festlegen (4 Zeichen Länge)
LAD_VV    V_SERNR, 200         // Inhalt der Variable 200 ausgeben
SETSER    SER1, SEND           // Zahl ausgeben
```

### Wichtiger Hinweis

Grundsätzlich ist die Funktion SEND des Befehls SETSER identisch mit der Funktion DSP des Befehls SETDSP, d.h. die Programmierung erfolgt wie bei der Displayausgabe. Bitte beachten Sie jedoch, daß einige der Informationen, die Sie mit LAD\_DT bzw. LAD\_DV für die Formatierung angeben, vom Befehl SETSER ignoriert werden. Hierbei handelt es sich um folgende Informationen:

n Angabe der Zeilennummer

n Angabe der Spaltennummer

## n SETSER SEND\_CRLF

Mit dieser Funktion wird die Zeichensequenz CR/LF an das serielle Erweiterungsmodul übertragen. Dies ist vor allem sinnvoll, wenn an das Erweiterungsmodul ein Protokolldrucker angeschlossen ist.

### Beispiel

```
SETSER SER1, SEND_CRLF // CR/LF an Modul 1 ausgeben
```

## n SETSER SEND\_ASCII

Diese Funktion ist nur zusammen mit dem Drucken auf die eingebaute serielle Schnittstelle sinnvoll: hier können Sie ein beliebiges ASCII-Zeichen – auch nicht druckbare – ausgeben. Das auszugebende ASCII-Zeichen wird in V\_SERNR geladen und dann die SETDSP-Funktion ausgeführt.

### Beispiel

```
// In folgendem Beispiel soll ein ETX für "Telegramm Ende" ausgegeben werden.
LAD_VA V_SERNR, 2 // ASCII 02 = ETX
SETSER SER1, SEND_ASCII // ausgeben
```

## n SETSER SEND\_FROM\_VAR

Mit dieser Funktion können Sie einen Text in ein Variablenfeld kopieren. Diese Funktion entspricht prinzipiell dem Befehl SNDSE, nur daß hier die Länge der Daten nicht mit dem ersten Eintrag im Variablenfeld festgelegt wird, sondern durch den Inhalt der Variable V\_SERMSK.

Diese Funktion ist – vor allem auch im Zusammenhang mit der Funktion 5 – sehr flexibel und erlaubt nicht nur das Erstellen von seriellen Protokollen, sondern auch z.B. die Verwaltung von Terminalsteuerungen und ähnlichen interaktiven Funktionen.

Die Nummer der ersten auszugebenden Variable wird in die Variable V\_SERNR eingetragen. Die Anzahl der auszugebenden Zeichen muß in die Variable V\_SERMSK geschrieben werden.

### Beispiel

```
// In diesem Beispiel empfangen wir Daten von einem seriellen Erweiterungsmodul
// und senden diese Daten - versehen mit einem Kommentar - zurück. Wir geben
// davon aus, das Text 10 in diesem Beispiel belegt ist mit "Sie haben geschickt:"
RCVSER 1, 1000 // Empfangsdaten des seriellen Moduls 1 abholen

LAD_VA V_SERNR, 10 // Text 10
LAD_DT V_SERMSK // Vorbelegen für Textausgabe
SETSER SER1, SEND // Text an seriell es Modul ausgeben
LAD_VA V_SERNR, 34 // ASCII-Code für Anführungszeichen
SETSER SER1, SEND_ASCII // Ausgeben
LAD_VA V_SERNR, 1001 // Variablenfeld ab 1001
LAD_VV V_SERMSK, 1000 // Länge der Daten steht in Variable 1000
SETSER SER1, SEND_FROM_VAR // Daten ausgeben
LAD_VA V_SERNR, 34 // ASCII-Code für Anführungszeichen
SETSER SER1, SEND_ASCII // Ausgeben
SETSER SER1, SEND_CRLF // Zeilenvorschub ausgeben
```

## n SETSER SEND\_TO\_VAR

Als Gegenstück zur Funktion SEND\_FROM\_VAR des Befehls SETSER kopiert die Funktion 5 einen definierten SPS-Text in ein Variablenfeld. Dies ermöglicht Ihnen, einen definierten SPS-Text zu verändern oder zu ergänzen, bevor Sie ihn auf der Schnittstelle ausgeben, oder auch den Vergleich eines empfangenen Textes mit einem in der Steuerung gespeicherten.

Tragen Sie in V\_SERNR die Textnummer ein. Schreiben Sie in die Variable V\_SERMSK die Nummer der ersten Variable, in die der Text kopiert werden soll. Nach Ausführung von "SETSER" enthält dann die erste angegebene Variable das erste Zeichen des SPS-Textes, die nächste Variable das zweite usw.

### Beispiel

```
// In diesem Beispiel vergleichen wir einen vom seriellen Erweiterungsmodul
// empfangenen Text mit dem gespeicherten SPS-Text Nr. 12

GEHURPI    SerTest           // Unterprogramm serielles Handling aufrufen
SPRING     Anfang           // Programmschleife

SerTest:
RCVSER     SER1, 1000        // Empfangsdaten in Feld ab Variable 1000
LAD_VA     V_SERNR, 12      // Text 12
LAD_VA     V_SERMSK, 1100   // In Feld ab Variable 1100
SETSER     SER1, SEND_TO_VAR // Kopieren
LAD_VV     V_COUNT, V1000   // Anzahl der Zeichen in V_COUNT kopieren
LAD_VA     V_ZEIGER1, 1001  // Zeigervariable auf erstes Empfangszeichen
LAD_VA     V_ZEIGER2, 1100  // Zeigervariable auf erstes Textzeichen

Loop:
VERG_I I   V_ZEIGER1, V_ZEIGER2 // Zeichen vergleichen
LAD_M      M_GLEICH           // Vergleichsergebnis abfragen
UPRENDN    // Nicht gleich, Unterprogramm Ende
INC_V      V_ZEIGER1, 1       // Nächstes Empfangszeichen
INC_V      V_ZEIGER2, 1       // Nächstes Zeichen des Textes
DEC_V      V_COUNT, 1         // Zeichenzähler herunterzählen
VERG_VA    V_COUNT, 0         // Alles verglichen?
LAD_M      M_GLEICH           // Vergleichsergebnis abfragen
SPRINGN    Loop              // Nein, weiter vergleichen

// Wenn wir im Programm an dieser Stelle ankommen, dann entspricht der
// empfangene Text dem Inhalt des SPS-Textes Nr. 12
```

## n SETVEL

SETVEL Achse , Geschwindigkeit

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

SETVEL setzt die Sollgeschwindigkeit der angegebenen Achse gesetzt auf den in der angegebenen Variable enthaltenen Wert. Die neue Geschwindigkeit ist beim nächsten Start der Achse aktiv. Die Geschwindigkeit der aktuellen Bewegung wird nicht beeinflusst.

### Beispiel

```
// Achse 2 soll beim nächsten Start mit der Geschwindigkeit 10000 fahren.
LAD_VA    V_GESW_A2, 10000    // Laden der Geschwindigkeit von 10000
// in die Variable V_GESW_A2
SETVEL    2, V_GESW_A2        // Lade Verfahrgeschwindigkeit der Achse 2
// mit dem Wert, der in der Variable V_GESW_A2
// enthalten ist
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.
- n Um einen sofortigen Geschwindigkeitswechsel herbeizuführen verwenden Sie bitte CHGVEL.

### Siehe auch

Positionierbefehle (Seite 41)  
 CHGVEL (Seite 51)  
 PWRDRV (Seite 102)

## n SLL\_V

SLL\_V Variable , Wert

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

SLL\_V 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_V     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 36)

## n SLL\_VV

SLL\_VV Variable1 , Variable2 

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 36)

## n SNDSER

SNDSER Modul , Pufferanfang 

Gruppe	Abhängig von BES	Verändert BES
Seriellmodul	- Ja	n Nein

SNDSER überträgt binäre Daten aus einem Variablenfeld zu einem seriellen Erweiterungsmodul. Für jedes zu sendende Byte wird eine Variable verwendet. Die erste hierfür zu verwendende Variable wird durch die Konstante angegeben. Das erste Byte des Variablenfelds enthält die Anzahl der zu übertragenen Daten in Byte, ab der zweiten Variable beginnen die eigentlichen Sendedaten.

### Beispiel

```
LAD_VA    V1000, 3           // 3 Byte übertragen
LAD_VA    V1001, 3           // Telegrammdateien
LAD_VA    V1002, 27          // Telegrammdateien
LAD_VA    V1003, 2           // Telegrammdateien
SNDSER    SER1, V1000        // Daten ab Variable 1000 senden
```

### Hinweise

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

n Die Verwaltung des eigentlichen Sendens erfolgt intern. Falls der Ausgabepuffer (64 Byte) voll ist, wird der Systemmerker M\_SEROUT\_OV\_x gesetzt, wobei x für die Modulnummer steht. Dieser Merker muss vom Programmierer vor der nächsten Datenausgabe mit SNDSER oder SETSER wieder zurückgesetzt werden. Bitte beachten Sie, daß bei einem Überlauf des Puffers das Senden des aktuellen Auftrags nicht gestartet wird, d.h. die Daten werden vollständig nicht gesendet.

### Siehe auch

Serielle Anbindung (Seite 44)

Seriellmodul (Seite 171)





## n SPRING

SPRING Label 

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	y Nein	siehe Text

SPRING setzt den Programmablauf ab dem angegebenen Label fort.

### Operation

Programmadresse  Label - Adresse  
 (Bit ergebnis)  EIN

### Beispiel

```
SPRING    INIT_TIMER           // Springt im Programm zum Label
INIT_TIMER:           // Programm INIT_TIMER
...                 // Programmcode
```

### Hinweise

n Nach der Ausführung des Befehls ist der Bitergebnisspeicher immer eingeschaltet.

### Siehe auch

Programmablaufbefehle (Seite 40)



## n SPRINGJ

SPRINGJ Label 

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	- Ja	siehe Text

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

### Operation

Programmadresse  Label - Adresse  
 (Bit ergebnis)  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.

n Nach der Ausführung des Befehls ist der Bitergebnisspeicher immer eingeschaltet.

### Siehe auch

Programmablaufbefehle (Seite 40)



## n SPRINGN

SPRINGN Label 

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	- Ja	siehe Text

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

### Operation

Programmadresse  Label - Adresse  
(Bitergebnis)  EIN

### Beispiel

```
SPRINGN    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.
- n Nach der Ausführung des Befehls ist der Bitergebnisspeicher immer eingeschaltet.

### Siehe auch

Programmablaufbefehle (Seite 40)


## n SRL\_V

SRL\_V Variable , Wert 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

SRL\_V schiebt den Inhalt der Variable um den angegebenen Wert nach rechts. Das Ergebnis wird in der Ergebnisvariable VARERG gespeichert.

### Operation

(VARERG)  (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_V     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 36)

## n SRL\_VV

SRL\_VV Variable1 , Variable2

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)  $\llcorner$  (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 36)

## n STCONT

STCONT Achse , Richtung

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STCONT startet die angegebene Achse. Die Bewegung erfolgt kontinuierlich in die angegebene Richtung. Mögliche Richtungsangaben sind PLUS und MINUS.

### Beispiel

// Die Achse 1 soll mit der Geschwindigkeit von 3000 in Plusrichtung fahren.

```
LAD_VA    GESW_A1, 3000      // Laden der Geschwindigkeit von 3000
SETVEL    1, GESW_A1        // Sollgeschwindigkeit Achse 1 setzen
STCONT    1, PLUS           // Starte kontinuierlich in Plusrichtung
```

### Hinweise

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

n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)

STPABS (Seite 129)

STPREL (Seite 129)

PWRDRV (Seite 102)

## n STHOME

STHOME Achse 

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STHOME startet eine Referenzfahrt für die angegebene Achse. Das System führt hierbei standardmäßig die für Servomotorcontroller übliche Nullung durch.

Der Ablauf des integrierten Referenzierungsablaufs sieht bei einem MC200 System wie folgt aus:

- n die Achse wird auf den Referenzschalter gefahren
- n der Referenzschalter wird anschließend automatisch freigefahren
- n die Ist-Position der Achse wird auf Null gesetzt

Diese Art der Nullung wird so normalerweise auch dann durchgeführt, wenn ein Schrittmotorcontroller angeschlossen ist. Sie können hier jedoch auch die andere Form der Nullung verwenden, die zumeist als "klassische Nullung" bezeichnet wird.

- n Sie müssen dafür Sorge tragen, daß der verwendete Initiator freigefahren ist.
- n STHOME startet den Nullpunktlauf, bis der Initiator schaltet, und fährt danach die definierte Überlaufstrecke.
- n Sie müssen anschließend mit SETNUL die Ist-Position auf Null setzen.
- n Falls Sie mit einer Nullpunktverschiebung arbeiten, müssen Sie anschließend die Nullpunktverschiebung über den Achsparameter 13 neu setzen.

Um diese Art der Nullung zu verwenden, müssen die Achsparameter entsprechend gesetzt sein. Vom SPS-Programm aus können Sie dies erreichen, indem Sie Bit 5 des Parameters 41 setzen und in Parameter 34 den Überfahrweg speichern.

### Beispiel

```
// Starten einer Referenzfahrt für Achse 1

STHOME      1, NORMAL           // Starten der Referenzfahrt

LOOP:
LAD_M       M_INP_A1           // Abfrage: Achse 1 in Position?
SPRINGN     LOOP              // Warten bis Achse in Position
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)  
 Achsparameter (Seite 179)  
 SETNUL (Seite 113)  
 PWRDRV (Seite 102)

## n STIABS

STIABS Achse , Absolute Position

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STIABS bereitet eine linear interpolierte Bewegung für die angegebene Achse vor. Die Zielposition dieser Bewegung wird durch den Wert der angegebenen Variable definiert. Bei STIABS wird – im Gegensatz zu STIREL – die Zielposition als absoluter Wert angegeben.

Welche Achsen miteinander interpolieren, wird über die Achsparameter definiert. Der Start der Achsen erfolgt erst dann, wenn alle Achsen, die innerhalb einer interpolierten Bewegung miteinander starten sollen, ihre Zielposition erhalten haben. Die Geschwindigkeit der interpolierten Bewegung wird durch die parametrisierte Geschwindigkeit der Achse vorgegeben, die den längsten Weg fahren muß. Alle anderen Achsen passen sich an diese Geschwindigkeitsvorgabe automatisch an, die benötigten Rampenwerte werden dynamisch errechnet.

### Beispiel

```
// Die Achse 1 soll auf die Position 250 und die Achse 2 auf die Position 300 mit
// linearer Interpolation positionieren.
```

```
SETVEL      1, V_GESW_A1      // Lade Verfahrensgeschwindigkeit der Achse 1
SETVEL      2, V_GESW_A2      // Lade Verfahrensgeschwindigkeit der Achse 2
// Die Bewegung wird mit der Geschwindigkeit
// der Achse durchgeführt, die den längsten
// Weg zu fahren hat.

LAD_VA      V_POS_A1, 250     // Zielposition der Achse 1 wird geladen
LAD_VA      V_POS_A2, 300     // Zielposition der Achse 2 wird geladen

STIABS      1, V_POS_A1      // Starte Achse 1 interpoliert
STIABS      2, V_POS_A2      // Starte Achse 2 interpoliert
// Die Bewegung wird jetzt automatisch
// gestartet, weil alle an der Interpolation
// beteiligten Achsen ihre Sollposition
// erhalten haben

LOOP:
LAD_M       M_INP_A1          // Abfrage: Achse 1 in Position
UND_M       M_INP_A2          // Abfrage: Achse 2 in Position
SPRINGN     LOOP
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.
- n Welche Achsen miteinander interpolieren, wird normalerweise in den Achsparametern festgelegt, die Sie mit der PC-Software bearbeiten können. Um die Auswahl der miteinander interpolierenden Achsen zu verändern, können Sie auch den Befehl SETPAR innerhalb des SPS-Programms verwenden.

### Siehe auch

Positionierbefehle (Seite 41)  
 Achsparameter (Seite 179)  
 STIREL (Seite 127)  
 STICIR (Seite 126)  
 PWRDRV (Seite 102)

## n STICIR

STICIR Achse1  $\square$ , Achse2  $\square$

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STICIR startet eine kreisförmig interpolierte Bewegung der angegebenen Achsen. Für die zirkulare Interpolation werden die Mittelpunkte, die beiden Endpunkte der Achsen auf dem Kreisbogen benötigt sowie die Kreisrichtung als zusätzliche Parameter benötigt. Aufgrund der flexiblen Parametrierung können Sie nicht nur Kreise, sondern auch Ausschnitte des Kreisbogens fahren.

Die zirkulare Interpolation ist mit Sicherheit einer der komplexeren Befehle innerhalb der MC-1B Sprache. Da für diesen Befehl insgesamt 7 Parameter benötigt werden, die Struktur der MC-1B Sprache jedoch nur eine begrenzte Anzahl Parameter zuläßt, müssen die Informationen vor dem Auslösen des Interpolationsbefehls in entsprechende Systemvariablen- und Merkern gespeichert werden:

- n M\_CICR\_DIR enthält die Richtung der Bewegung: wenn ausgeschaltet, läuft die Achse im Uhrzeigersinn; wenn eingeschaltet, läuft die Achse gegen den Uhrzeigersinn.
- n V\_CIRC\_MIDX enthält den Mittelpunkt des Kreises für die X-Achse.
- n V\_CIRC\_MIDY enthält den Mittelpunkt des Kreises für die Y-Achse.
- n V\_CIRC\_ENDX enthält den Endpunkt des Kreisbogens für die X-Achse
- n V\_CIRC\_ENDY enthält den Endpunkt des Kreisbogens für die Y-Achse

### Beispiel

Im folgenden Beispiel sollen die Achsen 2 und 3 miteinander zirkular interpolieren. Hierfür werden die beiden Mittelpunkte und Endpunkte auf der Kreisbahn benötigt.

```
EIN_M      M_CICR_DIR           // Legt die Richtung fest: wenn eingeschaltet,
// dann Bewegung gegen den Uhrzeigersinn
LAD_VA     V_CIRC_MIDX, 400 // Mittelpunkt des Kreisbogens für die X-Achse
LAD_VA     V_CIRC_MIDY, 200 // Mittelpunkt der Kreisbogens für die Y-Achse
LAD_VA     V_CIRC_ENDX, 200 // Endpunkt auf dem Kreisbogen für die X-Achse
LAD_VA     V_CIRC_ENDY, 0   // Endpunkt auf dem Kreisbogen für die Y-Achse
STICIR     2, 3             // Startet zirkulare Interpolation
```

### Tips

- n Der Kreisradius ergibt sich aus der Differenz zwischen der aktuellen Position der Achse und dem angegebenen Kreismittelpunkt. Umgekehrt bedeutet dies, daß Sie den Kreismittelpunkt berechnen können, indem Sie die Differenz zwischen der aktuellen Position und dem gewünschten Kreisradius bilden.
- n Wenn Sie einen vollständigen Kreis fahren möchten, geben Sie als Endposition der jeweiligen Achse auf dem Kreisbogen die aktuelle Position plus/minus einer Maßeinheit an.
- n Das System führt automatisch eine elliptische Bewegung durch, wenn die Kreisradien für die X- und Y-Achse sich unterscheiden.

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.
- n Anders bei den linearen Interpolationsbefehlen STIABS und STIREL werden bei der zirkularen Interpolation die beteiligten Achsen nicht über die Achsparameter angegeben, sondern direkt als Parameter beim Startbefehl übergeben. Sie können also eine Zirkularinterpolation durchführen, ohne daß Sie hierbei zuvor Achsparameter beeinflussen müssen.

**Siehe auch**

Positionierbefehle (Seite 41)

Achsparemeter (Seite 179)

STIABS (Seite 125)

STIREL (Seite 127)

PWRDRV (Seite 102)

**n STIREL**STIREL Achse , Relative Position 

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STIREL bereitet eine linear interpolierte Bewegung für die angegebene Achse vor. Die Zielposition dieser Bewegung wird durch den Wert der angegebenen Variable definiert. Bei STIREL wird – im Gegensatz zu STIABS – die Zielposition als relativer Wert angegeben.

Welche Achsen miteinander interpolieren, wird über die Achsparemeter definiert. Der Start der Achsen erfolgt erst dann, wenn alle Achsen, die innerhalb einer interpolierten Bewegung miteinander starten sollen, ihre Strecke erhalten haben. Die Geschwindigkeit der interpolierten Bewegung wird durch die parametrisierte Geschwindigkeit der Achse vorgegeben, die den längsten Weg fahren muß. Alle anderen Achsen passen sich an diese Geschwindigkeitsvorgabe automatisch an, die benötigten Rampenwerte werden dynamisch errechnet.

**Beispiel**

```
// Die Achse 1 soll um die Strecke 250 und die Achse 2 soll um die Strecke 300 mit
// linearer Interpolation positionieren.
```

```
LAD_VA    V_GESW_A1, 2000    // Laden der Geschwindigkeit 2000 für Achse 1
LAD_VA    V_GESW_A2, 3000    // Laden der Geschwindigkeit 3000 für Achse 2
LAD_VA    V_STRE_A1, 2500    // Speichern des Fahrweges in V_POS_A1
LAD_VA    V_STRE_A2, 3000    // Speichern des Fahrweges in V_POS_A2
SETVEL    1, V_GESW_A1       // Lade Verfahrgeschwindigkeit der Achse 1
SETVEL    2, V_GESW_A2       // Lade Verfahrgeschwindigkeit der Achse 2
STIREL    1, V_STRE_A1       // Lade relative Strecke für Achse 1
STIREL    2, V_STRE_A2       // Lade relative Strecke für Achse 2 und starte
// die lineare Interpolation

LOOP:
LAD_M     M_INP_A1           // Abfrage: Achse 1 in Position
UND_M     M_INP_A2           // Abfrage: Achse 2 in Position
SPRINGN   LOOP
```

**Hinweise**

- n** Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n** Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.
- n** Welche Achsen miteinander Interpolieren, wird normalerweise in den Achsparemetern festgelegt, die Sie mit der PC-Software bearbeiten können. Um die Auswahl der miteinander interpolierenden Achsen zu verändern, können Sie auch den Befehl SETPAR innerhalb des SPS-Programms verwenden.

**Siehe auch**

Positionierbefehle (Seite 41)

Achsparemeter (Seite 179)

STIABS (Seite 125)

STICIR (Seite 126)

PWRDRV (Seite 102)

**n STOPDN**STOPDN Achse 

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STOPDN stoppt die angegebene Achse mit parametrierter Bremsrampe.

**Beispiel**

```
// Die Achse 1 soll beim Erreichen von Eingang 1 (E_STOP) gestoppt werden.
```

```
LOOP:
```

```
LAD_E      E_STOP      // Abfrage: Eingang Stop erreicht?
```

```
STOPDN     1           // Stoppe Achse 1
```

```
SPRINGN    LOOP
```

**Hinweise**

- n Dieser Befehl ist abhängig vom Bitergebnisspeicher und wird nur bei eingeschaltetem Bitergebnis ausgeführt.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

**Siehe auch**

Positionierbefehle (Seite 41)

STOPEM (Seite 128)

PWRDRV (Seite 102)

**n STOPEM**STOPEM Achse 

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STOPEM stoppt die angegebene Achse mit maximaler Rampe. Diese Funktion wird normalerweise bei einem Notstop verwendet.

**Beispiel**

```
// Die Achse 1 soll mit maximaler Bremsrampe gestoppt werden, wenn Merker M_STOP
// eingeschaltet wird.
```

```
LOOP:
```

```
LAD_M      M_STOP      // Abfrage: Merker Stop eingeschaltet?
```

```
NUND_M     M_HLP       // Abfrage: Hilfsmerker nicht gesetzt?
```

```
STOPEM     1           // Stoppe Achse 1 mit maximaler Rampe
```

```
EIN_M      M_HLP       // Hilfsmerker setzen
```

```
LAD_M      M_INP_A1    // Abfrage: Achse 1 in Position
```

```
SPRINGN    LOOP
```

**Hinweise**

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

**Siehe auch**

Positionierbefehle (Seite 41)

STOPDN (Seite 128)

PWRDRV (Seite 102)



## n STPABS

STPABS Achse , Absolute Position

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STIABS startet die angegebene Achse. Die absolute Zielposition ist in der angegebenen Variable enthalten.

### Beispiel

// Die Achse 1 soll auf die Position 250 positionieren.

```
LAD_VA    V_SOLLP_A1, 250           // Lade Sollposition in die Variable SOLLP_A1
LAD_VA    V_GESW_A1, 5000          // Lade Verfahrgeschwindigkeit in V_GESW_A1
SETVEL    1, V_GESW_A1              // Sollgeschwindigkeit für Achse 1 setzen
STPABS    1, V_SOLLP_A1             // Starte Achse 1 auf die Sollposition,
// die in Variable SOLLP_A1 enthalten ist

LOOP:
LAD_M     M_INP_A1                  // M_INP_A1 ist eingeschaltet, sobald die
SPRINGN   LOOP                      // Achse ihre Position erreicht hat
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)

PWRDRV (Seite 102)

## n STPREL

STPREL Achse , Relative Position

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STIREL startet die angegebene Achse. Die relative Zielposition ist in der angegebenen Variable enthalten.

### Beispiel

// Die Achse 2 soll um die Strecke -500 verfahren werden

```
LAD_VA    V_STRE_A2, -500           // Lade Strecke in die V_STRE_A2
LAD_VA    V_GESW_A2, 6000          // Lade Verfahrgeschwindigkeit V_GESW_A2
SETVEL    2, V_GESW_A2              // Sollgeschwindigkeit für Achse 2 setzen
STPREL    2, V_STRE_A2             // Die Achse 2 verfährt die relative
// Strecke, die in Variable STRE_A2
// enthalten ist
```

### Hinweise

- n Dieser Befehl ist abhängig vom Bitergebnisspeicher und wird nur bei eingeschaltetem Bitergebnis ausgeführt.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden müssen, bevor Sie eine Achspositionierung auslösen können.

### Siehe auch

Positionierbefehle (Seite 41)

PWRDRV (Seite 102)

## n STPRLD

STPRLD Achse  $\square$

Gruppe	Abhängig von BES	Verändert BES
Positionierung	- Ja	n Nein

STPRLD startet die angegebene Achse auf die mit PRLABS oder PRLREL vorgeladene Position.

### Beispiel

// Die Achse 2 soll mit den vorgeladenen Daten gestartet werden.

```
LAD_VA      V_SOLLP_A1, 3250      // Lade Sollposition in die Variable SOLLP_A1
LAD_VA      V_GESW_A1, 5000     // Lade Verfahrensgeschwindigkeit in V_GESW_A1
SETVEL      1, V_GESW_A1       // Laden der Sollgeschwindigkeit für Achse 1
PRLABS      1, V_SOLLP_A1      // Lade die absolute Position der Achse mit dem
// Wert, der Variablen V_SOLLP_A1 vor
STPRLD      1                  // Starte Achse 1 mit den vorgeladenen Daten
```

### Hinweise

- n Dieser Befehl wird nur ausgeführt wenn der Bitergebnisspeicher eingeschaltet ist.
- n Bitte beachten Sie, daß Sie die Leistungsteile der Achse mit PWRDRV freigeschaltet werden, bevor Sie eine Achspositionierung auslösen.
- n Dieser Befehl wird automatisch ausgeführt, wenn Sie den automatischen Start der Achse über die Sonderfunktion AUTOSTART aktiviert haben. Eine Beschreibung finden Sie beim Befehl SETFUN.

### Siehe auch

Positionierbefehle (Seite 41)  
 PRLABS (Seite 100)  
 PRLREL (Seite 101)  
 PWRDRV (Seite 102)  
 SETFUN (Seite 112)

## n SUB\_II

SUB\_II Zeiger1  $\square$ , Zeiger2  $\square$

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

SUB\_II subtrahiert vom Inhalt der durch Zeiger 1 bestimmten Variable den Inhalt der durch Zeiger2 bestimmten Variable. Das Ergebnis der Subtraktion wird in VARERG gespeichert.

### Operation

(VARERG)  $\ominus$  (Zeiger1 à Variable) - (Zeiger2 à Variable)

(M\_GLEICH)  $\ominus$  (Ergebnis) = 0

(M\_KLEINER)  $\ominus$  (Ergebnis) < 0

(M\_GROESSER)  $\ominus$  (Ergebnis) > 0

### Beispiel

```
SUB_II      V_ZEIGER1, V_ZEIGER2 // Subtrahiert von dem Inhalt der Variable, auf
// die V_ZEIGER1 zeigt, den Inhalt der Variable
// auf die V_ZEIGER2 zeigt.
```

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)

## n SUB\_IV

SUB\_IV Zeiger  $\overline{V}$ , Variable  $\overline{V}$

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)  $\zeta$  (Zeiger  $\bar{a}$  Variable) - (Variable)

(M\_GLEICH)  $\zeta$  (Ergebnis) = 0

(M\_KLEINER)  $\zeta$  (Ergebnis) < 0

(M\_GROESSER)  $\zeta$  (Ergebnis) > 0

### 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 36)

## n SUB\_VA

SUB\_VA Variable  $\overline{V}$ , Wert  $\overline{K}$

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)  $\zeta$  (Variable) - Wert

(M\_GLEICH)  $\zeta$  (Ergebnis) = 0

(M\_KLEINER)  $\zeta$  (Ergebnis) < 0

(M\_GROESSER)  $\zeta$  (Ergebnis) > 0

### 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 36)

**n SUB\_VI**SUB\_VI Variable , Zeiger 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

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

**Operation**

(VARERG)  $\zeta$  (Variable) - (Zeiger  $\rightarrow$  Variable)

(M\_GLEICH)  $\zeta$  (Ergebnis) = 0

(M\_KLEINER)  $\zeta$  (Ergebnis) < 0

(M\_GROESSER)  $\zeta$  (Ergebnis) > 0

**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 36)

**n SUB\_VV**SUB\_VV Variable1 , Variable2 

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)  $\zeta$  (Variable1) - (Variable2)

(M\_GLEICH)  $\zeta$  (Ergebnis) = 0

(M\_KLEINER)  $\zeta$  (Ergebnis) < 0

(M\_GROESSER)  $\zeta$  (Ergebnis) > 0

**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 36)

## 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- 1B " ;
TEXT004 "*****" ;
TEXT005 "Version 1.0 30.10.98" ;
TEXT006 "TEXT006 " ;
TEXT007 " " ;
TEXT008 "TEXT008 " ;
```

### Sonderzeichen im Text

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

- n Zeilenumbruch (CRLF): Zeichenfolge "\n", z.B. "Auswertung\n"
- n Zeilenvorschub (LF): Zeichenfolge "\r", z.B. "Neue Zeile\r"
- n 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 hexadezimaler 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

- n Sie können auch mehrere Dateien zur Definition von Texten verwenden.
- n 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.
- n 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.
- n 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

Systembefehle (Seite 45)  
 Display und Texte (Seite 43)  
 Display-Programmierung (Seite 160)  
 Display-Zeichentabelle (Seite 256)

**n UND\_A****UND\_A Ausgang** 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\&$  (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 34)

Wahrheitstabelle (Seite 255)

**n UND\_E****UND\_E Eingang** 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> 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)  $\&$  (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 34)

Wahrheitstabelle (Seite 255)

## n UND\_II

UND\_II Zeiger1 , Zeiger2 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

UND\_II verknüpft die Inhalte der durch Zeiger1 und Zeiger2 bestimmten Variablen mit einem binären UND. Das Ergebnis wird in der Ergebnisvariablen VARERG gespeichert.

### Operation

(VARERG)  $\Leftarrow$  (Zeiger1  $\rightarrow$  Variable) & (Zeiger2  $\rightarrow$  Variable)

### Beispiel

// In diesem Beispiel verknüpfen wir den Inhalt der ersten Tabelle, die bei  
// Variable 200 beginnt, mit dem Inhalt der zweiten Tabelle, die bei Variable 400  
// beginnt. Das Verknüpfungsergebnis wird wieder in der ersten Tabelle gespeichert.

```

DEF_W      200, TABELLE1           // Anfang der ersten Tabelle
DEF_W      400, TABELLE2           // Anfang der zweiten Tabelle
DEF_V      200, ANZ_GLEICH          // Vergleichszähler definieren
LAD_VA     V_ZEIGER1, TABELLE1     // V_ZEIGER1 mit Originaltabelle laden
LAD_VA     V_ZEIGER2, TABELLE2     // V_ZEIGER2 mit Zieltable laden
LAD_VA     V_ZAEHLER, 99           // Wir wollen 99 Variablen vergleichen

LOOP:
UND_II     V_ZEIGER2, V_ZEIGER1    // Erste Tabelle mit zweiter verknüpfen
LAD_IV     V_ZEIGER1, VARERG        // Ergebnis der Verknüpfung in Tabelle schreiben
INC_V      V_ZEIGER1, 1             // Zeiger auf TABELLE1 erhöhen
INC_V      V_ZEIGER2, 1             // Zeiger auf TABELLE2 erhöhen
DEC_V      V_ZAEHLER, 1             // Zähler für Vergleichen verringern
VERG_VA    V_ZAEHLER, 0            // Bereits alles verglichen?
LAD_M      M_GLEICH                 // Vergleichsergebnis abfragen
SPRINGN    LOOP                    // Noch nicht alles kopiert, weiter

```

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)

**n UND\_IV**UND\_IV Zeiger  $\overline{M}$ , Variable  $\overline{M}$ 

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)  $\overline{C}$  (Zeiger  $\overline{a}$  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 36)

**n UND\_M**UND\_M Merker  $\overline{M}$ 

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	y Nein	p 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)  $\overline{C}$  (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 33)

Wahrheitstabelle (Seite 255)



## n UND\_VA

UND\_VA Variable  $\square$ , Wert  $\square$

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)  $\zeta$  (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 36)

## n UND\_VI

UND\_VI Variable  $\square$ , Zeiger  $\square$

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)  $\zeta$  (Variable) & (Zeiger  $\rightarrow$  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 36)

**n UND\_VV**UND\_VA Variable1  $\boxtimes$ , Variable2  $\boxtimes$ 

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)  $\boxtimes$  (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**

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

**Siehe auch**

Variablenbefehle (Seite 36)

**n UPREND**

UPREND

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	y Nein	p Ja

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

**Operation**

Programmadresse  $\boxtimes$  (Stackpointer)  
 Stackpointer  $\boxtimes$  Stackpointer + 2  
 (Bitergebnis)  $\boxtimes$  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**

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

n Der Bitergebnisspeicher ist der Rückkehr aus dem Unterprogramm eingeschaltet.

**Siehe auch**

Programmablaufbefehle (Seite 40)

**n UPRENDJ****UPRENDJ**

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	- Ja	<b>p</b> Ja

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

**Operation**

Programmadresse  $\zeta$  (Stackpointer)  
 Stackpointer  $\zeta$  Stackpointer + 2  
 (Bitergebnis)  $\zeta$  EIN

**Beispiel**

```

GEHUPR    UPROG           // Ruft Unterprogramm UPROG auf
SPRING    LOOP

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

**Hinweise**

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

**Siehe auch**

Programmablaufbefehle (Seite 40)

**n UPRENDN****UPRENDN**

Gruppe	Abhängig von BES	Verändert BES
Programmablauf	- Ja	<b>p</b> Ja

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

**Operation**

Programmadresse  $\zeta$  (Stackpointer)  
 Stackpointer  $\zeta$  Stackpointer + 2  
 (Bitergebnis)  $\zeta$  EIN

**Beispiel**

```

GEHUPR    UPROG           // Ruft Unterprogramm UPROG auf
SPRING    LOOP

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

**Hinweise**

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

**n** Der Bitergebnisspeicher ist nach der Rückkehr aus dem Unterprogramm eingeschaltet.

**Siehe auch**

Programmablaufbefehle (Seite 40)

## n VERG\_II

VERG\_II Zeiger1  $\overline{V}$ , Zeiger2  $\overline{V}$

Gruppe	Abhängig von BES	Verändert BES
Variablenvergleich	y Nein	n Nein

VERG\_II vergleicht die Inhalte der durch Zeiger1 und Zeiger2 bestimmten Variablen. Das Ergebnis wird in den Variablenvergleichsmerkern gespeichert:

Merker	Wird gesetzt wenn...
M_GLEICH	...die Inhalte der durch Zeiger1 und Zeiger2 bestimmten Variablen gleich sind
M_KLEINER	...der Inhalt der durch Zeiger1 bestimmten Variable kleiner ist als der Inhalt der durch Zeiger2 bestimmten Variable
M_GROESSER	...der Inhalt der durch Zeiger1 bestimmten Variable größer ist als der Inhalt der durch Zeiger2 bestimmten Variable

n Tabelle 29 – Variablenvergleichsmerker bei VERG\_II

### Operation

(M\_GLEICH)  $\zeta$  (Zeiger1  $\hat{a}$  Variable) = (Zeiger2  $\hat{a}$  Variable)  
 (M\_KLEINER)  $\zeta$  (Zeiger1  $\hat{a}$  Variable) < (Zeiger2  $\hat{a}$  Variable)  
 (M\_GROESSER)  $\zeta$  (Zeiger1  $\hat{a}$  Variable) > (Zeiger2  $\hat{a}$  Variable)

### Beispiel

```
// Dieses Beispiel vergleicht die Inhalte zweier Tabellen und speichert die
// Anzahl übereinstimmender Einträge in V_ANZ_GLEICH.

LAD_VA    V_ZEIGER1, 200           // V_ZEIGER1 mit Originaltabelle laden
LAD_VA    V_ZEIGER2, 400           // V_ZEIGER2 mit Zieltabelle laden
LAD_VA    V_ZAEHLER, 99            // Wir wollen 99 Variablen vergleichen
LAD_VA    V_ANZ_GLEICH, 0          // Zähler initialisieren

LOOP:
VERG_II   V_ZEIGER2, V_ZEIGER1     // Variable der ersten Tabelle mit Variable der
// zweiten Tabelle vergleichen
LAD_M     M_GLEICH                  // Übereinstimmender Eintrag gefunden?
SPRINGJ   VERGLEICH                // Wenn ja, Springe zum Label "Fertig"

WEITER:
INC_V     V_ZEIGER1, 1              // Zeiger auf TABELLE1 erhöhen
INC_V     V_ZEIGER2, 1              // Zeiger auf TABELLE2 erhöhen
DEC_V     V_ZAEHLER, 1              // Zähler für Vergleichen verringern
LAD_M     M_GLEICH                  // Vergleichsergebnis abfragen
SPRINGN   LOOP                     // Noch nicht alles kopiert, weiter

VERGLEICH:
INC_V     V_ANZ_GLEICH, 1           // Vergleichszähler erhöhen
SPRING    WEITER                   // Rücksprung zum Programm
```

### Hinweise

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

### Siehe auch

Variablenvergleichsbefehle (Seite 39)  
 Ergebnismerker- und Variablen (Seite 195)

## n VERG\_IV

VERG\_IV Zeiger , Variable 

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 30 – Variablenvergleichsmerker bei VERG\_IV

**Operation**(M\_GLEICH)     $\Leftrightarrow$  (Zeiger  $\hat{a}$  Variable) = (Variable)(M\_KLEINER)     $\Leftrightarrow$  (Zeiger  $\hat{a}$  Variable) < (Variable)(M\_GROESSER)     $\Leftrightarrow$  (Zeiger  $\hat{a}$  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-Variablen1 initialisieren
LAD_VA     V_ZEIGER2, TABELLE2      // Zeiger-Variablen2 initialisieren

LOOP:
VERG_IV    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 39)

Ergebnismerker- und Variablen (Seite 195)

**n VERG\_VA**VERG\_VA Variable  $\mathbb{V}$ , Wert  $\mathbb{K}$ 

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 31 – Variablenvergleichsmerker bei VERG\_VA

**Operation**(M\_GLEICH)  $\zeta$  (Variable) = (Wert)(M\_KLEINER)  $\zeta$  (Variable) < (Wert)(M\_GROESSER)  $\zeta$  (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 39)

Ergebnismerker- und Variablen (Seite 195)

## n VERG\_VI

VERG\_VI Variable  $\bar{V}$ , Zeiger  $\bar{V}$ 

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 32 – Variablenvergleichsmerker bei VERG\_VI

**Operation**

(M\_GLEICH)     $\bar{C}$  (Variable) = (Zeiger  $\bar{a}$  Variable)  
(M\_KLEINER)     $\bar{C}$  (Variable) < (Zeiger  $\bar{a}$  Variable)  
(M\_GROESSER)     $\bar{C}$  (Variable) > (Zeiger  $\bar{a}$  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-Variablen1 initialisieren
LAD_VA    V_ZEIGER2, TABELLE2    // Zeiger-Variablen2 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 39)  
Ergebnismerker- und Variablen (Seite 195)

## n VERG\_VV

VERG\_VV Variable1  $\bar{V}$ , Variable2  $\bar{V}$

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 grösser ist als der Inhalt der Variable2

n Tabelle 33 – Variablenvergleichsmerker bei VERG\_VV

### Operation

(M\_GLEICH)  $\bar{C}$  (Variable1) = (Variable2)

(M\_KLEINER)  $\bar{C}$  (Variable1) < (Variable2)

(M\_GROESSER)  $\bar{C}$  (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 39)

Ergebnismerker- und Variablen (Seite 195)



## 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)  $\zeta$  (Adresse des letzten WART\_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...WART\_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 40)

**n XODER\_A****XODER\_A Ausgang** 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> Ja

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

**Operation**

(Bitergebnis)  $\oplus$  (Bitergebnis)  $\wedge$  (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**

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

**Siehe auch**

Ein-/Ausgangsbefehle (Seite 34)

Wahrheitstabelle (Seite 255)

**n XODER\_E****XODER\_E Eingang** 

Gruppe	Abhängig von BES	Verändert BES
I/O Befehle	<b>y</b> Nein	<b>p</b> Ja

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

**Operation**

(Bitergebnis)  $\oplus$  (Bitergebnis)  $\wedge$  (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**

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

**Siehe auch**

Ein-/Ausgangsbefehle (Seite 34)

Wahrheitstabelle (Seite 255)

## n XODER\_M

XODER\_M Merker  $\overline{M}$

Gruppe	Abhängig von BES	Verändert BES
Merkerbefehle	y Nein	p Ja

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

### Operation

(Bitergebnis)  $\oplus$  (Bitergebnis)  $\wedge$  (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 33)

Wahrheitstabelle (Seite 255)

## n XOR\_II

XOR\_II Zeiger1  $\overline{M}$ , Zeiger2  $\overline{M}$

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

XOR\_II verknüpft den Inhalt der durch die Zeiger bestimmten Variablen mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.

### Operation

(VARERG)  $\oplus$  (Zeiger1  $\rightarrow$  Variable)  $\wedge$  (Zeiger2  $\rightarrow$  Variable)

### Beispiel

```
// In diesem Beispiel verknüpfen wir den Inhalt zweier Tabellen. Das jeweilige
// Verknüpfungsergebnis wird wieder in der ersten Tabelle gespeichert.

LAD_VA     V_ZEIGER1, 200     // V_ZEIGER1 mit Originaltabelle laden
LAD_VA     V_ZEIGER2, 400     // V_ZEIGER2 mit Zieltabelle laden
LAD_VA     V_ZAEHLER, 99      // Wir wollen 99 Variablen vergleichen

LOOP:
XOR_II     V_ZEIGER2, V_ZEIGER1 // Variable der beiden Tabellen verknüpfen
LAD_IV     V_ZEIGER1, VARERG   // Ergebnis der Verknüpfung in Tabelle schreiben
INC_V      V_ZEIGER1, 1        // Zeiger auf TABELLE1 erhöhen
INC_V      V_ZEIGER2, 1        // Zeiger auf TABELLE2 erhöhen
DEC_V      V_ZAEHLER, 1        // Zähler für Vergleichen verringern
LAD_M      M_GLEICH           // Bereits alles verglichen?
SPRINGN    LOOP               // Noch nicht alles kopiert, weiter
```

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)

**n XOR\_IV**XOR\_IV Zeiger  $\mathbb{V}$ , Variable  $\mathbb{V}$ 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

XOR\_IV verknüpft den Inhalt der durch die Zeiger bestimmten Variable und der angegebenen Variable mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.

**Operation**

$$(\text{VARERG}) \text{ ☞ } (\text{Zeiger} \text{ à } \text{Variable}) \wedge (\text{Variable})$$
**Beispiel**

```
// In diesem Beispiel haben wir ein Variablenfeld, welches z.B. vom PC geschrieben
// wird, und die unteren 16 Bit mit einer Variablen. Das Ergebnis der Verknüpfung
// wird in der Tabelle gespeichert.
```

```
LAD_VA    V_ZEIGER, 2000    // Zeiger-Variable initialisieren
LAD_VA    V_MASKE, 0xFFFF  // Zum Maskieren mit XOR
```

```
LOOP:
```

```
XOR_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 36)

**n XOR\_VA**XOR\_VA Variable  $\mathbb{V}$ , Wert  $\mathbb{K}$ 

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

XOR\_VA verknüpft den Inhalt der Variable und den angegebenen Wert mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.

**Operation**

$$(\text{VARERG}) \text{ ☞ } (\text{Variable}) \wedge (\text{Wert})$$
**Beispiel**

```
// In diesem Beispiel verwenden wir den Befehl LAD_VM, um 32 Merker in einer
// Variable zu transportieren. Die Variable wird mit einer Konstanten
// Exklusiv-ODER-Verknüpft.
```

```
LAD_VM    V_DATEN, ERSTER_MERKER // Übertragen von 32 Merkern in die Variable
UND_VA    V_DATEN, 0xFFFF        // mit hex FFFF werden die unteren 16 Bit
// (=Merker) werden verknüpft
```

**Hinweise**

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

**Siehe auch**

Variablenbefehle (Seite 36)

## n XOR\_VI

XOR\_VI Variable  $\bar{V}$ , Zeiger  $\bar{V}$

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

XOR\_VI verknüpft den Inhalt der angegebenen Variable und der durch den Zeiger bestimmten Variable mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.

### Operation

(VARERG)  $\bar{C}$  (Variable)  $\wedge$  (Zeiger  $\bar{a}$  Variable)

### Beispiel

```
// In diesem Beispiel haben wir ein Variablenfeld, welches z. B. vom PC geschrieben
// wird, und verknüpfen die unteren 16 Bit über eine Exklusiv-ODER-Verknüpfung mit
// FFFFh. Das wird wieder in der Tabelle gespeichert
```

```
LAD_VA    V_ZEIGER, 2000           // Zeiger-Variable initialisieren
LAD_VA    V_MASKE, 0xFFFF         // entspricht hexadezimal FFFF
LOOP:
XOR_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 36)

## n XOR\_VV

XOR\_VV Variable1  $\bar{V}$ , Variable2  $\bar{V}$

Gruppe	Abhängig von BES	Verändert BES
Variablenbefehle	- Ja	n Nein

XOR\_VV verknüpft den Inhalt der beiden angegebenen Variablen mit einem binären Exklusiv-ODER (XOR). Das Ergebnis der Verknüpfung wird im Ergebnisspeicher VARERG abgelegt.

### Operation

(VARERG)  $\bar{C}$  (Variable1)  $\wedge$  (Variable2)

### Beispiel

```
// In diesem Beispiel verwenden wir den Befehl LAD_VM, um 32 Merker in einer
// Variable zu transportieren. Die Variable wird mit einer zweiten Variable
// Exklusiv-ODER-Verknüpft.
```

```
LAD_VA    V_MASKE, 0xFFFF         // hex FFFF zum Ausmaskieren der oberen Bits
LAD_VM    V_DATEN, ERSTER_MERKER  // Übertragen von 32 Merkern in die Variable
XOR_VV    V_DATEN, V_MASKE        // Verknüpfung der unteren 16 Bit (= Merker)
```

### Hinweise

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

### Siehe auch

Variablenbefehle (Seite 36)

**n #ELSE****#ELSE**

Gruppe	Abhängig von BES	Verändert BES
Compilierung	<b>y</b> Nein	<b>n</b> 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 32)

**n #ENDIF****#ENDIF**

Gruppe	Abhängig von BES	Verändert BES
Compilierung	<b>y</b> Nein	<b>n</b> 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 32)

## n #IF

#IF Abfragebedingung   
 #IF Wert1  EQ Wert2 

Gruppe	Abhängig von BES	Verändert BES
Compilierung	y Nein	n Nein

#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 34 – 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 32)

**n #INCLUDE**#INCLUDE Quellcodedateiname 

Gruppe	Abhängig von BES	Verändert BES
Compilierung	<b>y</b> Nein	<b>n</b> 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 32)



**n Raum für Ihre Notizen**

**n Raum für Ihre Notizen**

# Kapitel 4 Besondere Anwendungen

---

In diesem Kapitel finden Sie Beschreibungen zur Programmierung spezieller Module, wie z.B. eines Displays, der seriellen Erweiterungsmodule oder eines analogen Ein-/Ausgangsmoduls. Auch für die Einbindung von Sonderfunktionen – wie z.B. der integrierten Meßfunktionen – in Ihr Projekt finden Sie die entsprechenden Beschreibungen.

In den folgenden Abschnitten finden Sie Erläuterungen zu diesen Anwendungen:

- n Kapitel 4.1 - Messfunktionen (Seite 156)
- n Kapitel 4.2 - Display-Programmierung (Seite 160)
- n Kapitel 4.3 - Analog Ein-/ Ausgänge (Seite 164)
- n Kapitel 4.4 - Timer (Seite 166)
- n Kapitel 4.5 - Serielles Modul (Seite 171)
- n Kapitel 4.6 - EAU-T Emulation (Seite 174)

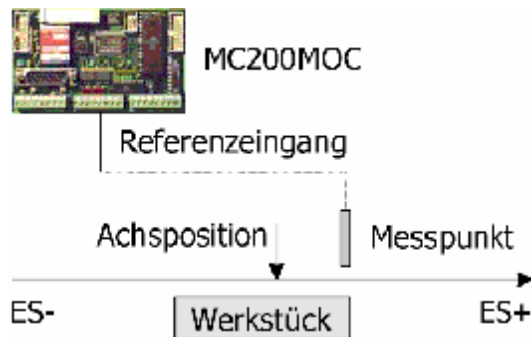
## 4.1 Messfunktionen

Das MC200-System unterstützt schnelle Meßfunktionen, mit denen Sie z.B. eine fliegende Messung realisieren können. Die entsprechenden Meßeingänge sind auf den Achskontrollern untergebracht: es wird hier stets der Referenz-Eingang des Achsmoduls verwendet, um eine schnelle Messung durchzuführen. Aufgrund dieser technischen Lösung können Sie natürlich nicht mehr mit einem getrennten Referenzschalter arbeiten, da der Referenz-Eingang für die Meßfunktion benötigt wird. Wir empfehlen in diesem Fall, den Referenzschalter auf einen der beiden Endschalter-Eingänge zu legen und in der Achsparametrierung den entsprechenden Endschalter zum Referenzschalter umzudefinieren.

### Beispiel für eine Meßkonfiguration

Über eine Lichtschranke soll während der Bewegung der Versatz eines Werkstückes gemessen werden. Hierzu wird eine Lichtschranke installiert, die auf den Referenz-Eingang des Achskontrollers verdrahtet wird. Der Endschalter Minus wird nun als Referenzschalter definiert. Dazu sind in der Achsparametrierung folgende Einstellungen notwendig:

- n Endschalter Minus muß als "nicht vorhanden" markiert werden, weil dieser Endschalter ja nun als Referenzschalter benutzt werden soll. Würde der Endschalter Minus in dieser Konfiguration als "vorhanden" markiert werden, würde jedes Nullen der Achse eine Störung auslösen.
- n In den Einstellungen für die Referenzfahrt muß markiert werden: Endschalter Minus wird als Referenzschalter benutzt.



n Abbildung 2 – Aufbau Messfunktion

### Vorgehensweise

Wie aus der Abbildung ersichtlich, lässt sich – vorausgesetzt, die Länge des Werkstücks ist bekannt – aus der Differenz zwischen Messpunkt und Achsposition ermitteln, an welcher Stelle des Werkstücks der Greifer liegt, d.h. welcher Versatz vorhanden ist.

$$\Rightarrow (\text{Position des Messpunkts}) - (\text{Gemessene Position}) = (\text{Versatz des Werkstücks})$$

## n Ablauf der Messung

Der Ablauf der Messung wird intern vom Achskontroller selbst gehandhabt. Sobald ein neuer Meßwert vorliegt, wird für das SPS-Programm ein entsprechender Merker gesetzt:

Merker-Nummer	Name	Bedeutung
2113	M_SPEC DAT_A1	Neue Meßdaten für Achse 1 verfügbar
2114	M_SPEC DAT_A2	Neue Meßdaten für Achse 2 verfügbar
2115	M_SPEC DAT_A3	Neue Meßdaten für Achse 3 verfügbar
2116	M_SPEC DAT_A4	Neue Meßdaten für Achse 4 verfügbar
2117	M_SPEC DAT_A5	Neue Meßdaten für Achse 5 verfügbar
2118	M_SPEC DAT_A6	Neue Meßdaten für Achse 6 verfügbar
2119	M_SPEC DAT_A7	Neue Meßdaten für Achse 7 verfügbar
2120	M_SPEC DAT_A8	Neue Meßdaten für Achse 8 verfügbar
2121	M_SPEC DAT_A9	Neue Meßdaten für Achse 9 verfügbar
2122	M_SPEC DAT_A10	Neue Meßdaten für Achse 10 verfügbar
2123	M_SPEC DAT_A11	Neue Meßdaten für Achse 11 verfügbar
2124	M_SPEC DAT_A12	Neue Meßdaten für Achse 12 verfügbar
2125	M_SPEC DAT_A13	Neue Meßdaten für Achse 13 verfügbar
2126	M_SPEC DAT_A14	Neue Meßdaten für Achse 14 verfügbar
2127	M_SPEC DAT_A15	Neue Meßdaten für Achse 15 verfügbar
2128	M_SPEC DAT_A16	Neue Meßdaten für Achse 16 verfügbar

n Tabelle 35 – Merker für Meßfunktion

In Ihrem SPS-Programm würden Sie prüfen, ob der entsprechende Merker für die Meßfunktion gesetzt ist und dann das Auslesen des gemessenen Wertes starten. Bitte beachten Sie, daß Sie diesen Merker in Ihrem SPS-Programm zurücksetzen müssen; die Steuerung selbst setzt diese Merker nur beim Einschalten der Steuerung zurück.

### Auslesen des Meßwertes

Der jeweils aktuelle Meßwert wird im Parameter 5 des Achskontrollers gespeichert. Sie können den letzten Meßwert dann über die Funktion GETPAR auslesen.

#### Beispiel

```

SETFUN      1, AFUN_MSR_ON           // Meßfunktion einschalten
LAD_VA      V_ZIELPOS, 2000         // Zielposition laden
STPABS      1, V_ZIELPOS            // Achse 1 auf Zielposition starten

LOOP:
LAD_M       M_SPEC DAT_A1           // Warten, daß neue Daten verfügbar sind
SPRINGN     LOOP                   // keine neuen Daten...
AUS_M       M_SPEC DAT_A1           // Merker löschen
GETPAR      105, V_OFFSET           // Gemessene Achsposition bei E4 laden
...         // zuletzt geschaltet hat
...         // Danach kann nun der Meßwert für die Berechnung
...         // von Werkzeugkorrekturen verwendet werden

```

## n Auswerten von mehreren Meßwerten

Sie haben nicht nur die Möglichkeit – wie oben beschrieben – einen einzelnen Meßwert auszulesen und zu verrechnen, sondern Sie können auch ganze Blöcke von Meßwerten in von Ihnen gewählte Variablenbereiche ablegen lassen und zu einem späteren Zeitpunkt auswerten.

### Auswahl der Variablenbereiche

Innerhalb des MC200 Systems gibt es zwei voneinander getrennte Datenbereiche, in die Sie Meßdaten schreiben lassen können. Dies bedeutet: Sie können entweder gleichzeitig Meßdaten für zwei unterschiedliche Achsen speichern lassen, oder Sie können bei einer Achse unterschiedliche Meßabläufe speichern.

Um nun den Variablenbereich festzulegen, speichern Sie in V\_DIAGBEG die Nummer der ersten Variable, die Sie für die Speicherung der Meßdaten benutzen möchten. Hiermit definieren Sie dann den Start des ersten Puffers. Dieser Puffer reicht bis zum Beginn des zweiten Diagnosepuffers, der über die Variable V\_DIAGBEG2 festgelegt wird. Der zweite Puffer reicht per Definition bis zum Ende des Variablenbereiches.

### Beispiel für das Setzen der Puffer

```
LAD_VA    V_DIAGBEG, 7500    // Erster Diagnosepuffer ab Variable 7500
LAD_VA    V_DIAGBEG2, 7800  // Zweiter Diagnosepuffer ab Variable 7800
// Damit wird automatisch das Ende des ersten
// Puffers auf die Variable 7799 gesetzt
```

### Kennzeichnung des letzten Meßwertes

Die Variablen V\_DIAGPOI und V\_DIAGPOI2 zeigen jeweils auf die nächste freie Position im jeweiligen Puffer. Es sind also die Zeigervariablen für die Meßdatenauswertung. Der letzte gespeicherte Meßwert liegt immer eine Variable vor dem Inhalt von V\_DIAGPOI bzw. V\_DIAGPOI2.

Sie können diese Zeigervariablen auch jederzeit aus Ihrem SPS-Programm heraus verändern, um z.B. eine neue Meßdatenreihe zu beginnen. Wenn Sie die Anfangsadressen der Puffer verändern, müssen Sie sogar diese Zeiger auf den jeweilige Startadresse legen:

```
LAD_VA    V_DIAGBEG, 7500    // Erster Diagnosepuffer ab Variable 7500
LAD_VV    V_DIAGPOI, V_DIAGBEG // Zeiger auf ersten Eintrag des Diagnosepuffers
LAD_VA    V_DIAGBEG2, 7800  // Zweiter Diagnosepuffer ab Variable 7800
// Damit wird automatisch das Ende des ersten
// Puffers auf die Variable 7799 gesetzt
LAD_VV    V_DIAGPOI2, V_DIAGBEG2 // Zeiger auf ersten Eintrag des Diagnosepuffers
```

Mit jedem neuen Meßwert wird der Zeiger um eins erhöht. Sobald das Ende des Puffers erreicht ist, beginnt die Aufzeichnung wieder am Anfang des Puffers und überschreibt so die ältesten aufgezeichneten Meßwerte. Man nennt dies auch eine "Ring-Puffer" Funktion.

### Zuordnen der Puffer zu den Achsen

Damit die Meßdaten der Achskontroller automatisch in den entsprechenden Puffer geschrieben werden, müssen Sie natürlich dem System mitteilen, welcher Puffer für welche Achse verwendet wird. Dies geschieht über den Systemparameter 57:

- n Parameter 57 = 0: Schreiben der Meßfunktion in Puffer deaktiviert
- n Parameter 57 = 1: Meßdaten dieser Achse werden in Puffer 1 geschrieben
- n Parameter 57 = 2: Meßdaten dieser Achse werden in Puffer 2 geschrieben

Gleichzeitig müssen Sie mit SETFUN Achse, AFUN\_MSR\_ON die Meßfunktion einschalten

## Starten der Auswertung

Nachdem Sie alle Parameter gesetzt haben und die Startadressen für die Diagnosepuffer definiert haben, können Sie mit Hilfe des Merkers M\_DIAGS die Speicherung der Meßdaten in diesen Puffern ein- oder ausschalten. Wenn der Merker M\_DIAGS gesetzt ist, werden die Meßdaten in die Puffer übertragen. Ist der Merker M\_DIAGS ausgeschaltet, erfolgt keine Speicherung der Meßdaten in die definierten Puffern.

## Beispiel

```
// In diesem Beispiel reservieren wir den Speicher von Variable 7000 bis
// Variable 7799 für Achse 1, und den Speicher von Variable 7800 bis
// Variable 8192 für Achse 2. Wir weisen die Puffer über die Achsparameter
// zu, schalten den Merker M_DIAGS ein und starten die Positionierung.

LAD_VA    V_DIAGBEG, 7000           // Erster Diagnosepuffer ab Variable 7000
LAD_VV    V_DIAGPOI, V_DIAGBEG     // Zeiger auf erste Variable setzen
LAD_VA    V_DIAGBEG2, 7800         // Zweiter Diagnosepuffer ab Variable 7800
LAD_VV    V_DIAGPOI2, V_DIAGBEG2   // Zeiger auf erste Variable setzen

LAD_VA    V_PUFFER, 1              // Auswahl des ersten Puffers
SETPAR    157, V_PUFFER            // Zuordnen des Puffers zu Achse 1
LAD_VA    V_PUFFER, 2              // Auswahl des zweiten Puffers
SETPAR    257, V_PUFFER            // Zuordnen des Puffers zu Achse 2
SETFUN    1, AFUN_MSR_ON           // Meßfunktion für Achse 1 aktivieren
SETFUN    2, AFUN_MSR_ON           // Meßfunktion für Achse 2 aktivieren

EIN_M     M_DIAGS                  // Aufzeichnung der Meßdaten einschalten

STPABS    1, 6000                  // Achse 1 starten
STPABS    2, 4000                  // Achse 2 starten

LOOP:                                           // Warten bis beide Achsen in Position
LAD_M     M_INPOS_A1
UND_M     M_INPOS_A2
SPRINGN   LOOP

AUS_M     M_DIAGS                  // Aufzeichnung der Meßdaten ausschalten

// Jetzt zeigt V_DIAGPOI auf die erste noch nicht beschriebene Variable des
// ersten Puffers, und V_DIAGPOI2 auf die erste noch nicht beschriebene
// Variable des zweiten Puffers. Sie können nun die aufgezeichneten Meßdaten
// mit den normalen Variablenfunktionen auswerten.
```

## Siehe auch

SETFUN, Liste der Systemvariablen

## 4.2 Display-Programmierung

Mit der MC200-Familie wird eine Vielzahl unterschiedlicher Displays und damit auch teilweise unterschiedlicher Programmiervarianten angeboten. Zudem kann bei jeder Tastatur und jedem Handbedienteil eine individuelle Beschriftung und Zuordnung der Tasten erfolgen. Diese Flexibilität wird natürlich durch einen gewissen Mehraufwand bei der Programmierung der Displays bezahlt. In diesem Abschnitt zeigen wir Ihnen, wie Sie dennoch schnell und erfolgreich ein Display programmieren.

### n SPS-Texte

Texte werden innerhalb des SPS-Projekts meistens in einer separaten Datei angelegt, da das Schlüsselwort "TEXT" im Programm bedeutet, daß alle nachfolgenden Zeilen bis zum Dateiende SPS-Texte enthalten. Jede tatsächliche Text ab dem Befehl "TEXT" muss hierbei in Anführungszeichen eingeschlossen werden, alles andere wird als Kommentar behandelt und ignoriert. Dies können Sie für sich nutzen, indem Sie z.B. die Texte am Beginn einer Zeile durchnummern.

#### Texte und Textnummern

Texte beginnen stets mit dem SPS-Text Nr. 1, d.h. der erste Text nach dem Schlüsselwort "TEXT" trägt die Nummer 1. Der darauffolgende Text wird der Nummer 2 zugeordnet usw. Lücken in dieser Nummerierung sind nicht möglich, der Textnummer 10 kann also nie die Textnummer 20 folgen.

Sie haben auf diese Nummerierung keinerlei Einfluß. Deshalb sollten Sie darauf achten, Ihre Texte entsprechend zu kennzeichnen, um Verwechslungen auszuschließen. Hier ein Beispiel:

```
TEXT // ab jetzt alles Texte
T0001 "Dies ist ein Test "
T0002 "Noch ein Test... "
T0003 " " // reserviert für später
T0004 "Letzter Text, Nr. 4"
```

Die eingefügte Nummerierung in obigem Beispiel wird vom Compiler einfach ignoriert, hilft Ihnen aber, zu jedem Text gleich die richtige Nummer zu finden.

#### Textlängen

Im MC200 System ist jeder Text stets 20 Zeichen lang. Wenn Sie einen längeren Text eingeben, wird der Rest ignoriert. Wenn Sie einen kürzeren Text eingeben, wird der Text automatisch auf 20 Zeichen erweitert.

Um mit diesen vorgegebenen Längen eine ansprechende Benutzeroberfläche zu erstellen ist es ratsam, eine Art "Lineal" zu verwenden, welches Sie selbst im Quellcode einfügen:

```
TEXT // ab jetzt alles Texte
Lineal 12345678901234567890
T0001 "Version: 1. 20. 4"
T0002 "Datum: 26. 08. 99"
T0003 "Autor: Michael Folz"
```

#### Zeichensätze

Die MC200 Familie verwendet für das Display einen IBM-kompatiblen 8-Bit Zeichensatz. Dieser Zeichensatz ist vom PC her auch bekannt als "OEM-Zeichensatz" oder als "Codepage 370". Der OEM-Zeichensatz ist nicht kompatibel mit dem Windows-Zeichensatz, der auch oft "ANSI-Zeichensatz" genannt wird. Während der Programmierung brauchen Sie sich hierum nicht zu kümmern, weil die MICRO DESIGN Programmiersoftware sämtliche Texte automatisch in den für die Steuerung benötigten Zeichensatz umwandelt.

Von der Verwendung von Sonderzeichen, wie z.B. grafische Zeichen oder Symbole, wird jedoch dringend abgeraten. Es kann nicht gewährleistet werden, daß diese Zeichen hinterher im Display genauso aussehen wie in der Entwicklungsoberfläche. Eine MC200-Zeichentabelle finden Sie im Anhang B - Display-Zeichentabelle (Seite 256) dieser Dokumentation.



## n Texte formatieren und darstellen

Mit dem MC200 System können Sie Texte frei auf dem verfügbaren Platz des Displays positionieren. Dies erfolgt innerhalb der MC-1B Sprache durch eine entsprechende Formatierungsanweisung vor der eigentlichen Displayausgabe. Aber auch wenn Sie keine spezielle Formatierung wünschen, beachten Sie bitte:

**Vor jeder Textausgabe muss die Darstellung mit LAD\_DT ausgewählt werden!**

Nur so ist gewährleistet, daß auch tatsächlich der Text so dargestellt wird, wie Sie es wünschen. Anschließend tragen Sie die Text-Nummer in V\_ANZNRx ein, wobei x für die Displaynummer steht:

```
LAD_DT    V_ANMSK1, 0, 0           // Text in erster Zeile, Position 1
LAD_VA    V_ANZNR1, 10            // Text Nr. 10
SETDSP    1, 2                    // Ausgeben in großer Schrift
LAD_DT    V_ANZMSK1, 3, 9, 5      // Vierte Zeile, Position 10, 5 Zeichen lang
LAD_VA    V_ANZNR1, 13            // Text Nr. 13
SETDSP    1, 1                    // Ausgeben in normaler Schrift
```

Wann immer Sie bei dem Befehl LAD\_DV nicht alle möglichen Parameter angeben, werden die übrigen Werte mit den Standardvorgaben gefüllt. Der Befehl

```
LAD_DT    V_ANZMSK1
```

trägt folgende Werte ein:

- ⇒ Textausgabe in Zeile 1, Spalte 1
- ⇒ 20 Zeichen Länge

## n Zahlenwerte formatieren und darstellen

Genauso wie Texte können auch Zahlenwerte ausgegeben werden. Auch hier müssen Sie eine Formatierung auswählen, die Ihnen jedoch noch mehr Gestaltungsfreiraum lässt als bei Texten:

**Vor jeder Zahlenausgabe muss die Darstellung mit LAD\_DV ausgewählt werden!**

Nur so ist gewährleistet, daß auch tatsächlich der Wert so dargestellt wird, wie Sie es wünschen. Dann tragen Sie den darzustellenden Wert in die Variable V\_ANZNRx, wobei x für die Nummer des Displays steht, ein:

```
LAD_DV    V_ANMSK1, 1, 14, 5, 2   // Text in zweiter Zeile, Position 15, 5 Zeichen
                                                // lang, 2 Kommastellen, kein Vorzeichen
LAD_VV    V_ANZNR1, V210           // Inhalt der Variable 210 darstellen
SETDSP    1, 1                    // Ausgeben in normaler Schrift
LAD_DV    V_ANMSK1, 2, 9, 10      // Text in dritter Zeile, Position 10, 10 Zeichen
                                                // lang, keine Kommastellen, kein Vorzeichen
LAD_VA    V_ANZNR1, 555           // Konstanten Wert 555 darstellen
SETDSP    1, 1                    // Ausgeben in normaler Schrift
```

Wann immer Sie bei dem Befehl LAD\_DV nicht alle möglichen Parameter angeben, werden die übrigen Werte mit den Standardvorgaben gefüllt. Der Befehl

```
LAD_DV    V_ANZMSK1
```

trägt also folgende Werte ein:

- ⇒ Zahlenausgabe in Zeile 1, Spalte 1
- ⇒ 20 Zeichen Länge
- ⇒ Keine Kommastellen
- ⇒ Kein Vorzeichen

## n Zahlenwerte editieren

Natürlich können Sie mit der MC-1B Programmiersprache auch Zahlenwerte durch den Anwender editieren bzw. verändern lassen. Hierzu formatieren Sie – genau wie bei der Ausgabe von Zahlenwerten – zunächst die Darstellung des Wertes.

### Vor jeder Eingabe muss die Darstellung mit LAD\_DV ausgewählt werden!

Nur so ist gewährleistet, daß auch tatsächlich der Text so dargestellt wird, wie Sie es wünschen. Dann tragen Sie den editierenden Wert in die Variable V\_INPNRx, wobei x für die Nummer des Displays steht, ein:

```
LAD_DV      V_INPMSK1, 1, 14, 5, 2      // Text in zweiter Zeile, Position 15, 5 Zeichen
// lang, 2 Kommastellen, kein Vorzeichen
LAD_VV      V_INPNR1, V210              // Inhalt der Variable 210 editieren
SETEDI      1, 1                        // Editor aktivieren
```

Die Editierfunktion selbst ist im Betriebssystem der MC200 Familie integriert und wird mit dem Befehl "SETEDI 1,1" bzw. "SETEDI 2,1" aktiviert. Jetzt kann der Wert durch den Benutzer editiert werden. Bitte beachten Sie jedoch:

### Während der Editierung läuft das SPS-Programm weiter!

Die Programmausführung bleibt also nicht in der Zeile stehen, in der Sie den Editor aufrufen. Die ermöglicht Ihnen, Editierungen auch während dem normalen Betrieb zuzulassen, da Sie weiterhin Ihr Programm abarbeiten und auch alle Zustände der Anlage kontrollieren können.

Natürlich müssen Sie nun prüfen, wann die Editierung beendet wurde. Dies üblicherweise gelöst, indem Sie eine Taste für die Bestätigung der Eingabe und eine Taste für den Abbruch der Eingabe vorsehen und, nachdem die Editierung gestartet wurde, einfach prüfen, ob eine dieser Tasten gedrückt wurde:

```
LAD_M      M_KEY_ENTER                  // Enter-Taste gedrückt?
ODER_M     M_KEY_ESCAPE                 // Escape-Taste gedrückt?
UPRENDN                                         // Nein, Editierung läuft weiter
SETEDI     1, 0                         // Editor abschalten
LAD_M      M_KEY_ENTER                  // Eingabe mit Enter bestätigt?
UPRENDN                                         // Nein, abgebrochen, Wert nicht übernehmen
LAD_VV     V210, V_INPNR1               // Editierten Wert in V210 übernehmen
URPEND                                         // Ende
```

Bitte beachten Sie, daß:

- n Jede Editierung mit "SETEDI 1,0" abgeschlossen werden muss,
- n nach Abschluß der Editierung der bearbeitete Wert immer in V\_INPNRx zurückgespeichert wird, gleichgültig, mit welcher Taste die Editierung beendet wurde.

## n Tastaturbelegung

Um die unterschiedlichen Displays, die für die MC200 Familie verfügbar sind, gleichermaßen zu unterstützen, müssen Sie am Anfang Ihres SPS-Programms die entsprechende Tastaturbelegung auswählen. Zwar werden alle "normalen" Tasten in Merkern gespiegelt – und können dementsprechend von Ihnen flexibel abgefragt werden – doch ist die Editierfunktion im Betriebssystem der MC200 integriert und benötigt eine Angabe zur gewünschten Tastenbelegung.

Eine Beschreibung der verfügbaren Tastenbelegungen finden beim Befehl SETEDI (Seite 110), eine Übersicht aller Tastenmerker im Kapitel 6.14 - Displayprogrammierung (Seite 226).

## n Die Format-Variablen V\_ANZMSK/V\_INPMSK

In den Format-Variablen V\_ANZMSKx bzw. V\_INPMSKx, wobei x hier für das jeweilige Display steht, wird die von Ihnen gewählte Formatierung gespeichert. Wann immer Sie also den Befehl LAD\_DT oder LAD\_DV schreiben, verändert die Steuerung nach einem bestimmten Algorithmus die Variable V\_ANZMSKx (für Ausgaben) bzw. V\_INPMSKx (für Eingaben). Sie können sich dies zu Nutzen machen, wenn Sie z.B. die Anzeigeparameter dynamisch verändern möchten: die Befehle LAD\_DT und LAD\_DV lassen als Parameter nur Konstanten zu, weshalb es Ihnen nicht möglich ist, hier z.B. die Anzeigeposition einer Variable während dem Programmablauf dynamisch zu verändern.

Jedoch bleibt Ihnen hier die Möglichkeit, den Inhalt der Variable V\_ANZMSKx bzw. V\_INPMSKx direkt zu verändern. Dafür sind jedoch Kenntnisse im Aufbau von binären Zahlenwerten notwendig:

Bit...	31	30	29	28	27	24-26	16-23	8-15	0-7
LAD_DT	Ein	Ni/A	Invertiert	Doppelte Größe	N/A	N/A	Spalte (erste=0)	Zeile (erste=0)	Länge
LAD_DV	Aus	N/A	Invertiert	Doppelte Größe	Vorzeichen	Nachkommastellen	Spalte (erste=0)	Zeile (erste=0)	Länge

n Tabelle 36 – Formatvariablen V\_ANZMSK/V\_INPMSK

Um jetzt dynamisch während der Laufzeit ein Anzeigeformat festzulegen, müssen Sie sich die entsprechende Formatvariable selbst zusammenbauen. Das funktioniert dann – hier im Beispiel für LAD\_DV – in etwa so:

```

LAD_VA    V_ANZMSK1, (gewünschte Länge)
LAD_VA    VARERG, (Anzahl Nachkommastellen; 0=keine)
SLL_V     VARERG, 24 // Nachkommastellen 24 Bit nach links schieben
ODER_VV   VARERG, V_ANZMSK1 // In Formatvariable hineinmaskieren
LAD_VV    V_ANZMSK1, VARERG // In Formatvariable speichern
LAD_VA    VARERG, (Spalte - 1)
SLL_V     VARERG, 16 // Spaltenposition um 16 Bit nach links schieben
ODER_VV   VARERG, V_ANZMSK1 // In Formatvariable hineinmaskieren
LAD_VV    V_ANZMSK1, VARERG // In Formatvariable speichern
LAD_M     (Vorzeichen erwünscht?)
ODER_VA   V_ANZMSK1, 0x08000000 // Falls Vorzeichen gewünscht, dazumaskieren
LAD_VV    V_ANZMSK1, VARERG // In Formatvariable speichern
LAD_M     M_EIN // BES wieder einschalten
LAD_VA    VARERG, (Zeilennummer - 1)
SLL_V     VARERG, 8 // Zeilennummer um 4 nach links
ODER_VV   VARERG, V_ANZMSK1 // In Formatvariable hineinmaskieren
LAD_VV    V_ANZMSK1, VARERG // Ergebnis in Formatvariable speichern

// Folgendes entspricht LAD_DT V_ANZMSK1, ZEILE2, 4, 10
LAD_VA    VARERG, 10 // Länge: 10
SLL_V     VARERG, 24 // 24 Bit nach links
LAD_VV    V_ANZMSK1, VARERG // In Formatvariable speichern
LAD_VA    VARERG, 3 // Spalte: (4-1) = 3
SLL_V     VARERG, 16 // 16 Bit nach links
ODER_VV   V_ANZMSK1, VARERG // Mit Formatvariable maskieren
LAD_VV    V_ANZMSK1, VARERG // In Formatvariable speichern
LAD_VA    VARERG, 1 // Zeile: (2-1) = 1
SLL_V     VARERG, 8 // 8 Bit nach links
ODER_VV   V_ANZMSK1, VARERG // Mit Formatvariable maskieren
ODER_VA   VARERG, 0x80000000 // Kennung für LAD_DT dazumaskieren
LAD_VV    V_ANZMSK1, VARERG // In Formatvariable speichern

```

## 4.3 Analog Ein-/ Ausgänge

### n Schreiben von analogen Ausgängen

Mit dem Befehl SETAIO wird der Inhalt der Variable an ein analogen Ausgang übergeben. Bei dem Sollwert handelt es sich um einen 12 Bit Wert, der also Werte zwischen 0 und 4095 enthalten kann. Wird ein Wert größer als 4095 oder kleiner als 0 angegeben, so wird stets der maximale Analogwert (4095) ausgegeben.

Bitte beachten Sie auch die Beschreibung des Befehls SETAIO (Seite 103).

### n Lesen von analogen Eingängen

Für das Lesen von analogen Eingängen steht kein separater SPS-Befehl zur Verfügung. Stattdessen können Sie den jeweils aktuellen Meßwert direkt aus einer Systemvariable lesen. Folgende Systemvariablen sind hierfür reserviert:

Funktion	Name	Adresse
Analog Modul 1, Eingang 1 (12 Bit 0-4095)	V_ANAINP1_1	Variable 71
Analog Modul 1, Eingang 2 (12 Bit 0-4095)	V_ANAINP1_2	Variable 72
Analog Modul 1, Eingang 3 (12 Bit 0-4095)	V_ANAINP1_3	Variable 73
Analog Modul 1, Eingang 4 (12 Bit 0-4095)	V_ANAINP1_4	Variable 74
Analog Modul 2, Eingang 1 (12 Bit 0-4095)	V_ANAINP2_1	Variable 75
Analog Modul 2, Eingang 2 (12 Bit 0-4095)	V_ANAINP2_2	Variable 76
Analog Modul 2, Eingang 3 (12 Bit 0-4095)	V_ANAINP2_3	Variable 77
Analog Modul 2, Eingang 4 (12 Bit 0-4095)	V_ANAINP2_4	Variable 78
Analog Modul 3, Eingang 1 (12 Bit 0-4095)	V_ANAINP3_1	Variable 79
Analog Modul 3, Eingang 2 (12 Bit 0-4095)	V_ANAINP3_2	Variable 80
Analog Modul 3, Eingang 3 (12 Bit 0-4095)	V_ANAINP3_3	Variable 81
Analog Modul 3, Eingang 4 (12 Bit 0-4095)	V_ANAINP3_4	Variable 82
Analog Modul 4, Eingang 1 (12 Bit 0-4095)	V_ANAINP4_1	Variable 83
Analog Modul 4, Eingang 2 (12 Bit 0-4095)	V_ANAINP4_2	Variable 84
Analog Modul 4, Eingang 3 (12 Bit 0-4095)	V_ANAINP4_3	Variable 85
Analog Modul 4, Eingang 4 (12 Bit 0-4095)	V_ANAINP4_4	Variable 86

n Tabelle 37 – Analoge Ein- und Ausgänge

### n Mittelwertbildung

Das MC200AIO Modul bildet automatisch einen Mittelwert der letzten vier eingelesenen analogen Eingangswerte. Dieser Mittelwert wird als Eingangswert an das CPU-Modul übertragen und in den oben beschriebenen Variablen zur Verfügung gestellt.

Durch diese Mittelwertbildung werden grobe Messungenauigkeiten egalisiert, die z.B. durch die eingesetzten Messaufnehmer oder durch Fehler in der Datenübermittlung zum MC200AIO Modul entstehen können.

## n Werteaktualisierung

Das CPU-Modul der MC200 Familie holt neue analoge Meßwerte in einem fest definierten Intervall vom analogen Modul ab. Dies bedeutet, daß die Analogwerte zwar in Echtzeit aufgezeichnet, verarbeitet und gemittelt werden, jedoch erst mit gewisser Verzögerung in der CPU verfügbar sind. Damit Sie innerhalb des SPS-Programms eine Auswertung programmieren können, die nur neu gelesene analoge Werte berücksichtigt, wird für jedes analoge Modul ein Merker gesetzt, sobald neue Daten verfügbar sind.

Das System setzt diesen Merker nur, wenn Sie anhand des Merkers Auswertungen steuern, sollten Sie ihn nach jeder Bearbeitung im Programm zurücksetzen.

Funktion	Name	Adresse
Neue Daten für Analog Modul 1 verfügbar	M_ANAINP1	Merker 2209
Neue Daten für Analog Modul 2 verfügbar	M_ANAINP2	Merker 2210
Neue Daten für Analog Modul 3 verfügbar	M_ANAINP3	Merker 2211
Neue Daten für Analog Modul 4 verfügbar	M_ANAINP4	Merker 2212

n Tabelle 38 – Merker für analoge Werteaktualisierung

## n Umwandlung Analog/Digital

Welchem tatsächlichen Analogwert der jeweilige Zahlenwert von 0 bis 4095 entspricht hängt von der Konfiguration Ihres Analogmoduls ab. Die A/D-Wandler auf dem Modul können in sehr unterschiedlichen Konfigurationen geschaltet werden.

Wertebereich	Wert "0" entspricht	Wert "4095" entspricht
0 bis 10 VDC	0 VDC	10 VDC
-10 bis 10 VDC	-10 VDC	10 VDC
0 bis 20 mA	0 mA	20 mA

n Tabelle 39 – Umsetzung analoge auf digitale Werte

## 4.4 Timer

Das MC200 System verfügt über insgesamt 32 integrierte Systemtimer, die Sie in Ihren SPS-Programmen frei verwenden können. 30 dieser Timer arbeiten mit einer Auflösung von 1/10s, zwei spezielle Timer – die Timer 7 und 8 - mit der höheren Auflösung von 1/100s. Eine Übersicht der entsprechenden Systemvariablen- und Merker finden Sie Kapitel 6.9 - Systemtimer (Seite 220).

### Wozu verwendet man Timer?

Die im MC200 System integrierten Timer ermöglichen Ihnen eine genaue zeitgesteuerte Programmierung einzelner Funktionen und Abläufe. So verwendet man die Systemtimer häufig, um z.B. eine zeitgesteuerte Fehlerüberwachung zu programmieren. Das typische Beispiel hierfür ist natürlich der Zylinder: mit einem Timer können Sie einfach kontrollieren, ob der Zylinder in einer von Ihnen zuvor festgelegten Zeit Endlage erreicht hat. Natürlich lassen sich Timer noch für viele andere Funktionen verwenden, sei es, um Zykluszeiten zu messen, zeitversetzte Achsenstarts zu realisieren und vieles mehr.

### n Die integrierten Systemtimer

Wie bereits erwähnt, verfügt das MC200 System über 32 fest integrierte Timer. Diese Timer sind mit den symbolischen Namen V\_TIM\_1 bis V\_TIM\_32 bezeichnet. Ein zusätzliches Flag, daß Sie über den Zustand des Timers informiert, ist jeweils in den Merkern M\_TIM\_1 bis M\_TIM\_32 gespeichert.

### Aufbau des Timers

Jeder Timer besteht aus einer Variable und einem Merker: die Variable gibt vor dem Start des Timers den Initialisierungswert vor, also wie lange der Timer laufen soll. Der Merker startet den Timer mit dem Initialisierungswert und wird vom System zurückgesetzt, sobald der Timer abgelaufen ist.

### Programmieren eines Timers

Schalten Sie als erstes einmal den Timer-Merker für den von Ihnen verwendeten Timer aus. Falls der Timer nämlich gegenwärtig von einem vorherigen Start noch aktiv ist, würden Sie Ungenauigkeiten provozieren:

```
{ AUS_M      M_TIM_1          // Timer 1 stoppen
```

Schreiben Sie in die Timervariable, die Sie verwenden möchten, den gewünschten Startwert für den Timer, also z.B.

```
{ LAD_VA     V_TIM_1, 100      // Timer 1 mit 100 x 1/10s = 10s laden
```

**Bitte beachten Sie, daß die Timer 1-6 sowie 9-32 mit einer Auflösung von 1/10s, die Timer 7 und 8 jedoch mit einer Auflösung von 1/100s arbeiten. Wenn Sie also in den Timer 7 den Wert 100 laden, dann entspricht dies nicht  $100 \times 1/10s = 10s$ , sondern vielmehr  $100 \times 1/100s = 1s$ .**

Schalten Sie jetzt den Timer-Merker für den entsprechenden Timer ein. Der Timer läuft dann automatisch los:

```
{ EIN_M      M_TIM_1          // Timer 1 starten
```

Sie können permanent – also auch während der Timer läuft – den aktuellen Timerwert über die Variable V\_TIM\_1 abfragen. Sobald diese Variable den Wert 0 erreicht hat, ist der Timer abgelaufen. Zur einfacheren Kontrolle in Ihrem SPS-Programm wird automatisch der entsprechende Timermerker – in diesem Beispiel M\_TIM\_1 – mit ausgeschaltet.

**Beispiel**

```
// In diesem Beispiel kontrollieren wir die Funktion eines Zylinders.
// Wir setzen den Ausgang A_ZYL, um den Zylinder in die Endlage zur bringen
// und kontrollieren über den Eingang E_ZYL_END, ob der Zylinder die Endlage
// erreicht hat. Gleichzeitig starten wir einen Timer für 10 x 1/10s = 1s.
// Hat der Zylinder in diesem Zeitraum von einer Sekunde die Endlage nicht
// erreicht, dann betrachten wir dies als Störung.

EIN_A      A_ZYL          // Zylinder in Endlage fahren lassen

AUS_M      M_TIM_1       // Timer 1 stoppen
LAD_VA     V_TIM_1, 100  // Timer 1 wird mit dem Wert 100 geladen
EIN_M      M_TIM_1       // Timer 1 wird gestartet

Loop:      // Wir prüfen jetzt, ob der Zylinder die
           // Endlage erreicht hat, oder der Timer
           // abgelaufen ist.

LAD_E      E_ZYL_END     // Hat Zylinder Endlage erreicht?
SPRINGJ    AllesOk      // Ja, weiter
           // Zylinder hat die Endlage noch nicht erreicht.

NLAD_M     M_TIM_1       // Ist der Timer noch nicht abgelaufen?
SPRINGN    Loop         // Nein, dann warten wir noch
SPRING     Stoerung     // Timer abgelaufen, Stoerung
```

**Vordefinierte Makros**

In der Datei MC200.MAC, die automatisch zu jedem MC200-Projekt hinzugefügt wird, sind auch zusätzliche Makros zur Kontrolle der Timer enthalten. Falls Sie also lieber mit Makros programmieren, statt die Timer-Variablen und –Merker direkt zu beschreiben, können Sie folgende Makros verwenden:

Makro	Arg 1	Arg 2	Bedeutung
TSTART	Konstante	Konstante	Startet den in dem ersten Parameter angegebenen Timer mit dem Ausgangswert, der über den zweiten Parameter angegeben wird.
TSTART_V	Konstante	Variable	Startet den in dem ersten Parameter angegebenen Timer mit dem Ausgangswert, der über den Wert der Variable bestimmt wird.
TCONTR	Konstante		Falls der als Parameter angegebene Timer abgelaufen ist, wird das Bitergebnis eingeschaltet.
AUS_TI	Konstante		Stoppt den als Parameter angegebenen Timer.
UND_TI	Konstante		Führt eine UND-Verknüpfung des Bitergebnisses mit dem Zustand des als Parameter angegebenen Timers durch.
ODER_TI	Konstante		Führt eine ODER-Verknüpfung des Bitergebnisses mit dem Zustand des als Parameter angegebenen Timers durch.
NODR_TI	Konstante		Führt eine ODER-Verknüpfung des Bitergebnisses mit dem invertieren Zustand des als Parameter angegebenen Timers durch.
DEC_TI	Konstante		Verringert den Wert des als Parameter angegebenen Timers um 1. Falls der Timer nach dieser Operation auf 0 steht, d.h. abgelaufen ist, wird das Bitergebnis eingeschaltet.

n Tabelle 40 – Timer-Makros

## n Virtuelle Timer

Sollten Ihnen bei der Programmierung die vorhandenen 32 Systemtimer nicht ausreichen, dann können Sie auch mit Software-Timern, den sogenannten virtuellen Timern, arbeiten. Virtuelle Timer stellen Ihnen beliebig viele zusätzliche Timer zur Verfügung, die innerhalb des SPS-Programms über Variablen und Merker realisiert werden. Hierzu wird nur einer der fest integrierten MC200 Systemtimer benötigt.

### Einbinden der virtuellen Timer

Um die virtuellen Timer in Ihrem SPS-Programm zu verwenden, können Sie entweder die Datei VTIMER.MC, die im Verzeichnis "Gemeinsame Dateien" Ihrer VMC Workbench Installation enthalten ist, in Ihr Projekt einbinden, oder aber einen entsprechenden Quelltext in eine Ihrer Quellcode-Dateien mit einbauen.

Ändern Sie dann am Anfang der VTIMER.MC-Datei den zu verwendenden Timer. Standardmäßig läuft das VTIMER-Modul mit dem Systemtimer 6. Sie können jedoch jeden beliebigen Timer hier verwenden. Achten Sie bitte darauf, daß die Timer 17-32 mit einer Auflösung von 1/100s arbeiten.

Definieren Sie anschließend in Ihrer Definitionsdatei folgende symbolischen Variablen:

- n V\_TIMER\_VPTR: Wird intern von VTIMER.MC verwendet.
- n V\_TIMER\_MPTR: Wird intern von VTIMER.MC verwendet
- n V\_TIMER\_COUNT: Wird intern von VTIMER.MC verwendet

Anschließend definieren Sie bitte noch folgende Konstanten:

- n K\_TIMER\_VBLOCK: Definieren Sie diesen Wert mit der Nummer der ersten Variable, die für die virtuellen Timer verwendet werden soll.
- n K\_TIMER\_MBLOCK: Definieren Sie diesen Wert mit der Nummer des ersten Merkers, der für die virtuellen Timer verwendet werden soll.
- n K\_TIMER\_COUNT: Definieren Sie diesen Wert mit der Anzahl der virtuellen Timer, die Sie verwenden möchten.

Jetzt sind Ihre virtuellen Timer schon fast einsatzfähig. Als letzten Schritt müssen Sie nun im Hauptzyklus Ihres SPS-Programms noch einen regelmäßigen Aufruf der VTIMER-Routine einbauen, und zwar mit:

```
{ GEHUPRI    VTIMER                // Virtuelle Timer aktualisieren
```

Spätestens jetzt verfügen Sie über vollständig in der SPS-Sprache geschriebene, virtuelle Timer.

### Arbeiten mit virtuellen Timern

Die virtuellen Timer verhalten sich prinzipiell genauso wie die integrierten Systemtimer: Sie schreiben den Startwert in eine Variable, setzen den zugeordneten Merker und warten darauf, daß das VTIMER-Modul diesen Merker wieder zurücksetzt.

Etwas komplizierter wird das Ganze für Sie nur dadurch, daß Sie ja selbst bestimmen, welche Variablen und welche Merker für die virtuellen Timer verwendet werden sollen, und es deshalb keine vom System bereits definierten symbolischen Namen für diese Merker und Variablen gibt.

Es wird deshalb empfohlen, daß Sie in Ihrer Definitionsdatei noch zusätzlich symbolische Namen für die entsprechenden Merker und Variablen definieren, wie auch in folgendem Beispiel:



```

DEF_V    200, V_TIMER_VPTR      // Interne Variablen für VTIMER definieren
DEF_V    201, V_TIMER_MPTR
DEF_V    202, V_TIMER_COUNT

DEF_K    220, K_TIMER_VBLOCK    // VTIMER soll Variablen ab 220 verwenden
DEF_K    2000, K_TIMER_MBLOCK   // VTIMER soll Merker ab 2000 verwenden
DEF_K    48, K_TIMER_COUNT      // Wir wollen 48 virtuelle Timer verwenden

// Damit steht nun fest: unserem ersten virtuellen Timer ist die Variable 220
// und der Merker 2000 zugeordnet, dem zweiten virtuellen Timer die Variable
// 221 und der Merker 2001, usw. Wir wollen unseren wichtigsten virtuellen
// Timern aber noch zusätzliche, symbolische Namen geben

DEF_V    220, V_VTIM_1         // Variable für ersten virtuellen Timer
DEF_M    2000, M_VTIM_1        // Merker für ersten virtuellen Timer
DEF_V    221, V_VTIM_2         // Variable für zweiten virtuellen Timer
DEF_M    2001, M_VTIM_2        // Merker für zweiten virtuellen Timer
...
// usw.

// Die Programmierung der virtuellen Timer entspricht genau dem Umgang mit
// den integrierten Timern, außer, daß Sie nicht auf die vordefinierten
// Timer-Makros zugreifen können. Wir zeigen dies hier nochmals an dem
// Beispiel der Fehlerüberwachung für einen Zylinder:

EIN_A    A_ZYL                 // Zylinder in Endlage fahren lassen

AUS_M    M_VTIM_1              // Timer 1 stoppen
LAD_VA   V_VTIM_1, 100        // Timer 1 wird mit dem Wert 100 geladen
EIN_M    M_VTIM_1              // Timer 1 wird gestartet

Loop:
// Wir prüfen jetzt, ob der Zylinder die
// Endlage erreicht hat, oder der Timer
// abgelaufen ist.
GEHUPRI  VTIMER                // Virtuelle Timer aktualisieren
LAD_E    E_ZYL_END             // Hat Zylinder Endlage erreicht?
SPRINGJ  AllesOk              // Ja, weiter
// Zylinder hat die Endlage noch nicht erreicht.
NLAD_M   M_VTIM_1              // Ist der Timer noch nicht abgelaufen?
SPRINGN  Loop                  // Nein, dann warten wir noch
SPRING   Stoerung              // Timer abgelaufen, Stoerung

```

### Ungenauigkeit der virtuellen Timer

Weil die virtuellen Timer natürlich vom Ablauf Ihres SPS-Programms abhängig sind, kann es natürlich zu Ungenauigkeiten kommen. Erfahrungsgemäß liegt diese bei einem größeren SPS-Programm im Bereich von etwa 5%. Verwenden Sie deshalb virtuelle Timer nicht für Funktionen, die sehr zeitkritisch ausgeführt werden müssen. Der typische Anwendungsfall für virtuelle Timer ist z.B. die bereits mehrfach angesprochene Fehlerüberwachung, da es hier auf die Genauigkeit der integrierten Timer nicht in diesem Maße ankommt.

## n Globaler Systemtimer

Zusätzlich zu den 32 integrierten Timern verfügt das MC200 System über einen globalen Systemtimer. In diesem Timer wird die Anzahl der 1/100s seit dem letzten Neustart der Steuerung festgehalten. Mit etwas Programmiergeschick können Sie auch diesen Timer, dessen Wert in der Variable V\_SYSTIMER enthalten ist, für zeitgesteuerte Aufgaben verwenden.

Bitte ändern Sie niemals den Wert dieser Variable aus Ihrem SPS-Programm heraus, weil auch viele andere Funktionen des MC200 Systems über den globalen Systemtimer gesteuert werden.

## n Blinkmerker

Für spezielle zeitgesteuerte Aufgaben können Sie auch die im MC200 System integrierten Blinkmerker verwenden: wenn Sie z.B. eine Warnleuchte an Ihrer Maschine blinken lassen möchten, dann koppeln Sie den entsprechenden Ausgang einfach mit einem der Blinkmerker des MC200 Systems. So läßt sich dann ein Blinklicht realisieren, ohne daß Sie hierfür einen wertvollen Systemtimer "opfern" müssen – und natürlich zudem noch mit wesentlich weniger Programmieraufwand.

### Verfügbare Blinkmerker

Die Blinkmerker im MC200 System gibt es in Ausführungen mit verschiedenen Blink-Zyklen. Der langsamste Blinkmerker wird alle 1,28s umgeschaltet, der schnellste schaltet jede 1/100s um. Eine Übersicht der verfügbaren Blinkmerker finden Sie im Kapitel 6.10 - Blinkmerker (Seite 222).

### Beispiel

```
// In diesem Beispiel lassen wir einen Ausgang alle 0,64s blinken.  
LAD_M      M_BLINK64           // Blinkmerker 0,64s  
MOD_A      A_BLINKLICHT       // Lampe umschalten
```

## 4.5 Serielles Modul

Mit dem seriellen Erweiterungsmodul der MC200 Familie erhalten Sie eine Vielzahl neuer Möglichkeiten zum Anschluß der Steuerung an fremde Peripherie-Gerät, wie z.B.

- n Protokoll-Drucker
- n Barcode-Leser
- n Fremdsteuerungen

Die Programmierung der seriellen Schnittstelle erfolgt vollständig aus der SPS heraus. Deshalb kann jedes beliebige Binärprotokoll von Ihnen implementiert werden.

### n Sende- und Empfangspuffer

Zur Kommunikation mit seriellen Geräten wurden ausreichende Pufferspeicher sowohl in der MC200CPU als auch im seriellen Modul selbst realisiert. Die Puffergrößen sehen wie folgt aus:

Art des Puffers	Enthalten im Modul..	Puffergröße
Empfangspuffer (FIFO)	MC200CPU / MC200PROFI	64 Byte
Empfangspuffer (FIFO)	MC200SER	64 Byte
Sendepuffer (FIFO)	MC200CPU / MC200PROFI	64 Byte
Sendepuffer (FIFO)	MC200SER	16 Byte

n Tabelle 41 – Puffergrößen serielles Modul

Durch die verteilten Puffer ist das asynchrone Verhalten des seriellen Erweiterungsmoduls in bezug auf die SPS-CPU gewährleistet, d.h. das SPS-Programm wird vollständig unabhängig vom seriellen Modul abgearbeitet. In einem regelmäßigen Zyklus kommuniziert die SPS-CPU mit dem seriellen Erweiterungsmodul und tauscht Sende- und Empfangsdaten aus, oder, in anderen Worten, gleicht die Sende- und Empfangspuffer ab.

Um die Puffer im seriellen Modul brauchen Sie sich nicht zu kümmern; das erledigt die CPU automatisch. Die internen Puffer der CPU jedoch sollten Sie im Auge behalten, schließlich werden hier die Daten von Ihnen gelesen und geschrieben.

### Puffergrößen

Aus der verteilten Datenspeicherung ergibt sich somit eine Pufferkapazität von 128 Byte für Empfangsdaten und 80 Byte für Sendedaten. Diese ungleiche Aufteilung ist bewusst gewählt: falls der Sendepuffer nicht ausreicht, kann die Übertragung jederzeit durch das SPS-Programm wiederholt werden. Ein angeschlossenes Peripheriegerät, wie z.B. ein Barcode-Leser, verfügt jedoch nicht über eine entsprechende Wiederholungs-Logik und sendet einfach Daten. Um zu vermeiden, daß hierbei Daten verloren gehen, muß der Empfangspuffer ausreichend groß dimensioniert sein.

### Ich sende Daten: was passiert dann?

Wenn Sie aus dem SPS-Programm heraus Daten senden wird folgender Ablauf ausgelöst:

- n Die CPU prüft, ob genügend Speicherplatz für die zu sendenden Daten im Sendepuffer vorhanden ist. Falls nicht, wird ein Fehlermerker (M\_SEROUT\_OV\_x) gesetzt und der Vorgang wird abgebrochen. Hierbei werden keine Daten in den Puffer kopiert! Der Fehlermerker muß aus dem SPS-Programm wieder zurückgesetzt werden, bevor neue serielle Daten gesendet werden können.
- n Die Sendedaten werden in den seriellen Sendepuffer der CPU kopiert. Die Ausführung des SPS-Befehls ist hiermit beendet, das SPS-Programm wird normal fortgesetzt.
- n Asynchron zum Programmablauf prüft die CPU regelmäßig, wie viel Speicher im Sendepuffer des seriellen Erweiterungsmoduls zur Verfügung steht. Falls freie Speicherplätze vorhanden sind, werden Sendedaten zum seriellen Modul geschickt. Dies wird solange wiederholt, bis der Sendepuffer im CPU-Modul leer ist.

**Ich empfangen Daten: was passiert dann?**

Der Empfang serieller Daten verläuft ähnlich asynchron wie auch das Senden:

- n Das serielle Modul empfängt Daten und trägt diese im eigenen Empfangspuffer ein.
- n Die CPU prüft regelmäßig, ob neue Daten im Empfangspuffer des seriellen Moduls vorhanden sind. Falls ja, werden diese Daten aus dem Empfangspuffer des seriellen Moduls in den Empfangspuffer des CPU-Moduls übertragen. Der Empfangspuffer des seriellen Moduls wird gelöscht und steht für neue Empfangsdaten zur Verfügung. In der CPU wird die Systemvariable V\_SERIN\_x auf die Anzahl Byte im Empfangspuffer der CPU gesetzt, der Merker M\_SERIN\_x wird eingeschaltet. Falls der Puffer überläuft, wird zusätzlich der Merker M\_SERIN\_OV\_x eingeschaltet.
- n Sobald das SPS-Programm serielle Daten abholt, wird der Empfangspuffer in der CPU geleert, die Systemvariable V\_SERIN\_x auf 0 gesetzt und der Merker M\_SERIN\_x gelöscht. Falls ein Überlauf stattgefunden hat, muß der Merker M\_SERIN\_OV\_x durch das SPS-Programm zurückgesetzt werden.

**n Wenn es eng wird: Pufferüberläufe**

Sollte der großzügig dimensionierte Puffer für serielle Sende- und Empfangsdaten trotzdem einmal voll sein, wird dies durch einen der Systemfehlermerker M\_SERIN\_OV\_x bzw. M\_SEROUT\_OV\_x signalisiert.

**M\_SERIN\_OV**

M\_SERIN\_OV\_x wird gesetzt, wenn neue serielle Empfangsdaten bereitstehen, aber kein freier Speicher mehr im Empfangspuffer des CPU-Moduls zur Verfügung steht. In diesem Fall ist davon auszugehen, daß Daten verloren gegangen sind.

Sobald dieser Fehler aufgetreten ist, sollten Sie am besten den Empfangspuffer vollständig leeren und das serielle Modul zurücksetzen. Da vermutlich sowieso Daten verloren gegangen sind, ist eine Analyse der jetzt noch im Puffer befindlichen Daten meist nicht sonderlich sinnvoll. Verwenden Sie den Befehl CLRSER (Seite 53), um alle Puffer und Fehlermerker zu löschen sowie das serielle Modul zurückzusetzen.

```
LAD_M      M_SERIN_1           // Neue serielle Empfangsdaten verfügbar?
UPRENDN                                         // Nein, Unterprogramm Ende
LAD_M      M_SERIN_OV_1       // Empfangspuffer voll?
CLRSER     1                   // Dann Puffer löschen
SPRINGJ    Fehl erbehandl ung // Fehl ermel dung ausgeben
RCVSR     1, 200              // Ansonsten Daten in Variable ab 200 laden
```

**M\_SEROUT\_OV**

Weniger kritisch ist ein Überlauf bei der Ausgabe von Daten: hier wissen Sie in der Regel ganz genau, welche Daten nicht übermittelt wurden: Ihre letzte Sendeaufforderung.

Löschen Sie den Fehlermerker M\_SERIN\_OV\_x und wiederholen Sie das serielle Senden solange, bis nach der Sendeaufforderung kein Fehlermerker mehr gesetzt bleibt:

```
Loop:
SND SER     1, 500             // Daten ab Variable 500 senden
LAD_M      M_SEROUT_OV_1     // Pufferüberlauf?
AUS_M      M_SEROUT_OV_1     // Fehlermerker wieder löschen
SPRINGJ    Loop              // Ausgabe wiederholen
```

**n Beispiel: Barcode-Leser**

Wir empfangen und analysieren Daten von einem Barcode-Leser. Wird ein Barcode erkannt, gibt das Unterprogramm in VARERG die Nummer des gültigen Barcodes zurück, ansonsten eine 0. Im Rahmen der Analyse kopieren wir als SPS-Texte gespeicherte Barcodes in ein Variablenfeld und vergleichen den Inhalt dieses Variablenfelds mit den empfangenen Daten. Stimmen – in diesem Beispiel - mindestens die ersten acht des Barcodes überein, wurde ein korrekte Code erkannt.

**Datei BARCODE.DEF**

In dieser Datei sind einige grundsätzliche Definitionen für die Barcode-Analyse enthalten:

```
DEF_W      1, K_FIRST_TXT           // Erste Text-Nummer mit Barcodes
DEF_W      5, K_LAST_TXT           // Letzte Text-Nummer mit Barcodes
DEF_W      8, K_BARCODE_LEN        // Zu analysierende Länge der Barcodes
DEF_W      10, K_BARCODE_RCV       // Gesamtlänge Empfangsdaten pro Barcode (CR/LF!)
DEF_V      200, V_Z_SER            // Zeigervariable für serielle Empfangsdaten
DEF_V      201, V_Z_TXT           // Zeiger für gespeicherte Textdaten
DEF_V      1200, V_FELD_SER        // Anfang Variablenfeld für den Datenempfang
DEF_V      1300, V_FELD_TXT       // Anfang Variablenfeld für Barcode-Vergleich
```

**Datei BARCODE.MCT**

In dieser Datei sind die gültigen Barcodes als SPS-Texte definiert.

```
TEXT                               // Anfang der Texte
CODE0001 "44659201"               // Erster gültiger Barcode
CODE0002 "64332673"               // Barcode
CODE0003 "54408717"               // Barcode
CODE0004 "01540789"               // Barcode
CODE0005 "77850447"               // Letzter gültiger Barcode
```

**Datei BARCODE.MC**

Diese Datei enthält das eigentliche Unterprogramm zur Barcode-Analyse.

```
BARCODE:                           // Unterprogramm Barcode-Analyse
LAD_M      M_SERIN_1                // Neue Empfangsdaten verfügbar?
UPRENDN                               // Nein, zurück
VERG_VA    V_SERIN_1, K_BARCODE_RCV // Bereits kompletten Barcode empfangen?
NLAD_M     M_KLEINER                // Vergleichsergebnis abfragen
UPRENDN                               // Nein, zurück
RCVSER     1, V_FELD_SER            // Empfangsdaten in Variablenfeld kopieren
LAD_VA     V_SERNR, K_FIRST_TXT     // Nummer des ersten Textes in Format laden

BARCODE01:                          // Schleife zum Umkopieren von Texten
LAD_VA     V_SERMSK, V_FELD_TXT     // Ziel des Textes aus V_SERNR ist V_BARCODE_COMP
SETSER     1, SEND_TO_VAR           // Text aus V_SERNR in Variablenfeld kopieren
LAD_VA     V_Z_SER, V_FELD_SER      // Zeiger auf Empfangsdaten initialisieren
LAD_VA     V_Z_TXT, V_FELD_TXT      // Zeiger auf gespeicherte Texte initialisieren
LAD_VA     V_ZAEHLER, K_BARCODE_LEN // Zu analysierende Länge des Barcodes

BARCODE02:                          // Schleife zum Vergleich zweier Texte
VERG_II    V_Z_SER, V_Z_TXT         // Vergleiche Zeigerinhalte (indirekt-indirekt)
LAD_M      M_GLEICH                 // Inhalte identisch?
SPRINGN    BARCODE03               // Nein, nächsten Text analysieren
INC_V      V_Z_SER, 1               // Zeiger auf nächstes Zeichen
INC_V      V_Z_TXT, 1               // Zeiger auf nächstes Zeichen
DEC_V      V_ZAEHLER, 1             // Zähler bereits verglichene Zeichen
LAD_M      M_GLEICH                 // Wenn Zähler=0, dann haben wir Barcode gefunden
SUB_VA     V_SERNR, K_FIRST_TXT     // V_SERNR - Erster Text = Textoffset von 0-4
INC_V      VARERG, 1                // +1, damit der erste Text 1 ist
UPRENDJ                               // Barcode gefunden, Unterprogramm Ende

BARCODE03:                          // Auf nächsten Text zeigen
INC_V      V_SERNR, 1               // Auf nächsten Text zeigen
VERG_VA    V_SERNR, K_LAST_TXT      // Bereits der letzte gültige Text?
LAD_M      M_GROESSER               // Vergleich abfragen
LAD_VA     VARERG, 0                // Ja, Barcode nicht gefunden
UPRENDJ                               // Unterprogramm Ende
SPRING     BARCODE01               // Weitere Texte vergleichen
```

## 4.6 EAU-T Emulation

Im Rahmen des Vorgängersystems zur MC200 Familie, der MC100, war ein Spezialmodul mit der Bezeichnung MC100EAU-T erhältlich. Dieses Modul erlaubte logische Verknüpfung zwischen digitalen Eingängen und digitalen Ausgängen in Abhängigkeit einer parametrierbaren Verzögerungszeit.

Eine ähnliche Funktionalität ist in den CPU-Modulen der MC200 Familie integriert. Hauptzweck dieser Sonderfunktion ist eine extrem schnelle Reaktion auf einen Eingangsflankenwechsel: die EAT-T Emulation erkennt den Eingangswechsel und schaltet den Ausgang binnen 0,2ms nach Entprellung des digitalen Eingangs. Eine ähnliche Reaktionszeit ist aus einem großen SPS-Programm heraus nur schwer zu realisieren.

### n Funktionsweise

Die EAU-T Emulation wird mit einem Eingang, einem Ausgang und einer Verzögerungszeit parametrierbar. Diese Informationen werden in Systemvariablen gespeichert. Durch Einschalten eines Systemmerkers wird die Funktion aktiviert. Folgender interner Ablauf wird hierbei ausgelöst:

- n Das CPU-Modul speichert den aktuellen Zustand des parametrierbaren Eingangs (ein oder aus).
- n Im CPU-Modul wird eine schnelle Überwachungsfunktion aktiviert, die permanent einen Flankenwechsel des parametrierbaren Eingang kontrolliert.
- n Sobald der Eingangszustand umgeschaltet wird, startet das CPU-Modul die parametrierte Verzögerungszeit.
- n Nach Ablauf der Verzögerungszeit ermittelt das CPU-Modul den aktuellen Status des parametrierbaren digitalen Ausgangs und schaltet den Ausgang um.
- n Der erfolgreiche Abschluß der Funktion wird durch das Zurücksetzen des Systemmerkers signalisiert. Die EAU-T Emulation kann auch während des Ablaufs jederzeit durch Zurücksetzen des Systemmerkers deaktiviert, sprich: angehalten werden.

### n Parametrierung

Für die Parametrierung der EAU-T Emulation stehen zwei Systemvariablen und ein Systemmerker zur Verfügung:

Variable	Nummer	Bedeutung
V_EAUT_TIM	Variable 47	Enthält die Verzögerungszeit zwischen dem Flankenwechsel des Eingangs und dem auszulösenden Flankenwechsel des Ausgangs in Millisekunden.
V_EAUT_MASK	Variable 48	Enthält in den unteren 16 Bit die Eingangsnummer, in den oberen 16 Bit die Ausgangsnummer.
M_EAUT	Merker 2273	Aktiviert die Funktion.

n Tabelle 42 – Puffergrößen serielles Modul

### n Beispiel

In diesem Beispiel wird der Ausgang 3 in Abhängigkeit vom Eingang 10 nach 4ms umgeschaltet.

```
LAD_VA    V_EAUT_TIM, 4           // 4ms Verzögerungszeit
LAD_VA    VARERG, 3             // Ausgang 3
SLL_V     VARERG, 16           // in die oberen 16 Bit schieben
ODER_VA   VARERG, 10          // Eingangsnummer 10 dazumaskieren
LAD_VV    V_EAUT_MASK, VARERG  // Ein-/Ausgangsmaske in Systemvariable speichern
EIN_M     M_EAUT               // Funktion auslösen
```

## n Raum für Ihre Notizen

**n Raum für Ihre Notizen**



# Kapitel 5 System- und Achsparameter

Die Parameter sind eine besondere Gruppe Daten innerhalb des MC200 Systems: grundsätzlich zwar ähnlich aufgebaut wie die SPS-Variablen steuern sie das Verhalten einzelner Komponenten des Systems bis in das kleinste Detail. Mit Parametern konfigurieren Sie also die Komponenten eines MC200 Systems – gleichgültig, ob es sich nun um Achsen, das CPU-Modul oder serielle Erweiterungsmodule handelt.

Normalerweise werden System- und Achsparameter nur einmal eingestellt, nämlich bei Inbetriebnahme der Anlage. Für diese Zwecke stehen unterschiedliche PC-Programme aus dem VMC Workbench Paket zur Verfügung, wie z.B. das vollständig integrierte VMC SAT, das Achsentestprogramm MCMove oder das altbewährte VMC Setup200.

## Parameter ändern aus der SPS heraus

Manchmal kann es jedoch notwendig sein, einen Parameter aus dem SPS-Programm heraus zu beeinflussen, z.B. dann, wenn während eines Ablaufs immer wieder unterschiedliche Achsen miteinander interpolieren sollen oder Sie einzelne Konfigurationspunkte durch den Endanwender veränderbar in Ihr SPS-Programm integrieren. Das könnte dann in etwa so aussehen:

```
// Als Beispiel: Normalerweise fahren die Achsen 1, 2 und 3 interpoliert.
// Jetzt soll vom SPS-Programm aus eingestellt werden, daß die Achsen
// 1, 2 und 4 miteinander interpolieren. Über eine Bitmanipulation soll das
// Interpolationssteuerwort (Parameter 16 = bitcodiert) geändert werden.
// Für die Interpolation der Achsen 1, 2 und 3 sind die Bits 1, 2 und 3
// gesetzt, damit die Achsen 1, 2 und 4 miteinander interpolieren müssen die Bits
// 1, 2 und 4 gesetzt werden.

LAD_VA    V_Maske1, 4           // Bitmaske zum Löschen der Achse 3
                                           // (binär 00000100, hex 04h)
LAD_VA    V_Maske2, 12        // Bitmaske zum Setzen der Achse 4
                                           // (binär 00001000, hex 08h)

GETPAR    116, VARERG          // Auslesen des Interpolationssteuerwortes
UND_VV    VARERG, V_Maske1    // Achse 3 löschen
ODER_VV   VARERG, V_Maske2    // Achse 4 setzen
SETPAR    116, VARERG          // Neues Interpolationssteuerwort setzen

GETPAR    216, VARERG          // Auslesen des Interpolationssteuerwortes
UND_VV    VARERG, V_Maske1    // Achse 3 löschen
ODER_VV   VARERG, V_Maske2    // Achse 4 setzen
SETPAR    216, VARERG          // Neues Interpolationssteuerwort setzen

LAD_VA    VARERG, 0           // Achse 3 soll nicht mehr interpolieren
SETPAR    316, VARERG          // Neues Interpolationssteuerwort setzen

LAD_VA    VARERG, 3           // Achse 4 soll jetzt mit Achse 1 + 2
                                           // interpolieren (binär 00000011, hex 03h)
SETPAR    416, VARERG          // Neues Interpolationssteuerwort setzen
```

## 5.1 Systemparameter

Einige Parameter innerhalb der MC200 steuern das Verhalten des Systems an sich bzw. dienen der Identifikation von SPS-Projekten. Diese Parameter werden im Folgenden aufgeführt.

### n Ändern der Parameter

Alle Systemparameter theoretisch beliebig verändert werden. Es macht jedoch keinerlei Sinn, z.B. die Projekt-ID des SPS-Programms manuell zu verändern. Sinnvoll ist hier lediglich eine Veränderung des Parameters 3 (Profibusadresse) und des Parameters 8 (Baudrate der internen RS232-Schnittstelle).

### n Übersicht Systemparameter

Nummer	Bezeichnung	Bedeutung
3	Profibusadresse (nur MC200PROFI)	Aktuelle Profibus-Adresse des angeschlossenen MC200PROFI-Moduls im Profibus als numerischer Wert. Achtung: Der neue Wert wird erst nach einem Steuerungsreset und Neuaufsetzen auf dem Profibusstrang übernommen.
5	Projekt-ID, obere 32 Bit	Bei der Arbeit mit der VMC Workbench werden für SPS-Projekte eindeutige, 64 Bit lange IDs erzeugt. In diesem Parameter sind die oberen 32 Bit dieser ID enthalten.
6	Projekt ID, untere 32 Bit	Bei der Arbeit mit der VMC Workbench werden für SPS-Projekte eindeutige, 64 Bit lange IDs erzeugt. In diesem Parameter sind die unteren 32 Bit dieser ID enthalten.
7	Projekt-Änderungsstand	Bei der Arbeit mit der VMC Workbench wird jede Änderung am Projekt durch das Erhöhen eines internen Änderungszählers gekennzeichnet. In diesem Parameter ist der Änderungszähler enthalten.
8	Aktuelle Baudrate	Dieser Parameter enthält die aktuelle Baudrate für die Kommunikation der Steuerung mit dem PC. 0 kennzeichnet die niedrigste Baudrate (9600 Baud), 4 die gegenwärtig höchste (115200 Baud).

n Tabelle 43 - Systemparameter

#### Siehe auch

SETPAR (Seite 115)

GETPAR (Seite 70)

Interne Systemdaten (Seite 197)

## 5.2 Achsparameter

Das MC200 System ermöglicht den Betrieb von bis zu 16 Achsen an einer Steuerung. Hierbei ist auch der vollständig gemischte Betrieb von Servo- und Schrittmotorachsen möglich. Dem trägt die MC-1B Sprache Rechnung: alle Funktionen und alle Statusinformationen sind für Servo- und Schrittmotorachsen identisch. Innerhalb der Programmierung mit MC-1B macht es also keinerlei Unterschied, ob physikalisch hier eine Servo- oder eine Schrittmotorachse angeschlossen ist.

Auch bei den Achsparametern sind nahezu alle Parameter identisch. Lediglich dort, wo es in die "Tiefen des Systems" geht, wenn also z.B. ein PID-Regler programmiert werden soll, unterscheiden sich die Parameter für die Achsen. Um diese Unterschiede zu berücksichtigen, finden Sie auf den folgenden Seiten zunächst eine Übersicht der Achsparameter für Servomotoren (MC200MOC), danach eine Übersicht der Achsparameter für Schrittmotorcontroller (MC200SM2).

### n Ändern der Parameter

Alle Parameter auf den nächsten Seiten können (fast) beliebig verändert werden. Einige Achsparameter, die erweiterte Statusinformationen zurückliefern und nur gelesen werden dürfen, sind in den folgenden Übersichten nicht enthalten. Bitte vergleichen Sie hierzu die Übersichten in Kapitel 6.7 - Achsenstatus (Seite 202).

#### Bitcodierte Parameter

Einige Achsparameter sind in bitcodierter Form gespeichert. Dies bedeutet: nicht die eigentlich gespeicherte Zahl ist entscheidend, sondern vielmehr die einzelnen Bits innerhalb des 32-Bit Wertes. Diese Parameter sind innerhalb der Übersicht entsprechend gekennzeichnet. Bei Veränderung der Parameter aus einem SPS-Programm heraus sollten Sie jedoch beachten, daß einige Einstellungen sich gegenseitig ausschließen. Um z.B. das Messsystem auf eine Auflösung von 1,000mm zu setzen, wird das Bit 8 im Parameter 30 gesetzt, für eine Auflösung von 0,100mm das Bit 9 im gleichen Parameter. Diese Bits dürfen niemals gleichzeitig gesetzt werden! Dies würde bedeuten, dass das Messsystem sowohl mit 1,000mm als auch mit 0,100mm Auflösung gleichzeitig arbeiten würde und zu nicht voraussehbaren Ergebnissen führen.

### n Berechnung der Parameter-Nummer

Die Übersichten auf den folgenden Seiten führen zu jedem Parameter die Basisparameter-Nummer auf. Sie dürfen jedoch niemals direkt mit der Basisparameter-Nummer arbeiten, sondern müssen sich zuvor die Parameter-Nummer für die gewünschte Achse errechnen:

$$\text{Parameter-Nummer} = \text{Achse} \times 100 + \text{Basisparameter-Nummer}$$

Soll also z.B. der Parameter 13 (Nullpunkt-Offset) für die Achse 2 geändert werden, so würde sich die Parameter-Nummer wie folgt berechnen:

$$\begin{aligned} \text{Parameter-Nummer} &= \text{Achse } 2 \times 100 + \text{Basisparameter } 13 \\ \text{Parameter-Nummer} &= 2 \times 100 + 13 \\ \text{Parameter-Nummer} &= 213 \end{aligned}$$

Bitte denken Sie stets daran, daß Sie beim Lesen oder Schreiben eines Parameters den Wert für die Achse dazuzählen. Das Schreiben von Parameterwerten unter 100 kann zu unerwarteten Ergebnissen führen, weil in diesen Parametern systeminterne Daten gespeichert wurden.

#### Siehe auch

SETPAR (Seite 115)

GETPAR (Seite 70)

SETFUN (Seite 112)

Achsenstatus (Seite 202)

## n Übersicht Achspanparameter für Servomotorcontroller

Nummer	Bezeichnung	Bedeutung
10	Schleppfehler	Gibt an, wie groß der Schleppfehler für diese Achse werden darf, bevor eine Achsstörung ausgelöst wird.
11	Sollgeschwindigkeit	Gibt die gewünschte Sollgeschwindigkeit an. Die Einstellung wird nur bei Verwendung des Profils 0, d.h. ohne Beschleunigungsvorsteuerung, verwendet.
12	Sollbeschleunigung	Gibt den gewünschten Beschleunigungswert (die Rampe) an.
13	Nullpunkt-Offset	Gibt an, um wieviel Maßeinheiten der logische Nullpunkt von der Position des physikalischen Nullpunkts (Referenzschalter) entfernt liegt. Wird z.B. verwendet, um eine Werkzeugkorrektur durchzuführen.
14	Software-Endschalter Plus	Dieser Parameter enthält die Position des Software-Endschalters Plus im jeweiligen Umrechnungssystem. Ob dieser Software-Endschalter verwendet wird, entscheidet das Bit 4 im Parameter 30.
15	Software-Endschalter Minus	Dieser Parameter enthält die Position des Software-Endschalters Minus im jeweiligen Umrechnungssystem. Ob dieser Software-Endschalter verwendet wird, entscheidet das Bit 4 im Parameter 30.
16	Interpolationssteuerwort (bitcodiert)	Dieser Parameter entscheidet, welche Achsen miteinander interpolieren. Die Informationen sind hierbei bitcodiert, d.h. für die Interpolation mit Achse 1 muß das Bit 0 gesetzt werden, für die Interpolation mit Achse 2 das Bit 1 usw..
Bit 0	Interpolation mit Achse 1	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 1.
Bit 1	Interpolation mit Achse 2	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 2.
Bit 2	Interpolation mit Achse 3	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 3.
Bit 3	Interpolation mit Achse 4	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 4.
Bit 4	Interpolation mit Achse 5	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 5.
Bit 5	Interpolation mit Achse 6	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 6.
Bit 6	Interpolation mit Achse 7	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 7.
Bit 7	Interpolation mit Achse 8	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 8.
Bit 8	Interpolation mit Achse 9	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 9.
Bit 9	Interpolation mit Achse 10	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 10.
Bit 10	Interpolation mit Achse 11	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 11.
Bit 11	Interpolation mit Achse	Wenn gesetzt, interpoliert diese Achse beim Starten über einen

Nummer	Bezeichnung	Bedeutung
	12	Interpolationsbefehl mit der Achse 12.
Bit 12	Interpolation mit Achse 13	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 13.
Bit 13	Interpolation mit Achse 14	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 14.
Bit 14	Interpolation mit Achse 15	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 15.
Bit 15	Interpolation mit Achse 16	Wenn gesetzt, interpoliert diese Achse beim Starten über einen Interpolationsbefehl mit der Achse 16.
20	Parametrierung Lageregler (bitcodiert)	Dieser Parameter enthält die Konfiguration des Lagereglers. Die entsprechenden Werte sind bitcodiert, d.h. jede einzelne Information ist in einem Bit abgespeichert. Es macht deshalb keinen großen Sinn, die Parametrierung des Lagereglers vom SPS-Programm aus zu verändern.
Bit 0	Lageregelsinn invertiert	Wenn gesetzt, wird der Lageregelsinn logisch invertiert ausgewertet.
Bit 1	Begrenzung Sollwert im Stillstand auf +/- 1,0V	Wenn gesetzt, wird der maximal ausgegebene Sollwert auf eine Spannung von +/- 1,0V begrenzt.
Bit 2	Simulation	Wenn gesetzt, arbeitet das System im Simulationsmodus. Das bedeutet, daß Encoder-Signale nicht ausgewertet werden, sondern durch den Achscontroller simuliert werden.
Bit 3	Auswertung In-Position Fenster (Stillstandsfenster)	Wenn gesetzt, wird das im Parameter 25 angegebene In-Positionsfenster ausgewertet, d.h. der Achscontroller meldet bereits dann "In Position", wenn die Distanz zur Zielposition nicht mehr größer als die im Parameter 25 angegebene Strecke ist.
Bit 4	Schleppfehler = Notaus	Wenn gesetzt, wird beim Erreichen eines Schleppfehlerwertes, der den im Parameter 10 angegebenen, maximalen Schleppfehler übersteigt, ein Notaus ausgelöst.
Bit 5	Stromreglermodus	Wenn gesetzt, arbeitet der Achscontroller im Stromregler-Modus.
21	P-Anteil	Gibt den P-Anteil des Lagereglers an.
22	I-Anteil	Gibt den I-Anteil des Lagereglers an.
23	D-Anteil	Gibt den D-Anteil des Lagereglers an.
24	I-Begrenzung	Gibt die I-Begrenzung des Lagereglers an.
25	In-Position Fenster (Stillstandsfenster)	Definiert das In-Positionsfenster in Maßeinheiten des aktuellen Zählsystems. Dieses Fenster wird nur dann ausgewertet, wenn das Bit 3 des Parameters 20 gesetzt ist.
30	Parametrierung Zählsystem (bitcodiert)	Mit diesem Parameter definieren Sie das von Ihnen verwendete Zähl- und Maßsystem. Alle Werte innerhalb dieses Parameters sind bitcodiert, d.h. jede Information belegt ein oder mehrere Bits. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Umrechnung für Zahnriemen, Zahnstangen usw. (PI)	Wenn gesetzt, werden alle Positionswerte automatisch in Positionen auf einer Kreisbahn umgerechnet. Sie benötigen diese Umrechnung, wenn Sie z.B. einen Zahnriemen- oder Zahnstangenantrieb parametrieren möchten.

Nummer	Bezeichnung	Bedeutung
Bit 1	Umrechnung erfolgt in Inch	Wenn gesetzt, werden alle Positionswerte als Angaben in Inch betrachtet. Wenn nicht gesetzt, werden alle Positionswerte als Angaben in mm betrachtet.
Bit 2	Endlosbetrieb mit Reduzierung und Verfahrwegoptimierung	Wenn gesetzt, wird im 360°-Betrieb automatisch die kürzeste Fahrstrecke ausgesucht. Dies kann mit einer automatischen Umkehr der Verfahrriichtung einhergehen.
Bit 3	Zählrichtung invertiert	Wenn gesetzt, werden positive Encoder-Schritte als negative behandelt, und umgekehrt.
Bit 4	Software-Endschalter nach Referenzfahrt wirksam	Wenn gesetzt, gelten nach der Referenzfahrt die mit Parameter 14 und 15 definierten Software-Endschalter, d.h. wenn der Achscontroller eine Zielposition vorgegeben bekommt, die außerhalb des durch die Software-Endschalter definierten Bereichs liegt, wird eine Störung ausgelöst, und die Achse fährt nicht los.
Bit 5	Losekompensation freifahren vor Interpolation	Wenn gesetzt, und Sie mit Losekompensation arbeiten, fährt der Achscontroller vor Beginn einer interpolierten Bewegung automatisch die Lose frei.
Bit 8	Wertigkeit 1.000 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/1 mm bzw. 1/1 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 9	Wertigkeit 0.100 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/10 mm bzw. 1/10 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 10	Wertigkeit 0.010 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/100 mm bzw. 1/100 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 11	Wertigkeit 0.001 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/1000 mm bzw. 1/1000 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 12	Wertigkeit 0.0001 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/10000 mm bzw. 1/10000 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
31	Impulse Meßsystem pro Umdrehung	Mit diesem Parameter definieren Sie die Auslegung Ihres Meßsystems. Geben Sie hier an, wieviele Impulse der von Ihnen verwendete Encoder bei einer vollständigen Umdrehung des Motors liefert.
32	Umrechnungssystem Nenner	Mit diesem Parameter wird die physikalische Auslegung Ihres mechanischen Systems angegeben. Eine nähere Beschreibung der Funktionsweise des Umrechnungssystems finden Sie im Kapitel • - Einrichtung des Zählsystems (Seite 190).
33	Umrechnungssystem Zähler	Mit diesem Parameter wird die physikalische Auslegung Ihres mechanischen Systems angegeben. Eine nähere Beschreibung der Funktionsweise des Umrechnungssystems finden Sie im Kapitel • - Einrichtung des Zählsystems (Seite 190).
34	Reduzierungswert für Endlosantriebe	Der Reduzierungswert für Endlosantriebe gibt bei Verwendung eines Endlosantriebes an, bei welchem Positionswert die aktuelle Position wieder auf den Wert 0 gesetzt werden soll. Bei Drehantrieben ist diese in der Regel 360°.

Nummer	Bezeichnung	Bedeutung
35	Losekompensation	Verwenden Sie die Losekompensation, um mechanische Ungenauigkeiten Ihres Systems auszugleichen. In erster Linie findet die Losekompensation bei Endlosantrieben Anwendung, weil hier der Fehler durch Getriebelose verstärkt auftritt.
37	Parametrierung Geschwindigkeitssystem (bitcodiert)	In diesem Parameter werden die verwendeten Maßeinheiten für das Geschwindigkeitssystem angegeben. Prinzipiell ähnelt dieser Parameter der Einstellung des Zählsystems mit dem Parameter 25. Alle Einstellungen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Geschwindigkeit in mm bzw. Inch pro Sekunde	Wenn gesetzt, werden alle Positionswerte als Angaben in Inch/s betrachtet. Wenn nicht gesetzt, werden alle Positionswerte als Angaben in mm/s betrachtet.
Bit 1	Geschwindigkeitsvorsteuerung aktiv	Wenn gesetzt, werden Geschwindigkeits- und Rampenwerte anhand der zu fahrenden Strecke vom Achscontroller automatisch im Voraus berechnet.
Bit 8	Wertigkeit 1.000 (mm/inch) für Geschwindigkeitswerte	Wenn gesetzt, werden Geschwindigkeitsangaben als 1/1 mm/s bzw. 1/1 inch/s (je nach Auswahl über Bit 0) behandelt.
Bit 9	Wertigkeit 0.100 (mm/inch) für Geschwindigkeitswerte	Wenn gesetzt, werden Geschwindigkeitsangaben als 1/10 mm/s bzw. 1/10 inch/s (je nach Auswahl über Bit 0) behandelt.
38	Maximale Drehzahl Motor	Geben Sie hier die maximale Drehzahl Ihres Motors an. Entsprechende Angaben finden Sie im Datenblatt zu Ihrem Motor.
40	Parametrierung Eingänge (bitcodiert)	Mit Hilfe dieses Parameters geben Sie an, welche physikalischen Eingänge an dem Achscontroller angeschlossen sind. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Endschalter Plus vorhanden	Wenn gesetzt, geht der Achscontroller davon aus, daß ein Endschalter für die maximale Plus-Position angebracht und verdrahtet ist. Setzen Sie diese Option nicht, wenn Sie den Endschalter als Referenzschalter verwenden.
Bit 1	Endschalter Minus vorhanden	Wenn gesetzt, geht der Achscontroller davon aus, daß ein Endschalter für die maximale Minus-Position angebracht und verdrahtet ist. Setzen Sie diese Option nicht, wenn Sie den Endschalter als Referenzschalter verwenden.
Bit 2	Endschalter sind logisch getauscht	Wenn gesetzt, behandelt der Achscontroller den Endschalter Plus als Endschalter Minus, und umgekehrt.
Bit 3	Endschalter haben Notaus-Funktion	Wenn gesetzt, wird bei Schalten eines Endschalters automatisch Notaus ausgelöst.
Bit 8	Endschalter Plus ist als Schließer ausgeführt	Wenn gesetzt, wird der Endschalter als elektrischer Schließer betrachtet. Ansonsten wird der Endschalter als Öffner behandelt.
Bit 9	Endschalter Minus ist als Schließer ausgeführt	Wenn gesetzt, wird der Endschalter als elektrischer Schließer betrachtet. Ansonsten wird der Endschalter als Öffner behandelt.
Bit 10	Eingang Leistungsteil Betriebsbereit wird im	Wenn gesetzt, geht die Steuerung davon aus, daß im Falle einer Störung des Leistungsteils der Eingang "LT Betriebsbereit" am

Nummer	Bezeichnung	Bedeutung
	Fehlerfall bestromt	Achscontroller nicht stromlos geschaltet wird.
Bit 11	Referenzschalter ist als Schließer ausgeführt	Wenn gesetzt, wird der Referenzschalter als elektrischer Schließer betrachtet.
41	Parametrierung Referenzfahrt (bitcodiert)	Mit diesem Parameter definieren Sie, wie sich das System bei der Referenzfahrt verhalten soll. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Endschalter Plus wird als Referenzschalter benutzt	Wenn gesetzt, wird der Endschalter Plus als Referenzschalter verwendet. Ein ggf. auf den Referenz-Eingang der Steuerung verdrahteter Schalter wird ignoriert. Bitte beachten Sie, daß bei Verwenden dieser Option Endschalter Plus nicht als vorhanden definiert werden darf (Parameter 40, Bit 0).
Bit 1	Endschalter Minus wird als Referenzschalter benutzt	Wenn gesetzt, wird der Endschalter Minus als Referenzschalter verwendet. Ein ggf. auf den Referenz-Eingang der Steuerung verdrahteter Schalter wird ignoriert. Bitte beachten Sie, daß bei Verwenden dieser Option Endschalter Minus nicht als vorhanden definiert werden darf (Parameter 40, Bit 1).
Bit 4	Referenzfahrt erfolgt in Plusrichtung	Wenn gesetzt, erfolgt die Referenzfahrt nicht in Minus-, sondern in Plus-Richtung.
Bit 5	Referenzsignal des Encoders wird ausgewertet	Wenn gesetzt, wird ein ggf. übermitteltes Referenzsignal des angeschlossenen Encoders ausgewertet. Verwenden Sie diese Option, wenn Sie keinen an den Achscontroller verdrahteten Referenzschalter verwenden, und der Encoder ein entsprechendes Signal liefern kann.
42	Parametrierung Sonderfunktionen (bitcodiert)	Mit diesem Parameter können Sie einzelne Sonderfunktionen der Steuerung aktivieren. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Zentraler Override wirksam	Wenn gesetzt, wird der angegebene Override-Wert auf die jeweils aktive Geschwindigkeit angewendet. Wenn nicht gesetzt, ist der Override-Wert ohne Bedeutung.
Bit 1	Stromreduzierung eingeschaltet	Wenn gesetzt, arbeitet das System mit einer Stromreduzierung bei der Ausgabe des Sollwertes.
56	Momentbegrenzung	Geben Sie hier das maximal zulässige Drehmoment für den verwendeten Motor an.
50	Parametrierung Beschleunigung (bitcodiert)	Die Parameter setzt die Einstellungen für das Beschleunigungssystem des Achscontrollers. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Dynamische Rampenanpassung aktiv	Wenn gesetzt, errechnet und verwendet das System eine dynamische Rampenanpassung. Eine entsprechende Begrenzung der niedrigsten zulässigen Werte für die dynamische Rampenanpassung definieren Sie im Parameter 51. Der höchste zulässige Wert wird durch die maximale Rampe gesetzt, die Sie im Parameter 53 definieren.



Nummer	Bezeichnung	Bedeutung
Bit 1	Beschleunigungsvorsteuerung aktiv	Wenn gesetzt, errechnet und verwendet das System die Beschleunigungsvorsteuerung, die Sie im Parameter 54 angegeben haben.
Bit 2	Sinus <sup>2</sup> Beschleunigung	Wenn gesetzt, arbeitet das System mit der integrierten Sinus <sup>2</sup> -Beschleunigung. Wenn nicht gesetzt, wird eine lineare Beschleunigung verwendet.
Bit 3	Wertigkeit Beschleunigung 10mm/s <sup>2</sup>	Wenn gesetzt, werden Beschleunigungswerte in der Einheit 10 mm/s <sup>2</sup> behandelt. Ansonsten werden Beschleunigungswerte in der Einheit 1 mm/s <sup>2</sup> behandelt.
51	Begrenzung für dynamische Rampenanpassung	Gibt an, bis zu welchem Verhältnis im Vergleich zur definierten Rampe das System eine dynamische Rampe errechnen darf. Mit diesem Wert geben Sie das untere Limit für die dynamische Rampenanpassung in % zur definierten Rampe an. Die dynamisch angepaßte Rampe wird niemals unter dem hier definierten Wert liegen.
52	Anpassung Bremsrampe in Prozent zur Beschleunigungsrampe	Gibt das Verhältnis der Bremsrampe im Vergleich zur Beschleunigungsrampe in % an. Normalerweise werden Sie diesen Wert auf 100% setzen, d.h. daß das System mit der gleichen Rampe bremst und beschleunigt.
53	Maximale Rampe	Gibt die vom System maximal zu verwendende Rampe an. Verwenden Sie diesen Parameter, um Limitierungen des Antriebs oder Ihres mechanischen Systems in der Steuerung zu definieren.
54	Faktor Beschleunigungsvorsteuerung	Gibt den zu verwendenden Faktor für die Beschleunigungsvorsteuerung an. Diese Funktion ist nur aktiv, wenn das Bit 1 im Parameter 50 gesetzt ist.

n Tabelle 44 – Achsparameter Servomotorcontroller

## n Übersicht Achspanparameter für Schrittmotorcontroller

Nummer	Bezeichnung	Bedeutung
10	Schleppfehler	Gibt an, wie groß der Schleppfehler für diese Achse werden darf, bevor eine Achsstörung ausgelöst wird.
11	Sollgeschwindigkeit	Gibt die gewünschte Sollgeschwindigkeit an. Die Einstellung wird nur bei Verwendung des Profils 0, d.h. ohne Beschleunigungsvorsteuerung, verwendet.
12	Sollbeschleunigung	Gibt den gewünschten Beschleunigungswert (die Rampe) an.
13	Nullpunkt-Offset	Gibt an, um wieviel Maßeinheiten der logische Nullpunkt von der Position des physikalischen Nullpunkts (Referenzschalter) entfernt liegt. Wird z.B. verwendet, um eine Werkzeugkorrektur durchzuführen.
14	Software-Endschalter Plus	Dieser Parameter enthält die Position des Software-Endschalters Plus im jeweiligen Umrechnungssystem. Ob dieser Software-Endschalter verwendet wird, entscheidet das Bit 4 im Parameter 30.
15	Software-Endschalter Minus	Dieser Parameter enthält die Position des Software-Endschalters Minus im jeweiligen Umrechnungssystem. Ob dieser Software-Endschalter verwendet wird, entscheidet das Bit 4 im Parameter 30.
23	F <sub>MIN</sub> in Hz Start-/ Stopfrequenz	Dieser Parameter enthält die minimale Start-/ Stopfrequenz für den Schrittmotor.
24	Nullpunkt-Überlauf	Wenn das Bit 5 im Parameter 41 gesetzt ist, gibt dieser Parameter den Überlaufweg für die klassische Schrittmotornullung an.
25	In-Position Fenster (Stillstandsfenster)	Definiert das In-Positionsfenster in Maßeinheiten des aktuellen Zählsystems. Dieses Fenster wird nur dann ausgewertet, wenn das Bit 3 des Parameters 20 gesetzt ist.
30	Parametrierung Zählsystem (bitcodiert)	Mit diesem Parameter definieren Sie von Ihnen verwendete Zähl- und Maßsystem. Alle Werte innerhalb dieses Parameters sind bitcodiert, d.h. jede Information belegt ein oder mehrere Bits. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Umrechnung für Zahnriemen, Zahnstangen usw. (PI)	Wenn gesetzt, werden alle Positionswerte automatisch in Positionen auf einer Kreisbahn umgerechnet. Sie benötigen diese Umrechnung, wenn Sie z.B. einen Zahnriemen- oder Zahnstangenantrieb parametrieren möchten.
Bit 1	Umrechnung erfolgt in Inch	Wenn gesetzt, werden alle Positionswerte als Angaben in Inch betrachtet. Wenn nicht gesetzt, werden alle Positionswerte als Angaben in mm betrachtet.
Bit 2	Endlosbetrieb mit Reduzierung und Verfahrensoptimierung	Wenn gesetzt, wird im 360°-Betrieb automatisch die kürzeste Fahrstrecke ausgesucht. Dies kann mit einer automatischen Umkehr der Verfahrenrichtung einhergehen.
Bit 3	Zählrichtung invertiert	Wenn gesetzt, werden positive Encoder-Schritte als negative behandelt, und umgekehrt.
Bit 4	Software-Endschalter nach Referenzfahrt wirksam	Wenn gesetzt, gelten nach der Referenzfahrt die mit Parameter 14 und 15 definierten Software-Endschalter, d.h. wenn der Achscontroller eine Zielposition vorgegeben bekommt, die außerhalb des durch die Software-Endschalter definierten

Nummer	Bezeichnung	Bedeutung
		Bereichs liegt, wird eine Störung ausgelöst, und die Achse fährt nicht los.
Bit 8	Wertigkeit 1.000 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/1 mm bzw. 1/1 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 9	Wertigkeit 0.100 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/10 mm bzw. 1/10 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 10	Wertigkeit 0.010 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/100 mm bzw. 1/100 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 11	Wertigkeit 0.001 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/1000 mm bzw. 1/1000 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
Bit 12	Wertigkeit 0.0001 (mm/inch) für Positionswerte	Wenn gesetzt, werden alle Positionswerte als Angaben in 1/10000 mm bzw. 1/10000 inch (je nach gewählter Umrechnung, siehe Bit 1) behandelt.
31	Schritte pro Umdrehung	Gibt an, wieviele Schritte der Motor für eine vollständige Umdrehung benötigt.
32	Umrechnungssystem Nenner	Mit diesem Parameter wird die physikalische Auslegung Ihres mechanischen Systems angegeben. Eine nähere Beschreibung der Funktionsweise des Umrechnungssystems finden Sie im Kapitel • - Einrichtung des Zählsystems (Seite 190).
33	Umrechnungssystem Zähler	Mit diesem Parameter wird die physikalische Auslegung Ihres mechanischen Systems angegeben. Eine nähere Beschreibung der Funktionsweise des Umrechnungssystems finden Sie im Kapitel • - Einrichtung des Zählsystems (Seite 190).
34	Reduzierungswert für Endlosantriebe	Der Reduzierungswert für Endlosantriebe gibt bei Verwendung eines Endlosantriebes an, bei welchem Positionswert die aktuelle Position wieder auf der Wert 0 gesetzt werden soll. Bei Drehantrieben ist die in der Regel 360°.
35	Losekompensation	Verwenden Sie die Losekompensation, um mechanische Ungenauigkeiten Ihres Systems auszugleichen. In erster Linie findet die Losekompensation bei Endlosantrieben Anwendung, weil hier der Fehler durch Getriebelose verstärkt auftritt.
37	Parametrierung Geschwindigkeitssystem (bitcodiert)	In diesem Parameter werden die verwendeten Maßeinheiten für das Geschwindigkeitssystem angegeben. Prinzipiell ähnelt dieser Parameter der Einstellung des Zählsystems mit dem Parameter 25. Alle Einstellungen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Geschwindigkeit in mm bzw. Inch pro Sekunde	Wenn gesetzt, werden alle Positionswerte als Angaben in Inch/s betrachtet. Wenn nicht gesetzt, werden alle Positionswerte als Angaben in mm/s betrachtet.
Bit 8	Wertigkeit 1.000 (mm/inch) für Geschwindigkeitswerte	Wenn gesetzt, werden Geschwindigkeitsangaben als 1/1 mm/s bzw. 1/1 inch/s (je nach Auswahl über Bit 0) behandelt.
Bit 9	Wertigkeit 0.100 (mm/inch) für	Wenn gesetzt, werden Geschwindigkeitsangaben als 1/10 mm/s

Nummer	Bezeichnung	Bedeutung
	Geschwindigkeitswerte	bzw. 1/10 inch/s (je nach Auswahl über Bit 0) behandelt.
38	Max. Umdrehungen pro Minute	Dieser Parameter enthält die maximale Anzahl der Umdrehungen pro Minute für den Schrittmotor (Drehzahlbegrenzung).
40	Parametrierung Eingänge (bitcodiert)	Mit Hilfe dieses Parameters geben Sie an, welche physikalischen Eingänge an dem Achscontroller angeschlossen sind. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Endschalter Plus vorhanden	Wenn gesetzt, geht der Achscontroller davon aus, daß ein Endschalter für die maximale Plus-Position angebracht und verdrahtet ist. Setzen Sie diese Option nicht, wenn Sie den Endschalter als Referenzschalter verwenden.
Bit 1	Endschalter Minus vorhanden	Wenn gesetzt, geht der Achscontroller davon aus, daß ein Endschalter für die maximale Minus-Position angebracht und verdrahtet ist. Setzen Sie diese Option nicht, wenn Sie den Endschalter als Referenzschalter verwenden.
Bit 2	Endschalter sind logisch getauscht	Wenn gesetzt, behandelt der Achscontroller den Endschalter Plus als Endschalter Minus, und umgekehrt.
Bit 3	Endschalter haben Notaus-Funktion	Wenn gesetzt, wird bei Schalten eines Endschalters automatisch Notaus ausgelöst.
Bit 8	Endschalter Plus ist als Schließer ausgeführt	Wenn gesetzt, wird der Endschalter als elektrischer Schließer betrachtet. Ansonsten wird der Endschalter als Öffner behandelt.
Bit 9	Endschalter Minus ist als Schließer ausgeführt	Wenn gesetzt, wird der Endschalter als elektrischer Schließer betrachtet. Ansonsten wird der Endschalter als Öffner behandelt.
Bit 11	Referenzschalter ist als Schließer ausgeführt	Wenn gesetzt, wird der Referenzschalter als elektrischer Schließer betrachtet.
41	Parametrierung Referenzfahrt (bitcodiert)	Mit diesem Parameter definieren Sie, wie sich das System bei der Referenzfahrt verhalten soll. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Endschalter Plus wird als Referenzschalter benutzt	Wenn gesetzt, wird der Endschalter Plus als Referenzschalter verwendet. Ein ggf. auf den Referenz-Eingang der Steuerung verdrahteter Schalter wird ignoriert. Bitte beachten Sie, daß bei Verwenden dieser Option Endschalter Plus nicht als vorhanden definiert werden darf (Parameter 40, Bit 0).
Bit 1	Endschalter Minus wird als Referenzschalter benutzt	Wenn gesetzt, wird der Endschalter Minus als Referenzschalter verwendet. Ein ggf. auf den Referenz-Eingang der Steuerung verdrahteter Schalter wird ignoriert. Bitte beachten Sie, daß bei Verwenden dieser Option Endschalter Minus nicht als vorhanden definiert werden darf (Parameter 40, Bit 1).
Bit 4	Referenzfahrt erfolgt in Plusrichtung	Wenn gesetzt, erfolgt die Referenzfahrt nicht in Minus-, sondern in Plus-Richtung.
Bit 5	Klassische Nullung durchführen	Wenn gesetzt, wird bei der Referenzfahrt nicht die Nachbildung einer Servomotor-Nullung verwendet (d.h. mit anschließendem Freifahren des Initiators), sondern die klassische Schrittmotor-Nullung: die Achse fährt auf den Initiator und stoppt nach dem im

Nummer	Bezeichnung	Bedeutung
		Parameter 24 angegebenen Überlaufweg.
42	Parametrierung Sonderfunktionen (bitcodiert)	Mit diesem Parameter können Sie einzelne Sonderfunktionen der Steuerung aktivieren. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Zentraler Override wirksam	Wenn gesetzt, wird der angegebene Override-Wert auf die jeweils aktive Geschwindigkeit angewendet. Wenn nicht gesetzt, ist der Override-Wert ohne Bedeutung.
50	Parametrierung Beschleunigung (bitcodiert)	Die Parameter setzt die Einstellungen für das Beschleunigungssystem des Achscontrollers. Alle diese Informationen sind bitcodiert, d.h. jede Information ist in einem einzelnen Bit dieses Parameters enthalten. Deshalb ist es nicht sinnvoll, diese Parameter aus einem SPS-Programm heraus zu verändern.
Bit 0	Dynamische Rampenanpassung aktiv	Wenn gesetzt, errechnet und verwendet das System eine dynamische Rampenanpassung. Eine entsprechende Begrenzung für die niedrigsten zulässigen Wert für die dynamische Rampenanpassung definieren Sie im Parameter 51. Der höchste zulässige Wert wird durch die maximale Rampe gesetzt, die Sie im Parameter 53 definieren.
Bit 2	Sinus <sup>2</sup> Beschleunigung	Wenn gesetzt, arbeitet das System mit der integrierten Sinus <sup>2</sup> -Beschleunigung. Wenn nicht gesetzt, wird eine lineare Beschleunigung verwendet.
Bit 3	Wertigkeit Beschleunigung 10mm/s <sup>2</sup>	Wenn gesetzt, werden Beschleunigungswerte in der Einheit 10 mm/s <sup>2</sup> behandelt. Ansonsten werden Beschleunigungswerte in der Einheit 1 mm/s <sup>2</sup> behandelt.
51	Begrenzung für dynamische Rampenanpassung	Gibt an, bis zu welchem Verhältnis im Vergleich zur definierten Rampe das System eine dynamische Rampe errechnen darf. Mit diesem Wert geben Sie das untere Limit für die dynamische Rampenanpassung in % zur definierten Rampe an. Die dynamisch angepaßte Rampe wird niemals unter dem hier definierten Wert liegen.
52	Anpassung Bremsrampe in Prozent zur Beschleunigungsrampe	Gibt das Verhältnis der Bremsrampe im Vergleich zur Beschleunigungsrampe in % an. Normalerweise werden Sie diesen Wert auf 100% setzen, d.h. daß das System mit der gleichen Rampe bremst und beschleunigt.
53	Maximale Rampe	Gibt die vom System maximal zu verwendende Rampe an. Verwenden Sie diesen Parameter, um Limitierungen des Antriebs oder Ihres mechanischen Systems in der Steuerung zu definieren.

n Tabelle 45 – Achsparameter Schrittmotorcontroller

## n Einrichtung des Zählsystems

Das MC200 System verfügt über eine sehr flexible Möglichkeit, daß verwendete Zählsystem an Ihre eigenen Bedürfnisse anzupassen. Die entsprechende Einrichtung wird mit Hilfe der Achspanparameter 32 und 33 eingerichtet.

### Standardumrechnung

Die Standardumrechnung ist bei Spindeln und ähnlichen Antriebssystemen anzuwenden. Im Parameter 32 wird die Spindelsteigung in 1/100mm angegeben. Im Parameter 33 wird die Getriebeübersetzung des vorgeschalteten Getriebes, multipliziert mit 100, eingetragen.

### Beispiel für die Standardumrechnung

n Spindelsteigung beträgt 5mm

n Die Getriebeübersetzung ist 1:3

$$\Rightarrow \text{Parameter 32} = 5\text{mm} / 1/100\text{mm} = 500$$

$$\Rightarrow \text{Parameter 33} = 3 \times 100 = 300$$

### Umrechnung mit $\pi$

Die Umrechnung mit  $\pi$  ist bei Zahnriemen, Zahnstangen und ähnlichen Antriebssystemen anzuwenden. Im Parameter 32 wird der Teilkreisdurchmesser des Ritzels in 1/100mm angegeben. Im Parameter 33 wird die Getriebeübersetzung des vorgeschalteten Getriebes, multipliziert mit 100, eingetragen.

### Beispiel für die Umrechnung mit $\pi$

n Der Teilkreisdurchmesser beträgt 57,50 mm

n Die Getriebeübersetzung ist 1:9,25

$$\Rightarrow \text{Parameter 32} = 57,50\text{mm} / 1/100\text{mm} = 5750$$

$$\Rightarrow \text{Parameter 33} = 9,25 \times 100 = 925$$

### Ungerade Werte

Sollte bei der Berechnung der Parameter ein ungerader Wert herauskommen, multiplizieren Sie die Werte für die beiden Parameter solange gleichermaßen mit 10, bis Sie einen geraden Wert erhalten.

## 5.3 Parameter serielles Modul

Das MC200 System ermöglicht den Betrieb von bis zu 8 seriellen Erweiterungsmodulen an einer Steuerung. Das gewünschte serielle Protokoll sowie die Baudrate des Moduls werden über Parameter ausgewählt.

### Bitcodierte Parameter

Die Parameter des seriellen Moduls sind bitcodiert abgespeichert. Dies bedeutet: nicht die eigentlich gespeicherte Zahl ist entscheidend, sondern vielmehr die einzelnen Bits innerhalb des 32-Bit Wertes. Sie dürfen in den Parametern für das serielle Modul z.B. niemals das Bit 0 und das Bit 1 gleichzeitig setzen: dies würde bedeuten, daß das Modul sowohl mit einer Baudrate von 9600 Baud als auch mit einer Baudrate von 19200 Baud arbeiten soll und zu unvorhersehbaren Ergebnissen führen.

### n Berechnung der Parameter-Nummer

Die Konfiguration der seriellen Module ist ab dem Parameter 60 gespeichert. Dies bedeutet: die Konfiguration des zweiten Moduls ist im Parameter 61 enthalten, die des dritten in Parameter 62 usw. Zur Ermittlung der korrekten Parameter-Nummer verwenden Sie bitte folgende Formel:

$$\text{Parameter-Nummer} = 59 + \text{Modul-Nummer}$$

Soll also z.B. die Konfiguration des dritten angeschlossenen seriellen geändert werden, so würde sich die Parameter-Nummer wie folgt berechnen:

$$\text{Parameter-Nummer} = 59 + \text{Modul-Nummer}$$

$$\text{Parameter-Nummer} = 59 + 3$$

$$\text{Parameter-Nummer} = 62$$

### n Übersicht Parameter serielles Modul

Nummer	Bezeichnung	Bedeutung
60	Konfiguration serielles Modul (bitcodiert)	Die vollständige Konfiguration des seriellen Moduls ist in diesem Parameter enthalten.
Bit 0	Baudrate 9600 Baud	Das serielle Modul arbeitet mit einer Baudrate von 9600 Baud.
Bit 1	Baudrate 19200 Baud	Das serielle Modul arbeitet mit einer Baudrate von 19200 Baud.
Bit 2	Baudrate 38400 Baud	Das serielle Modul arbeitet mit einer Baudrate von 38400 Baud.
Bit 8	7-Bit Datenmodus	Wenn gesetzt, arbeitet das Modul im 7-Bit Datenmodus. Wenn nicht gesetzt, arbeitet das Modul im 8-Bit Datenmodus.
Bit 9	Gerade Parität	Wenn gesetzt, arbeitet das Modul mit gerader Parität (even parity) zur Datenabsicherung. Wenn Bit 9 und Bit 10 nicht gesetzt sind, arbeitet das Modul ohne Parität.
Bit 10	Ungerade Parität	Wenn gesetzt, arbeitet das Modul mit ungerader Parität (odd parity) zur Datenabsicherung. Wenn Bit 9 und Bit 10 nicht gesetzt sind, arbeitet das Modul ohne Parität.
Bit 11	RTS/CTS Handshake	Wenn gesetzt, arbeitet das Modul mit einem RTS/CTS Hardware-Handshake. Wenn nicht gesetzt, arbeitet das Modul ohne Hardware-Handshake.

n Tabelle 46 – Parameter serielles Modul

### Siehe auch

SETPAR (Seite 115)

GETPAR (Seite 70)

**n Raum für Ihre Notizen**



# Kapitel 6 Systemdaten

Wie jedes andere SPS-System auch verfügt die MC200 über Variablen und Merker:

- n Merker sind Bitvariablen, d.h. sie können ausschließlich den Zustand "ein" oder den Zustand "aus" speichern. Merker werden dann verwendet, wenn lediglich der Zustand eines Aggregats abgebildet werden soll, Sie den Rückgabewert eines Unterprogramms speichern möchten oder in ähnlichen Fällen, die lediglich eine "ein/aus" Information benötigen. Das MC200 System verfügt über insgesamt 4096 Merker, von denen Sie 2048 frei verwenden können.
- n Variablen speichern vollständige Integer-Zahlen in 32 Bit Länge, d.h. in einem Wertebereich von -2147483648 bis 2147483647. Nachkommastellen können in Variablen nicht gespeichert werden. Das MC200 System verfügt insgesamt über 8192 Variablen, von denen Sie 8072 frei verwenden können.
- n Parameter sind spezielle Variablen, die vom Betriebssystem der Steuerung verwendet werden. Diese Variablen stehen Ihnen nicht direkt im SPS-Programm zur Verfügung, weil die enthaltenen Informationen ständig vom System benötigt werden und zumeist auch nicht direkt verändert werden dürfen. Grundsätzlich speichern Parameter, wie auch Variablen, vollständige Integer-Zahlen in 32 Bit Länge, jedoch wird der gültige Zahlenbereich in den meisten Fällen durch entsprechende Wertebereiche für die einzelnen Parameter eingeschränkt.

## n Vom System reservierte Speicherbereiche

Wie oben erwähnt, sind einige der MC200 Variablen fest vom System belegt. Diese Variablen und Merker dürfen von Ihnen nicht im Programm verwendet werden, da dies zu Störungen des SPS-Ablaufs führen würde. Folgende Speicherbereiche sind vom System reserviert:

- n Die Variablen 1 bis 149 sind vom System reserviert und dürfen nur wie in den nachfolgenden Abschnitten verwendet werden.
- n Die Merker 2049 bis 4095 sind vom System reserviert und dürfen nur wie in den nachfolgenden Abschnitten verwendet werden.

## n Lese-/Schreibzugriff

Nicht alle in den folgenden Abschnitten beschriebenen Systemmerker, -Variablen und Parameter können vom SPS-Programm oder vom PC aus geschrieben werden. Deshalb findet sich in den Tabellen auf den nächsten Seiten in der jeweils letzten Spalte eines der folgenden Symbole:

**y** - Merker, Variable oder Parameter darf nur gelesen werden.

**p** - Merker, Variable oder Parameter darf sowohl geschrieben als auch gelesen werden.

## 6.1 Bitergebnisschieberegister

Bei der Ausführung von Befehlen, die den Bitergebnisspeicher beeinflussen, wird der Zustand des BES-Schieberegisters um eine Stelle nach links geschoben. Der neue Zustand des Bitergebnisses wird in M\_BES0 abgelegt.

Funktion	Name	Adresse	
Bitergebnisschieberegister 1	M_BES0 (aktuelles Bitergebnis)	Merker 2049	ŷ
Bitergebnisschieberegister 2	M_BES1	Merker 2050	ŷ
Bitergebnisschieberegister 3	M_BES2	Merker 2051	ŷ
Bitergebnisschieberegister 4	M_BES3	Merker 2052	ŷ
Bitergebnisschieberegister 5	M_BES4	Merker 2053	ŷ
Bitergebnisschieberegister 6	M_BES5	Merker 2054	ŷ
Bitergebnisschieberegister 7	M_BES6	Merker 2055	ŷ
Bitergebnisschieberegister 8	M_BES7	Merker 2056	ŷ
Bitergebnisschieberegister 9	M_BES8	Merker 2057	ŷ
Bitergebnisschieberegister 10	M_BES9	Merker 2058	ŷ
Bitergebnisschieberegister 11	M_BES10	Merker 2059	ŷ
Bitergebnisschieberegister 12	M_BES11	Merker 2060	ŷ
Bitergebnisschieberegister 13	M_BES12	Merker 2061	ŷ
Bitergebnisschieberegister 14	M_BES13	Merker 2062	ŷ
Bitergebnisschieberegister 15	M_BES14	Merker 2063	ŷ
Bitergebnisschieberegister 16	M_BES15	Merker 2064	ŷ

n Tabelle 47 – Systemmerker des Bitergebnisspeichers

## 6.2 Ergebnismarker- und Variablen

Ergebnisvariablen sind Variablen in denen nach einer mathematischen oder nach einem Vergleich das Ergebnis abgespeichert wird. Sie dienen dafür, daß die Werte der Operatoren nach der mathematischen oder nach dem Vergleich unverändert bleiben.

Funktion	Name	Adresse	
Variablenergebnis	VARERG	Variable 1	þ
Divisionsrest	DIV_REST	Variable 2	ý
Multiplikationsüberlauf	MUL_REST	Variable 3	ý
Variablenvergleichsergebnis gleich	M_GLEICH	Merker 2065	ý
Variablenvergleichsergebnis kleiner	M_KLEINER	Merker 2066	ý
Variablenvergleichsergebnis größer	M_GROESSER	Merker 2067	ý

n Tabelle 48 – Ergebnismarker und -Variablen

### n Variable VARERG – Generelle Ergebnisvariable

Bei einer mathematischen Operation (Addition, Subtraktion, Multiplikation, Division) oder bei einem Vergleichsbefehl oder bei Schiebebefehle, wird das Ergebnis in der Ergebnisvariablen VARERG gespeichert. Die Werte mit denen die mathematische Operation durchgeführt wird werden dabei nicht beeinflusst.

### n Variable DIV\_REST – Divisionsrest

Beim Dividieren zweier Werte wird in der Variable VAREG stets nur das ganzzahlige Ergebnis gespeichert. Der Rest der jeweiligen Division wird in der Variable DIV\_REST gespeichert.

#### Beispiel

Sie dividieren 5 durch 3. Daraufhin erhalten Sie folgende Ergebnisse:

⇒ in VARERG: 1

⇒ in DIV\_REST: 2

## n Variable MUL\_REST – Multiplikationsüberlauf

Beim Multiplizieren zweier Werte wird in der Variable VARERG immer das Ergebnis der Multiplikation gespeichert. Sollte bei einer Multiplikation der Wert des Ergebnisses grösser sein als der maximal in 32 Bit enthaltene Wert von -2147483648 bis 2147483647, so steht in der Variable MUL\_REST der Rest des Ergebnisses ab der Bitstelle 33. Dies bedeutet:

**Wann immer Sie eine Multiplikation durchführen, deren Ergebnis möglicherweise den größtmöglichen Zahlenbereich einer Variable überschreitet, sollten Sie anschließend den Inhalt der Variable MUL\_REST prüfen.**

Ist die Variable MUL\_REST nach einer Multiplikation gleich Null, so ist kein Überlauf bei der Multiplikation aufgetreten. Ist die Variable ungleich Null, dann ist bei der Multiplikation ein Überlauf aufgetreten. Die Variable MUL\_REST enthält dann den Teil des Multiplikationsergebnisses, der nicht mehr in 32 Bit gespeichert werden konnte.

### Beispiel

```
LAD_VA    MUL_1, 100000          // Multiplikator 1 mit Wert 100000 laden
LAD_VA    MUL_2, 100000          // Multiplikator 2 mit Wert 100000 laden
MUL_VV    MUL_1, MUL_2           // Multiplikation durchführen
VERG_VA    MUL_REST, 0           // Überlauf aufgetreten?
LAD_M     M_GLEICH               // dann ist M_GLEICH nicht ein
SPRINGN   FEHLER                // Multiplikationsüberlauf behandeln

// In diesem Beispiel sind anschließend folgende Werte in den Ergebnisvariablen
// gespeichert:
//
// - VARERG = 1410065408
// - MUL_REST = 2
```

Bitte beachten Sie: das MC200 System kann grundsätzlich nur mit 32 Bit großen Werten arbeiten. Es gibt keine sinnvolle Möglichkeit, wie Sie eine größere Zahl innerhalb Ihres SPS-Programms verwalten können. Die Variable MUL\_REST sollte in erster Linie zur Kontrolle eines Fehlers, nicht aber gezielt im Rahmen einer Berechnung verwendet werden.

## n Vergleichsergebnisse

Im Variablenvergleichsergebnis steht das Ergebnis eines logischen Vergleichs zweier Werte:

n Wert 1 ist gleich groß wie Wert 2: Merker M\_GLEICH wird gesetzt

n Wert 1 ist kleiner wie Wert 2: Merker M\_KLEINER wird gesetzt

n Wert 1 ist größer wie Wert 2: Merker M\_GROESSER wird gesetzt

Die Vergleiche können mit Konstanten und natürlich auch mit Variablen durchgeführt werden.

## 6.3 Interne Systemdaten

Einige Basisinformationen der MC200 Steuerung werden Ihnen über die internen Systemdaten zur Verfügung gestellt:

Funktion	Name	Adresse	
Merker immer EIN	M_EIN	Merker 2080	y
Merker Reset: Wenn dieser Merker gesetzt wird, führt die Steuerung einen Software-Reset durch	M_RESET	Merker 2081	b
Fernwartungspasswort. Enthält den Zugangscode, der bei einem Zugriff auf die Steuerung via Telefonleitung benötigt wird. Ein Wert von 0 schaltet die Sicherung ab.	V_PASSWORD	Variable 10	b
Globaler Override. Enthält einen prozentualen Faktor von 0 bis 100, mit dem die Sollgeschwindigkeit der Achsen verrechnet wird. Mit diesem Wert kann die Geschwindigkeit aller Achsen, für die der Override aktiviert ist, gleichmäßig verändert werden	V_OVERRD	Variable 8	b

n Tabelle 49 – Systemvariablen und Parameter interne Systemdaten

## 6.4 SPS-Tasks

CPU-Module der MC200 Familie unterstützen bis zu vier voneinander unabhängige SPS-Tasks. Sie können per Systemmerker und –Variablen diese Tasks deaktivieren oder ihnen unterschiedliche Ausführungsprioritäten zuordnen.

Funktion	Beschreibung	Adresse	
Merker Taskwechselabschaltung: Wenn dieser Merker gesetzt wird, werden alle Parallelprogramme blockiert.	M_NOTASK	Merker 2082	<b>p</b>
Priorität Task 1: Ermöglicht den Ablauf der Task zur verlangsamen und damit den anderen Tasks automatisch mehr Rechenzeit zur Verfügung zu stellen. Je größer der Wert, desto langsamer die Task. Siehe auch LAD_P1 (Seite 78).	V_TASK1_P	Variable 43	<b>p</b>
Priorität Task 2: Ermöglicht den Ablauf der Task zur verlangsamen und damit den anderen Tasks automatisch mehr Rechenzeit zur Verfügung zu stellen. Je größer der Wert, desto langsamer die Task. Siehe auch LAD_P2 (Seite 79).	V_TASK2_P	Variable 44	<b>p</b>
Priorität Task 3: Ermöglicht den Ablauf der Task zur verlangsamen und damit den anderen Tasks automatisch mehr Rechenzeit zur Verfügung zu stellen. Je größer der Wert, desto langsamer die Task. Siehe auch LAD_P3 (Seite 80).	V_TASK3_P	Variable 45	<b>p</b>
Priorität Task 4: Ermöglicht den Ablauf der Task zur verlangsamen und damit den anderen Tasks automatisch mehr Rechenzeit zur Verfügung zu stellen. Je größer der Wert, desto langsamer die Task. Siehe auch LAD_P4 (Seite 81).	V_TASK4_P	Variable 46	<b>p</b>

**n** Tabelle 50 – Systemmerker und –Variablen SPS-Tasks

## 6.5 RS485 Kommunikation

Displays und dezentralisierte I/O-Module werden bei der MC200 Familie über einen RS485 Feldbus angebunden. Mit Hilfe folgender Systemmerker und –Variablen erhalten Sie erweiterte Informationen über den Zustand des Bussystems:

Funktion	Beschreibung	Adresse	
Merker RS485 aktiv: wird gesetzt wenn die CPU während der Initialisierung Erweiterungsmodule auf dem RS485 Bus erkannt hat.	M_RS485	Merker 2097	ý
Fehlermerker RS485 abgeschaltet: wird gesetzt, wenn während des Betriebs sehr viele Störungen auf dem RS485 Bus aufgetreten sind und das System deshalb die RS485 Kommunikation abgeschaltet hat.	M_RS485_ERR	Merker 2098	ý
Erster Fehlerzähler RS485 Kommunikation: Anzahl Request Timeouts seit letztem Reset	V_RS485_ERR1	Variable 66	p
Zweiter Fehlerzähler RS485 Kommunikation: Anzahl Acknowledge Timeouts seit letztem Reset	V_RS485_ERR2	Variable 67	p
Dritter Fehlerzähler RS485 Kommunikation: Anzahl Prüfsummen Fehler seit letztem Reset	V_RS485_ERR3	Variable 68	p

n Tabelle 51 – Systemmerker und –Variablen RS485 Kommunikation

## 6.6 Angeschlossene Module

Beim Systemstart ermittelt das CPU-Modul der MC200 Familie erweiterte Informationen über die angeschlossenen Module. Sie können einen Teil dieser Informationen über die im Folgenden aufgeführten Systemparameter abrufen. Die Erläuterungen zu den Fußnoten-Kennzeichnungen in der Tabelle (<sup>1</sup> bis <sup>4</sup>) erhalten Sie am Ende der Tabelle.

Funktion	Beschreibung	Adresse	
Hardware-Typ CPU-Modul	Modul-Typcode CPU-Modul <sup>1</sup> .	Parameter 1	Ÿ
Softwareversion CPU-Modul	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 2	Ÿ
Anzahl der verfügbaren Achsen (bitcodiert)	Gibt an, welche Achsen im System verfügbar sind. Dieser Parameter ist bitcodiert, für jede verfügbare Achse wird ein Bit gesetzt. Achse 1 verfügbar bedeutet gesetztes Bit 0, Achse 2 verfügbar gesetztes Bit 1 usw.	Parameter 4	Ÿ
Hardware-Typ 1. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 11	Ÿ
Softwareversion 1. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 12	Ÿ
Hardware-Typ 2. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 13	Ÿ
Softwareversion 2. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 14	Ÿ
Hardware-Typ 3. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 15	Ÿ
Softwareversion 3. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 16	Ÿ
Hardware-Typ 4. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 17	Ÿ
Softwareversion 4. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 18	Ÿ
Hardware-Typ 5. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 19	Ÿ
Softwareversion 5. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 20	Ÿ
Hardware-Typ 6. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 21	Ÿ
Softwareversion 6. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 22	Ÿ
Hardware-Typ 7. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 23	Ÿ
Softwareversion 7. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 24	Ÿ
Hardware-Typ 8. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 25	Ÿ
Softwareversion 8. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 26	Ÿ
Hardware-Typ 9. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 27	Ÿ
Softwareversion 9. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 28	Ÿ
Hardware-Typ 10. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 29	Ÿ
Softwareversion 10. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 30	Ÿ
Hardware-Typ 11. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 31	Ÿ
Softwareversion 11. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 32	Ÿ
Hardware-Typ 12. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 33	Ÿ
Softwareversion 12. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 34	Ÿ
Hardware-Typ 13. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 35	Ÿ



Funktion	Beschreibung	Adresse	
Softwareversion 13. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 36	✓
Hardware-Typ 14. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 37	✓
Softwareversion 14. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 38	✓
Hardware-Typ 15. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 39	✓
Softwareversion 15. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 40	✓
Hardware-Typ 16. angeschlossene Achse	Modul-Typcode Achsmodul <sup>3</sup> .	Parameter 41	✓
Softwareversion 16. angeschlossene Achse	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 42	✓
Hardware-Typ erstes angeschlossenes Bedienteil	Modul-Typcode Displaymodul <sup>4</sup> .	Parameter 43	✓
Softwareversion erstes angeschlossenes Bedienteil	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 44	✓
Hardware-Typ zweites angeschlossenes Bedienteil	Modul-Typcode Displaymodul <sup>4</sup> .	Parameter 45	✓
Softwareversion zweites angeschlossenes Bedienteil	BCD-codierte Versionsnummer <sup>2</sup> .	Parameter 46	✓

n Tabelle 52 – Systemparameter mit Informationen zu angeschlossenen Modulen

### Erläuterung der Fußnotenkennzeichnungen

<sup>1</sup> Eine Übersicht der gültigen Modul-Typcodes finden Sie im Anhang D - Typcodes MC200 Familie (Seite 258).

<sup>2</sup> Die Versionsnummer wird als "binary coded digit" abgespeichert, wobei die letzten beiden Stellen für die Unterversionsnummer reserviert sind. Dies ergibt folgende Berechnungsformel: Versionsnummer / 100 = Hauptversionsnummer, Versionsnummer % 100 = Unterversionsnummer. Für eine Betriebssystemversion von "2.41" würde also der Wert "241" gespeichert werden.

<sup>3</sup> Dieser Parameter kann unter folgenden Umständen "0" enthalten: die Achse ist nicht vorhanden, bei der Achse handelt es sich um die Schrittmotorachse des MC200CPU/C Moduls oder bei der Achse handelt es sich um die zweite Achse eines MC200SM2 Moduls. Eine Übersicht der gültigen Modul-Typcodes finden Sie im Anhang D - Typcodes MC200 Familie (Seite 258).

<sup>4</sup> Dieser Parameter enthält "0", wenn das entsprechende Bedienteil nicht angeschlossen ist.

## 6.7 Achsenstatus

Bei dem MC200 System können 16 Achsen angeschlossen werden. Der Status jeder Achse wird innerhalb der SPS in Merkern, Variablen und Parameter gespiegelt, das bedeutet:

**Für jede Achse gibt es im MC200 System einen reservierten Bereich von Merkern, Variablen und Parametern, über die Sie zu jedem Zeitpunkt den kompletten Achsenstatus, unter Verwendung der normalen Variablen- und Merkerbefehlen, abfragen können. Alle diese Variablen, Merker und Parameter dürfen nur gelesen werden. Um den Achsstatus zu verändern, verwenden Sie bitte ausschließlich die Achsbefehle, die im Kapitel 3.2 - Alphabetische Befehlsübersicht (Seite 47) beschrieben sind.**

### n Bitcodierte Achsstatus-Parameter

Innerhalb der MC200 gibt es zum Abfragen des Achsstatus zwei bitcodierte Parameter:

- n Basisparameter 2 enthält den aktuellen bitcodieren Fehler und
- n Basisparameter 3 enthält den aktuellen Zustand der physikalischen Eingänge sowie Warnhinweise.

Diese Parameter werden in der vorliegenden Dokumentation lediglich der Vollständigkeit halber erwähnt. Es wird dringend davon abgeraten, diese Parameter innerhalb Ihres SPS-Programms zu verwenden. MICRO DESIGN kann nicht gewährleisten, daß die Bitcodierung der Basisparameter 2 und 3 auch zukünftig in dieser Form bestehen bleibt – Änderungen jederzeit vorbehalten.

Verwenden Sie deshalb zum Abfragen des Achsstatus bitte ausschließlich – soweit möglich - die hierfür vorgesehenen Merker und Variablen. Alle relevanten Informationen werden vom Betriebssystem fortlaufend aktualisiert und in die entsprechenden Variablen- und Merkerbereiche übertragen.

## n Achse 1

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A1	Variable 101	✓
Achse führt Positionierung durch	M_MOVIN_A1	Merker 2435	✓
Regler ist eingeschaltet	M_RFGON_A1	Merker 2436	✓
Achse ist in Position	M_INPOS_A1	Merker 2437	✓
Fehler aufgetreten	M_ERROR_A1	Merker 2438	✓
Schleppabstandsfehler	M_SLPST_A1	Merker 2439	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A1	Merker 2440	✓
Fehler: Endschalter Plus	M_STOE_ESP_A1	Merker 2441	✓
Fehler: Endschalter Minus	M_STOE_ESM_A1	Merker 2442	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A1	Merker 2443	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A1	Merker 2444	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A1	Merker 2445	✓
Fehler: Kommandofehler	M_MELD_KOM_A1	Merker 2446	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A1	Merker 2447	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A1	Merker 2448	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A1	Merker 2449	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A1	Merker 2457	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A1	Merker 2458	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A1	Merker 2459	✓
Physikalischer Zustand Referenzschalter	M_REFS_A1	Merker 2460	✓
Zustand Endschalter Plus	M_E_ESP_A1	Merker 2461	✓
Zustand Endschalter Minus	M_E_ESM_A1	Merker 2462	✓
Zustand Leistungsteil bereit	M_E_ERB_A1	Merker 2463	✓
Zustand Referenzschalter	M_E_REF_A1	Merker 2464	✓

n Tabelle 53 – Systemmerker und –Variablen Achsenstatus Achse 1

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 100	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 101	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 102	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 103	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 104	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 105	✓

n Tabelle 54 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 1

## n Achse 2

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A2	Variable 102	✓
Achse führt Positionierung durch	M_MOVIN_A2	Merker 2467	✓
Regler ist eingeschaltet	M_RFGON_A2	Merker 2468	✓
Achse ist in Position	M_INPOS_A2	Merker 2469	✓
Fehler aufgetreten	M_ERROR_A2	Merker 2470	✓
Schleppabstandsfehler	M_SLPST_A2	Merker 2471	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A2	Merker 2472	✓
Fehler: Endschalter Plus	M_STOE_ESP_A2	Merker 2473	✓
Fehler: Endschalter Minus	M_STOE_ESM_A2	Merker 2474	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A2	Merker 2475	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A2	Merker 2476	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A2	Merker 2477	✓
Fehler: Kommandofehler	M_MELD_KOM_A2	Merker 2478	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A2	Merker 2479	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A2	Merker 2480	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A2	Merker 2481	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A2	Merker 2489	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A2	Merker 2490	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A2	Merker 2491	✓
Physikalischer Zustand Referenzschalter	M_REFS_A2	Merker 2492	✓
Zustand Endschalter Plus	M_E_ESP_A2	Merker 2493	✓
Zustand Endschalter Minus	M_E_ESM_A2	Merker 2494	✓
Zustand Leistungsteil bereit	M_E_ERB_A2	Merker 2495	✓
Zustand Referenzschalter	M_E_REF_A2	Merker 2496	✓

n Tabelle 55 – Systemmerker und –Variablen Achsenstatus Achse 2

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 200	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 201	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 202	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 203	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 204	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 205	✓

n Tabelle 56 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 2

## n Achse 3

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A3	Variable 103	✓
Achse führt Positionierung durch	M_MOVIN_A3	Merker 2499	✓
Regler ist eingeschaltet	M_RFGON_A3	Merker 2500	✓
Achse ist in Position	M_INPOS_A3	Merker 2501	✓
Fehler aufgetreten	M_ERROR_A3	Merker 2502	✓
Schleppabstandsfehler	M_SLPST_A3	Merker 2503	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A3	Merker 2504	✓
Fehler: Endschalter Plus	M_STOE_ESP_A3	Merker 2505	✓
Fehler: Endschalter Minus	M_STOE_ESM_A3	Merker 2506	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A3	Merker 2507	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A3	Merker 2508	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A3	Merker 2509	✓
Fehler: Kommandofehler	M_MELD_KOM_A3	Merker 2510	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A3	Merker 2511	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A3	Merker 2512	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A3	Merker 2513	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A3	Merker 2521	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A3	Merker 2522	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A3	Merker 2523	✓
Physikalischer Zustand Referenzschalter	M_REFS_A3	Merker 2524	✓
Zustand Endschalter Plus	M_E_ESP_A3	Merker 2525	✓
Zustand Endschalter Minus	M_E_ESM_A3	Merker 2526	✓
Zustand Leistungsteil bereit	M_E_ERB_A3	Merker 2527	✓
Zustand Referenzschalter	M_E_REF_A3	Merker 2528	✓

n Tabelle 57 – Systemmerker und –Variablen Achsenstatus Achse 3

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 300	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 301	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 302	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 303	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 304	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 305	✓

n Tabelle 58 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 3

## n Achse 4

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A4	Variable 104	ý
Achse führt Positionierung durch	M_MOVIN_A4	Merker 2531	ý
Regler ist eingeschaltet	M_RFGON_A4	Merker 2532	ý
Achse ist in Position	M_INPOS_A4	Merker 2533	ý
Fehler aufgetreten	M_ERROR_A4	Merker 2534	ý
Schleppabstandsfehler	M_SLPST_A4	Merker 2535	ý
Referenzfahrt wurde durchgeführt	M_REFOK_A4	Merker 2536	ý
Fehler: Endschalter Plus	M_STOE_ESP_A4	Merker 2537	ý
Fehler: Endschalter Minus	M_STOE_ESM_A4	Merker 2538	ý
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A4	Merker 2539	ý
Fehler: Schleppabstandsfehler	M_STOE_SPF_A4	Merker 2540	ý
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A4	Merker 2541	ý
Fehler: Kommandofehler	M_MELD_KOM_A4	Merker 2542	ý
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A4	Merker 2543	ý
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A4	Merker 2544	ý
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A4	Merker 2545	ý
Physikalischer Zustand Endschalter Plus	M_ENDP_A4	Merker 2553	ý
Physikalischer Zustand Endschalter Minus	M_ENDM_A4	Merker 2554	ý
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A4	Merker 2555	ý
Physikalischer Zustand Referenzschalter	M_REFS_A4	Merker 2556	ý
Zustand Endschalter Plus	M_E_ESP_A4	Merker 2557	ý
Zustand Endschalter Minus	M_E_ESM_A4	Merker 2558	ý
Zustand Leistungsteil bereit	M_E_ERB_A4	Merker 2559	ý
Zustand Referenzschalter	M_E_REF_A4	Merker 2560	ý

n Tabelle 59 – Systemmerker und –Variablen Achsenstatus Achse 4

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 400	ý
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 401	ý
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 402	ý
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 403	ý
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 404	ý
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 405	ý

n Tabelle 60 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 4

## n Achse 5

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A5	Variable 105	✓
Achse führt Positionierung durch	M_MOVIN_A5	Merker 2563	✓
Regler ist eingeschaltet	M_RFGON_A5	Merker 2564	✓
Achse ist in Position	M_INPOS_A5	Merker 2565	✓
Fehler aufgetreten	M_ERROR_A5	Merker 2566	✓
Schleppabstandsfehler	M_SLPST_A5	Merker 2567	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A5	Merker 2568	✓
Fehler: Endschalter Plus	M_STOE_ESP_A5	Merker 2569	✓
Fehler: Endschalter Minus	M_STOE_ESM_A5	Merker 2570	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTБ_A5	Merker 2571	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A5	Merker 2572	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A5	Merker 2573	✓
Fehler: Kommandofehler	M_MELD_KOM_A5	Merker 2574	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A5	Merker 2575	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A5	Merker 2576	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A5	Merker 2577	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A5	Merker 2585	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A5	Merker 2586	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A5	Merker 2587	✓
Physikalischer Zustand Referenzschalter	M_REFS_A5	Merker 2588	✓
Zustand Endschalter Plus	M_E_ESP_A5	Merker 2589	✓
Zustand Endschalter Minus	M_E_ESM_A5	Merker 2590	✓
Zustand Leistungsteil bereit	M_E_ERB_A5	Merker 2591	✓
Zustand Referenzschalter	M_E_REF_A5	Merker 2592	✓

n Tabelle 61 – Systemmerker und –Variablen Achsenstatus Achse 5

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 500	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 501	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 502	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 503	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 504	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 505	✓

n Tabelle 62 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 5

## n Achse 6

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A6	Variable 106	✓
Achse führt Positionierung durch	M_MOVIN_A6	Merker 2595	✓
Regler ist eingeschaltet	M_RFGON_A6	Merker 2596	✓
Achse ist in Position	M_INPOS_A6	Merker 2597	✓
Fehler aufgetreten	M_ERROR_A6	Merker 2598	✓
Schleppabstandsfehler	M_SLPST_A6	Merker 2599	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A6	Merker 2600	✓
Fehler: Endschalter Plus	M_STOE_ESP_A6	Merker 2601	✓
Fehler: Endschalter Minus	M_STOE_ESM_A6	Merker 2602	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTBA6	Merker 2603	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A6	Merker 2604	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A6	Merker 2605	✓
Fehler: Kommandofehler	M_MELD_KOM_A6	Merker 2606	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A6	Merker 2607	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A6	Merker 2608	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A6	Merker 2609	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A6	Merker 2617	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A6	Merker 2618	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A6	Merker 2619	✓
Physikalischer Zustand Referenzschalter	M_REFS_A6	Merker 2620	✓
Zustand Endschalter Plus	M_E_ESP_A6	Merker 2621	✓
Zustand Endschalter Minus	M_E_ESM_A6	Merker 2622	✓
Zustand Leistungsteil bereit	M_E_ERB_A6	Merker 2623	✓
Zustand Referenzschalter	M_E_REF_A6	Merker 2624	✓

n Tabelle 63 – Systemmerker und –Variablen Achsenstatus Achse 6

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 600	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 601	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 602	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 603	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 604	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 605	✓

n Tabelle 64 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 6



## n Achse 7

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A7	Variable 107	✓
Achse führt Positionierung durch	M_MOVIN_A7	Merker 2627	✓
Regler ist eingeschaltet	M_RFGON_A7	Merker 2628	✓
Achse ist in Position	M_INPOS_A7	Merker 2629	✓
Fehler aufgetreten	M_ERROR_A7	Merker 2630	✓
Schleppabstandsfehler	M_SLPST_A7	Merker 2631	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A7	Merker 2632	✓
Fehler: Endschalter Plus	M_STOE_ESP_A7	Merker 2633	✓
Fehler: Endschalter Minus	M_STOE_ESM_A7	Merker 2634	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTБ_A7	Merker 2635	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A7	Merker 2636	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A7	Merker 2637	✓
Fehler: Kommandofehler	M_MELD_KOM_A7	Merker 2638	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A7	Merker 2639	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A7	Merker 2640	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A7	Merker 2641	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A7	Merker 2649	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A7	Merker 2650	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A7	Merker 2651	✓
Physikalischer Zustand Referenzschalter	M_REFS_A7	Merker 2652	✓
Zustand Endschalter Plus	M_E_ESP_A7	Merker 2653	✓
Zustand Endschalter Minus	M_E_ESM_A7	Merker 2654	✓
Zustand Leistungsteil bereit	M_E_ERB_A7	Merker 2655	✓
Zustand Referenzschalter	M_E_REF_A7	Merker 2656	✓

n Tabelle 65 – Systemmerker und –Variablen Achsenstatus Achse 7

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 700	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 701	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 702	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 703	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 704	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 705	✓

n Tabelle 66 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 7

## n Achse 8

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A8	Variable 108	✓
Achse führt Positionierung durch	M_MOVIN_A8	Merker 2659	✓
Regler ist eingeschaltet	M_RFGON_A8	Merker 2660	✓
Achse ist in Position	M_INPOS_A8	Merker 2661	✓
Fehler aufgetreten	M_ERROR_A8	Merker 2662	✓
Schleppabstandsfehler	M_SLPST_A8	Merker 2663	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A8	Merker 2664	✓
Fehler: Endschalter Plus	M_STOE_ESP_A8	Merker 2665	✓
Fehler: Endschalter Minus	M_STOE_ESM_A8	Merker 2666	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A8	Merker 2667	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A8	Merker 2668	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A8	Merker 2669	✓
Fehler: Kommandofehler	M_MELD_KOM_A8	Merker 2670	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A8	Merker 2671	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A8	Merker 2672	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A8	Merker 2673	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A8	Merker 2681	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A8	Merker 2682	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A8	Merker 2683	✓
Physikalischer Zustand Referenzschalter	M_REFS_A8	Merker 2684	✓
Zustand Endschalter Plus	M_E_ESP_A8	Merker 2685	✓
Zustand Endschalter Minus	M_E_ESM_A8	Merker 2686	✓
Zustand Leistungsteil bereit	M_E_ERB_A8	Merker 2687	✓
Zustand Referenzschalter	M_E_REF_A8	Merker 2688	✓

n Tabelle 67 – Systemmerker und –Variablen Achsenstatus Achse 8

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 800	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 801	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 802	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 803	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 804	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 805	✓

n Tabelle 68 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 8

## n Achse 9

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A9	Variable 109	✓
Achse führt Positionierung durch	M_MOVIN_A9	Merker 2691	✓
Regler ist eingeschaltet	M_RFGON_A9	Merker 2692	✓
Achse ist in Position	M_INPOS_A9	Merker 2693	✓
Fehler aufgetreten	M_ERROR_A9	Merker 2694	✓
Schleppabstandsfehler	M_SLPST_A9	Merker 2695	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A9	Merker 2696	✓
Fehler: Endschalter Plus	M_STOE_ESP_A9	Merker 2697	✓
Fehler: Endschalter Minus	M_STOE_ESM_A9	Merker 2698	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTБ_A9	Merker 2699	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A9	Merker 2700	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A9	Merker 2701	✓
Fehler: Kommandofehler	M_MELD_KOM_A9	Merker 2702	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A9	Merker 2703	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A9	Merker 2704	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A9	Merker 2705	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A9	Merker 2713	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A9	Merker 2714	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A9	Merker 2715	✓
Physikalischer Zustand Referenzschalter	M_REFS_A9	Merker 2716	✓
Zustand Endschalter Plus	M_E_ESP_A9	Merker 2717	✓
Zustand Endschalter Minus	M_E_ESM_A9	Merker 2718	✓
Zustand Leistungsteil bereit	M_E_ERB_A9	Merker 2719	✓
Zustand Referenzschalter	M_E_REF_A9	Merker 2720	✓

n Tabelle 69 – Systemmerker und –Variablen Achsenstatus Achse 9

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 900	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 901	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 902	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 903	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 904	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 905	✓

n Tabelle 70 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 9

## n Achse 10

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A10	Variable 110	✓
Achse führt Positionierung durch	M_MOVIN_A10	Merker 2723	✓
Regler ist eingeschaltet	M_RFGON_A10	Merker 2724	✓
Achse ist in Position	M_INPOS_A10	Merker 2725	✓
Fehler aufgetreten	M_ERROR_A10	Merker 2726	✓
Schleppabstandsfehler	M_SLPST_A10	Merker 2727	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A10	Merker 2728	✓
Fehler: Endschalter Plus	M_STOE_ESP_A10	Merker 2729	✓
Fehler: Endschalter Minus	M_STOE_ESM_A10	Merker 2730	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTБ_A10	Merker 2731	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A10	Merker 2732	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A10	Merker 2733	✓
Fehler: Kommandofehler	M_MELD_KOM_A10	Merker 2734	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A10	Merker 2735	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A10	Merker 2736	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A10	Merker 2737	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A10	Merker 2745	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A10	Merker 2746	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A10	Merker 2747	✓
Physikalischer Zustand Referenzschalter	M_REFS_A10	Merker 2748	✓
Zustand Endschalter Plus	M_E_ESP_A10	Merker 2749	✓
Zustand Endschalter Minus	M_E_ESM_A10	Merker 2750	✓
Zustand Leistungsteil bereit	M_E_ERB_A10	Merker 2751	✓
Zustand Referenzschalter	M_E_REF_A10	Merker 2752	✓

n Tabelle 71 – Systemmerker und –Variablen Achsenstatus Achse 10

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 1000	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 1001	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 1002	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 1003	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 1004	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 1005	✓

n Tabelle 72 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 10

## n Achse 11

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A11	Variable 111	✓
Achse führt Positionierung durch	M_MOVIN_A11	Merker 2755	✓
Regler ist eingeschaltet	M_RFGON_A11	Merker 2756	✓
Achse ist in Position	M_INPOS_A11	Merker 2757	✓
Fehler aufgetreten	M_ERROR_A11	Merker 2758	✓
Schleppabstandsfehler	M_SLPST_A11	Merker 2759	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A11	Merker 2760	✓
Fehler: Endschalter Plus	M_STOE_ESP_A11	Merker 2761	✓
Fehler: Endschalter Minus	M_STOE_ESM_A11	Merker 2762	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A11	Merker 2763	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A11	Merker 2764	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A11	Merker 2765	✓
Fehler: Kommandofehler	M_MELD_KOM_A11	Merker 2766	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A11	Merker 2767	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A11	Merker 2768	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A11	Merker 2769	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A11	Merker 2777	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A11	Merker 2778	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A11	Merker 2779	✓
Physikalischer Zustand Referenzschalter	M_REFS_A11	Merker 2780	✓
Zustand Endschalter Plus	M_E_ESP_A11	Merker 2781	✓
Zustand Endschalter Minus	M_E_ESM_A11	Merker 2782	✓
Zustand Leistungsteil bereit	M_E_ERB_A11	Merker 2783	✓
Zustand Referenzschalter	M_E_REF_A11	Merker 2784	✓

n Tabelle 73 – Systemmerker und –Variablen Achsenstatus Achse 11

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 1100	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 1101	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 1102	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 1103	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 1104	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 1105	✓

n Tabelle 74 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 11

## n Achse 12

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A12	Variable 112	✓
Achse führt Positionierung durch	M_MOVIN_A12	Merker 2787	✓
Regler ist eingeschaltet	M_RFGON_A12	Merker 2788	✓
Achse ist in Position	M_INPOS_A12	Merker 2789	✓
Fehler aufgetreten	M_ERROR_A12	Merker 2790	✓
Schleppabstandsfehler	M_SLPST_A12	Merker 2791	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A12	Merker 2792	✓
Fehler: Endschalter Plus	M_STOE_ESP_A12	Merker 2793	✓
Fehler: Endschalter Minus	M_STOE_ESM_A12	Merker 2794	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTБ_A12	Merker 2795	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A12	Merker 2796	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A12	Merker 2797	✓
Fehler: Kommandofehler	M_MELD_KOM_A12	Merker 2798	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A12	Merker 2799	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A12	Merker 2800	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A12	Merker 2801	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A12	Merker 2809	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A12	Merker 2810	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A12	Merker 2811	✓
Physikalischer Zustand Referenzschalter	M_REFS_A12	Merker 2812	✓
Zustand Endschalter Plus	M_E_ESP_A12	Merker 2813	✓
Zustand Endschalter Minus	M_E_ESM_A12	Merker 2814	✓
Zustand Leistungsteil bereit	M_E_ERB_A12	Merker 2815	✓
Zustand Referenzschalter	M_E_REF_A12	Merker 2816	✓

n Tabelle 75 – Systemmerker und –Variablen Achsenstatus Achse 12

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 1200	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 1201	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 1202	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 1203	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 1204	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 1205	✓

n Tabelle 76 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 12

## n Achse 13

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A13	Variable 113	✓
Achse führt Positionierung durch	M_MOVIN_A13	Merker 2819	✓
Regler ist eingeschaltet	M_RFGON_A13	Merker 2820	✓
Achse ist in Position	M_INPOS_A13	Merker 2821	✓
Fehler aufgetreten	M_ERROR_A13	Merker 2822	✓
Schleppabstandsfehler	M_SLPST_A13	Merker 2823	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A13	Merker 2824	✓
Fehler: Endschalter Plus	M_STOE_ESP_A13	Merker 2825	✓
Fehler: Endschalter Minus	M_STOE_ESM_A13	Merker 2826	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A13	Merker 2827	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A13	Merker 2828	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A13	Merker 2829	✓
Fehler: Kommandofehler	M_MELD_KOM_A13	Merker 2830	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A13	Merker 2831	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A13	Merker 2832	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A13	Merker 2833	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A13	Merker 2841	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A13	Merker 2842	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A13	Merker 2843	✓
Physikalischer Zustand Referenzschalter	M_REFS_A13	Merker 2844	✓
Zustand Endschalter Plus	M_E_ESP_A13	Merker 2845	✓
Zustand Endschalter Minus	M_E_ESM_A13	Merker 2846	✓
Zustand Leistungsteil bereit	M_E_ERB_A13	Merker 2847	✓
Zustand Referenzschalter	M_E_REF_A13	Merker 2848	✓

n Tabelle 77 – Systemmerker und –Variablen Achsenstatus Achse 13

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 1300	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 1301	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 1302	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 1303	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 1304	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 1305	✓

n Tabelle 78 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 13

## n Achse 14

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A14	Variable 114	✓
Achse führt Positionierung durch	M_MOVIN_A14	Merker 2851	✓
Regler ist eingeschaltet	M_RFGON_A14	Merker 2852	✓
Achse ist in Position	M_INPOS_A14	Merker 2853	✓
Fehler aufgetreten	M_ERROR_A14	Merker 2854	✓
Schleppabstandsfehler	M_SLPST_A14	Merker 2855	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A14	Merker 2856	✓
Fehler: Endschalter Plus	M_STOE_ESP_A14	Merker 2857	✓
Fehler: Endschalter Minus	M_STOE_ESM_A14	Merker 2858	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A14	Merker 2859	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A14	Merker 2860	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A14	Merker 2861	✓
Fehler: Kommandofehler	M_MELD_KOM_A14	Merker 2862	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A14	Merker 2863	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A14	Merker 2864	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A14	Merker 2865	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A14	Merker 2873	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A14	Merker 2874	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A14	Merker 2875	✓
Physikalischer Zustand Referenzschalter	M_REFS_A14	Merker 2876	✓
Zustand Endschalter Plus	M_E_ESP_A14	Merker 2877	✓
Zustand Endschalter Minus	M_E_ESM_A14	Merker 2878	✓
Zustand Leistungsteil bereit	M_E_ERB_A14	Merker 2879	✓
Zustand Referenzschalter	M_E_REF_A14	Merker 2880	✓

n Tabelle 79 Systemmerker und –Variablen Achsenstatus Achse 14

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 1400	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 1401	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 1402	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 1403	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 1404	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 1405	✓

n Tabelle 80 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 14



## n Achse 15

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A15	Variable 115	✓
Achse führt Positionierung durch	M_MOVIN_A15	Merker 2883	✓
Regler ist eingeschaltet	M_RFGON_A15	Merker 2884	✓
Achse ist in Position	M_INPOS_A15	Merker 2885	✓
Fehler aufgetreten	M_ERROR_A15	Merker 2886	✓
Schleppabstandsfehler	M_SLPST_A15	Merker 2887	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A15	Merker 2888	✓
Fehler: Endschalter Plus	M_STOE_ESP_A15	Merker 2889	✓
Fehler: Endschalter Minus	M_STOE_ESM_A15	Merker 2890	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A15	Merker 2891	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A15	Merker 2892	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A15	Merker 2893	✓
Fehler: Kommandofehler	M_MELD_KOM_A15	Merker 2894	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A15	Merker 2895	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A15	Merker 2896	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A15	Merker 2897	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A15	Merker 2905	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A15	Merker 2906	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A15	Merker 2907	✓
Physikalischer Zustand Referenzschalter	M_REFS_A15	Merker 2908	✓
Zustand Endschalter Plus	M_E_ESP_A15	Merker 2909	✓
Zustand Endschalter Minus	M_E_ESM_A15	Merker 2910	✓
Zustand Leistungsteil bereit	M_E_ERB_A15	Merker 2911	✓
Zustand Referenzschalter	M_E_REF_A15	Merker 2912	✓

n Tabelle 81 – Systemmerker und –Variablen Achsenstatus Achse 15

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 1500	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 1501	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 1502	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 1503	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 1504	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 1505	✓

n Tabelle 82 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 15

## n Achse 16

Funktion	Name	Adresse	
Aktuelle Ist-Position	V_ISTPOS_A16	Variable 116	✓
Achse führt Positionierung durch	M_MOVIN_A16	Merker 2915	✓
Regler ist eingeschaltet	M_RFGON_A16	Merker 2916	✓
Achse ist in Position	M_INPOS_A16	Merker 2917	✓
Fehler aufgetreten	M_ERROR_A16	Merker 2918	✓
Schleppabstandsfehler	M_SLPST_A16	Merker 2919	✓
Referenzfahrt wurde durchgeführt	M_REFOK_A16	Merker 2920	✓
Fehler: Endschalter Plus	M_STOE_ESP_A16	Merker 2921	✓
Fehler: Endschalter Minus	M_STOE_ESM_A16	Merker 2922	✓
Fehler: Störung Leistungsteil bereit	M_STOE_BTB_A16	Merker 2923	✓
Fehler: Schleppabstandsfehler	M_STOE_SPF_A16	Merker 2924	✓
Fehler: Erweiterter Schleppabstandsfehler	M_STOE_SPE_A16	Merker 2925	✓
Fehler: Kommandofehler	M_MELD_KOM_A16	Merker 2926	✓
Fehler: Softlimit Plus überschritten	M_MELD_SLP_A16	Merker 2927	✓
Fehler: Softlimit Minus überschritten	M_MELD_SLM_A16	Merker 2928	✓
Fehler: Störung auf Interpolations-Bus	M_STOE_INT_A16	Merker 2929	✓
Physikalischer Zustand Endschalter Plus	M_ENDP_A16	Merker 2937	✓
Physikalischer Zustand Endschalter Minus	M_ENDM_A16	Merker 2938	✓
Physikalischer Zustand Leistungsteil bereit	M_RGLB_A16	Merker 2939	✓
Physikalischer Zustand Referenzschalter	M_REFS_A16	Merker 2940	✓
Zustand Endschalter Plus	M_E_ESP_A16	Merker 2941	✓
Zustand Endschalter Minus	M_E_ESM_A16	Merker 2942	✓
Zustand Leistungsteil bereit	M_E_ERB_A16	Merker 2943	✓
Zustand Referenzschalter	M_E_REF_A16	Merker 2944	✓

n Tabelle 83 – Systemmerker und –Variablen Achsenstatus Achse 16

Funktion	Bezeichnung	Adresse	
Aktueller Schleppfehler (nicht über PC lesbar!)	Aktueller Schleppfehler	Parameter 1600	✓
Aktuelle Ist-Geschwindigkeit	Aktuelle Geschwindigkeit	Parameter 1601	✓
Bitcodierter Fehlerstatus	Aktueller Fehler	Parameter 1602	✓
Bitcodierter Status digitale Eingänge	Aktueller Eingangsstatus	Parameter 1603	✓
Aktuelle Ist-Position	Aktuelle Ist-Position	Parameter 1604	✓
Letzter Messwert (vgl. Kapitel 4.1 - Messfunktionen ab Seite 156)	Letzter Meßwert	Parameter 1505	✓

n Tabelle 84 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 16

## 6.8 Meßfunktionsdaten

Für die im System integrierten Meßfunktionen – beschrieben im Kapitel 4.1 - Messfunktionen (Seite 156) – werden folgende Merker zusätzlich zur Verfügung gestellt:

Funktion	Name	Adresse	
Meßfunktion ein-/ ausschalten	M_DIAGS	Merker 2096	Ⓟ
Neue Meßdaten für Achse 1 verfügbar	M_SPEC_DAT_A1	Merker 2113	Ⓟ
Neue Meßdaten für Achse 2 verfügbar	M_SPEC_DAT_A2	Merker 2114	Ⓟ
Neue Meßdaten für Achse 3 verfügbar	M_SPEC_DAT_A3	Merker 2115	Ⓟ
Neue Meßdaten für Achse 4 verfügbar	M_SPEC_DAT_A4	Merker 2116	Ⓟ
Neue Meßdaten für Achse 5 verfügbar	M_SPEC_DAT_A5	Merker 2117	Ⓟ
Neue Meßdaten für Achse 6 verfügbar	M_SPEC_DAT_A6	Merker 2118	Ⓟ
Neue Meßdaten für Achse 7 verfügbar	M_SPEC_DAT_A7	Merker 2119	Ⓟ
Neue Meßdaten für Achse 8 verfügbar	M_SPEC_DAT_A8	Merker 2120	Ⓟ
Neue Meßdaten für Achse 9 verfügbar	M_SPEC_DAT_A9	Merker 2121	Ⓟ
Neue Meßdaten für Achse 10 verfügbar	M_SPEC_DAT_A10	Merker 2122	Ⓟ
Neue Meßdaten für Achse 11 verfügbar	M_SPEC_DAT_A11	Merker 2123	Ⓟ
Neue Meßdaten für Achse 12 verfügbar	M_SPEC_DAT_A12	Merker 2124	Ⓟ
Neue Meßdaten für Achse 13 verfügbar	M_SPEC_DAT_A13	Merker 2125	Ⓟ
Neue Meßdaten für Achse 14 verfügbar	M_SPEC_DAT_A14	Merker 2126	Ⓟ
Neue Meßdaten für Achse 15 verfügbar	M_SPEC_DAT_A15	Merker 2127	Ⓟ
Neue Meßdaten für Achse 16 verfügbar	M_SPEC_DAT_A16	Merker 2128	Ⓟ

Ⓟ Tabelle 85 – Systemmerker und -Variablen für die Meßfunktionen

## 6.9 Systemtimer

Das MC200 System verfügt über 32 in das Betriebssystem fest integrierte Systemtimer. Sie können diese Timer in Ihrem SPS-Programm frei verwenden. Bitte beachten Sie, daß die Systemtimer unterschiedliche Auflösungen haben:

- n Timer 1 bis 6 sowie 9 bis 32 arbeiten mit einer Genauigkeit von 1/10 s.
- n Timer 7 und 8 arbeiten mit einer Genauigkeit von 1/100 s.

Wir empfehlen, daß Sie, wenn Sie die hohe Auflösung nicht benötigen, stets die niedriger auflösenden Systemtimer verwenden. Eine Einführung in die Timerprogrammierung finden Sie in Kapitel 4.4 - Timer (Seite 166).

Funktion	Name	Adresse	
Timerwert 1 (1-65000 in 0,1 s)	V_TIM_1	Variable 11	Ⓟ
Start-/Stopmerker für Timer 1	M_TIM_1	Merker 2241	Ⓟ
Timerwert 2 (1-65000 in 0,1 s)	V_TIM_2	Variable 12	Ⓟ
Start-/Stopmerker für Timer 2	M_TIM_2	Merker 2242	Ⓟ
Timerwert 3 (1-65000 in 0,1 s)	V_TIM_3	Variable 13	Ⓟ
Start-/Stopmerker für Timer 3	M_TIM_3	Merker 2243	Ⓟ
Timerwert 4 (1-65000 in 0,1 s)	V_TIM_4	Variable 14	Ⓟ
Start-/Stopmerker für Timer 4	M_TIM_4	Merker 2244	Ⓟ
Timerwert 5 (1-65000 in 0,1 s)	V_TIM_5	Variable 15	Ⓟ
Start-/Stopmerker für Timer 5	M_TIM_5	Merker 2245	Ⓟ
Timerwert 6 (1-65000 in 0,1 s)	V_TIM_6	Variable 16	Ⓟ
Start-/Stopmerker für Timer 6	M_TIM_6	Merker 2246	Ⓟ
Timerwert 7 (1-65000 in 0,01 s)	V_TIM_7	Variable 17	Ⓟ
Start-/Stopmerker für Timer 7	M_TIM_7	Merker 2247	Ⓟ
Timerwert 8 (1-65000 in 0,01 s)	V_TIM_8	Variable 18	Ⓟ
Start-/Stopmerker für Timer 8	M_TIM_8	Merker 2248	Ⓟ
Timerwert 9 (1-65000 in 0,1 s)	V_TIM_9	Variable 19	Ⓟ
Start-/Stopmerker für Timer 9	M_TIM_9	Merker 2249	Ⓟ
Timerwert 10 (1-65000 in 0,1 s)	V_TIM_10	Variable 20	Ⓟ
Start-/Stopmerker für Timer 10	M_TIM_10	Merker 2250	Ⓟ
Timerwert 11 (1-65000 in 0,1 s)	V_TIM_11	Variable 21	Ⓟ
Start-/Stopmerker für Timer 11	M_TIM_11	Merker 2251	Ⓟ
Timerwert 12 (1-65000 in 0,1 s)	V_TIM_12	Variable 22	Ⓟ
Start-/Stopmerker für Timer 12	M_TIM_12	Merker 2252	Ⓟ
Timerwert 13 (1-65000 in 0,1 s)	V_TIM_13	Variable 23	Ⓟ
Start-/Stopmerker für Timer 13	M_TIM_13	Merker 2253	Ⓟ
Timerwert 14 (1-65000 in 0,1 s)	V_TIM_14	Variable 24	Ⓟ
Start-/Stopmerker für Timer 14	M_TIM_14	Merker 2254	Ⓟ

Funktion	Name	Adresse	
Timerwert 15 (1-65000 in 0,1 s)	V_TIM_15	Variable 25	Ⓟ
Start-/Stopmerker für Timer 15	M_TIM_15	Merker 2255	Ⓟ
Timerwert 16 (1-65000 in 0,1 s)	V_TIM_16	Variable 26	Ⓟ
Start-/Stopmerker für Timer 16	M_TIM_16	Merker 2256	Ⓟ
Timerwert 17 (1-65000 in 0,1 s)	V_TIM_17	Variable 27	Ⓟ
Start-/Stopmerker für Timer 17	M_TIM_17	Merker 2257	Ⓟ
Timerwert 18 (1-65000 in 0,1 s)	V_TIM_18	Variable 28	Ⓟ
Start-/Stopmerker für Timer 18	M_TIM_18	Merker 2258	Ⓟ
Timerwert 19 (1-65000 in 0,1 s)	V_TIM_19	Variable 29	Ⓟ
Start-/Stopmerker für Timer 19	M_TIM_19	Merker 2259	Ⓟ
Timerwert 20 (1-65000 in 0,1 s)	V_TIM_20	Variable 30	Ⓟ
Start-/Stopmerker für Timer 20	M_TIM_20	Merker 2260	Ⓟ
Timerwert 21 (1-65000 in 0,1 s)	V_TIM_21	Variable 31	Ⓟ
Start-/Stopmerker für Timer 21	M_TIM_21	Merker 2261	Ⓟ
Timerwert 22 (1-65000 in 0,1 s)	V_TIM_22	Variable 32	Ⓟ
Start-/Stopmerker für Timer 22	M_TIM_22	Merker 2262	Ⓟ
Timerwert 23 (1-65000 in 0,1 s)	V_TIM_23	Variable 33	Ⓟ
Start-/Stopmerker für Timer 23	M_TIM_23	Merker 2263	Ⓟ
Timerwert 24 (1-65000 in 0,1 s)	V_TIM_24	Variable 34	Ⓟ
Start-/Stopmerker für Timer 24	M_TIM_24	Merker 2264	Ⓟ
Timerwert 25 (1-65000 in 0,1 s)	V_TIM_25	Variable 35	Ⓟ
Start-/Stopmerker für Timer 25	M_TIM_25	Merker 2265	Ⓟ
Timerwert 26 (1-65000 in 0,1 s)	V_TIM_26	Variable 36	Ⓟ
Start-/Stopmerker für Timer 26	M_TIM_26	Merker 2266	Ⓟ
Timerwert 27 (1-65000 in 0,1 s)	V_TIM_27	Variable 37	Ⓟ
Start-/Stopmerker für Timer 27	M_TIM_27	Merker 2267	Ⓟ
Timerwert 28 (1-65000 in 0,1 s)	V_TIM_28	Variable 38	Ⓟ
Start-/Stopmerker für Timer 28	M_TIM_28	Merker 2268	Ⓟ
Timerwert 29 (1-65000 in 0,1 s)	V_TIM_29	Variable 39	Ⓟ
Start-/Stopmerker für Timer 29	M_TIM_29	Merker 2269	Ⓟ
Timerwert 30 (1-65000 in 0,1 s)	V_TIM_30	Variable 40	Ⓟ
Start-/Stopmerker für Timer 30	M_TIM_30	Merker 2270	Ⓟ
Timerwert 31 (1-65000 in 0,1 s)	V_TIM_31	Variable 41	Ⓟ
Start-/Stopmerker für Timer 31	M_TIM_31	Merker 2271	Ⓟ
Timerwert 32 (1-65000 in 0,1 s)	V_TIM_32	Variable 42	Ⓟ
Start-/Stopmerker für Timer 32	M_TIM_32	Merker 2272	Ⓟ

n Tabelle 86 – Systemmerker und –Variablen für die SPS-Timer

## 6.10 Blinkmerker

Das MC200 System verfügt über 8 Blinkmerker. Sie können die Blinkmerker in Ihrem SPS-Programm frei verwenden. Bitte beachten Sie, daß die Blinkmerker unterschiedliche Zeiten haben.

Funktion	Name	Adresse	
Blinkmerker (blinkt im 0,01s Takt)	M_BLINK1	Merker 2225	Ÿ
Blinkmerker (blinkt im 0,02s Takt)	M_BLINK2	Merker 2226	Ÿ
Blinkmerker (blinkt im 0,04s Takt)	M_BLINK4	Merker 2227	Ÿ
Blinkmerker (blinkt im 0,08s Takt)	M_BLINK8	Merker 2228	Ÿ
Blinkmerker (blinkt im 0,16s Takt)	M_BLINK16	Merker 2229	Ÿ
Blinkmerker (blinkt im 0,32s Takt)	M_BLINK32	Merker 2230	Ÿ
Blinkmerker (blinkt im 0,64s Takt)	M_BLINK64	Merker 2231	Ÿ
Blinkmerker (blinkt im 1,28s Takt)	M_BLINK128	Merker 2232	Ÿ

n Tabelle 87 – Blinkmerker

Verwenden Sie die Blinkmerker, wenn Sie in bestimmten Zeiten synchronisieren möchten. Sie können natürlich stattdessen auch Systemtimer verwenden. Da aber nur eine begrenzte Anzahl von Timern zur Verfügung stehen, und außerdem die Systemtimer CPU-Rechenleistung in Anspruch nehmen, ist es sinnvoll Blinkmerker zu verwenden.

## 6.11 Zählermodul

Im MC200 System ist auch ein schnelles Zählermodul erhältlich. Wenn entsprechende Zählermodule an Ihrem System angeschlossen sind, erhalten Sie über die folgenden Systemvariablen Zugriff auf den jeweils aktuellen Zählerwert:

Funktion	Name	Adresse	
Aktueller Wert des Zählermodul 1	V_CNT1	Variable 61	ý
Zählrichtung des Zählermoduls CNT1 Aus = normal, Ein = invertiert	M_CNTDIR_1	Merker 2193	þ
Aktueller Wert des Zählermodul 2	V_CNT2	Variable 62	ý
Zählrichtung des Zählermoduls CNT2 Aus = normal, Ein = invertiert	M_CNTDIR_2	Merker 2194	þ
Aktueller Wert des Zählermodul 3	V_CNT3	Variable 63	ý
Zählrichtung des Zählermoduls CNT3 Aus = normal, Ein = invertiert	M_CNTDIR_3	Merker 2195	þ
Aktueller Wert des Zählermodul 4	V_CNT4	Variable 64	ý
Zählrichtung des Zählermoduls CNT4 Aus = normal, Ein = invertiert	M_CNTDIR_4	Merker 2196	þ

n Tabelle 88 – Systemmerker und –Variablen zum Zählermodul

## 6.12 Analoge Ein- und Ausgänge

Im MC200 System sind auch analoge Ein-/Ausgangsmodule erhältlich, die Auflösung beträgt 12Bit. Wenn entsprechende Module an Ihrem System angeschlossen sind, erhalten Sie über die folgenden Systemvariablen Zugriff auf die aktuellen Zustände der analogen Ein-/Ausgangsmodule:

Funktion	Name	Adresse	
Neue Daten für Analog Modul 1 verfügbar	M_ANAINP1	Merker 2209	Ⓟ
Neue Daten für Analog Modul 2 verfügbar	M_ANAINP2	Merker 2210	Ⓟ
Neue Daten für Analog Modul 3 verfügbar	M_ANAINP3	Merker 2211	Ⓟ
Neue Daten für Analog Modul 4 verfügbar	M_ANAINP4	Merker 2212	Ⓟ
Analog Modul 1, Eingang 1 (0-4095)	V_ANAINP1_1	Variable 71	Ÿ
Analog Modul 1, Eingang 2 (0-4095)	V_ANAINP1_2	Variable 72	Ÿ
Analog Modul 1, Eingang 3 (0-4095)	V_ANAINP1_3	Variable 73	Ÿ
Analog Modul 1, Eingang 4 (0-4095)	V_ANAINP1_4	Variable 74	Ÿ
Analog Modul 2, Eingang 1 (0-4095)	V_ANAINP2_1	Variable 75	Ÿ
Analog Modul 2, Eingang 2 (0-4095)	V_ANAINP2_2	Variable 76	Ÿ
Analog Modul 2, Eingang 3 (0-4095)	V_ANAINP2_3	Variable 77	Ÿ
Analog Modul 2, Eingang 4 (0-4095)	V_ANAINP2_4	Variable 78	Ÿ
Analog Modul 3, Eingang 1 (0-4095)	V_ANAINP3_1	Variable 79	Ÿ
Analog Modul 3, Eingang 2 (0-4095)	V_ANAINP3_2	Variable 80	Ÿ
Analog Modul 3, Eingang 3 (0-4095)	V_ANAINP3_3	Variable 81	Ÿ
Analog Modul 3, Eingang 4 (0-4095)	V_ANAINP3_4	Variable 82	Ÿ
Analog Modul 4, Eingang 1 (0-4095)	V_ANAINP4_1	Variable 83	Ÿ
Analog Modul 4, Eingang 2 (0-4095)	V_ANAINP4_2	Variable 84	Ÿ
Analog Modul 4, Eingang 3 (0-4095)	V_ANAINP4_3	Variable 85	Ÿ
Analog Modul 4, Eingang 4 (0-4095)	V_ANAINP4_4	Variable 86	Ÿ

▮ Tabelle 89 – Systemmerker und –Variablen zu den analogen Ein-/Ausgängen

Bitte beachten Sie auch die Einführung in die Analog I/O-Programmierung in Kapitel 4.3 - Analog Ein-/Ausgänge (ab Seite 164) sowie die Befehlsreferenz Analog I/O (Seite 35).



## 6.13 EAU-T Emulation

Wie in Kapitel 4.6 - EAU-T Emulation (Seite 174) beschrieben, verfügt das CPU-Modul der MC200 Familie über eine integrierte Intelligenz, um Flankenwechsel eines Eingangs zeitverzögert mit Flankenwechseln eines Ausgangs zu kombinieren. Dies ist zwar kein "echtes" Erweiterungsmodul, emuliert aber eine Hardware, die in gleicher Funktion z.B. für die MC100 Familie verfügbar ist. Hierzu werden folgende Variablen und Merker verwendet:

Funktion	Name	Adresse	
Aktivierung der EAU-T Emulation. Sobald der Merker eingeschaltet wird, läuft die Funktion. Wurde die gewünschte Aktion ausgeführt, wird der Merker abgeschaltet.	M_EAUT	Merker 2273	b
Zeitverzögerung in ms. Gibt an, wie lange das System nach einem Flankenwechsel des Eingangs wartet, bis der Ausgang umgeschaltet wird.	V_EAUT_TIM	Variable 47	b
Parameterisierung Eingang und Ausgang: die unteren 16 Bit der Variable enthalten die Eingangsnummer, die oberen 16 Bit enthalten die Ausgangsnummer	V_EAUT_MASK	Variable 48	b

n Tabelle 90 – Systemmerker und –Variablen zur EAU-T Emulation

## 6.14 Displayprogrammierung

Im MC200 System sind auch Display, Bedienteil und Handterminal erhältlich. Wenn entsprechende Module an Ihrem System angeschlossen sind, erhalten Sie über die folgenden Systemvariablen Zugriff auf die aktuellen Zustände des Display, Bedienteil und Handterminal:

Funktion	Name	Adresse	
Textnummer, Zahl oder ASCII-Code für Anzeige des Display Nr.1	V_ANZNR1	Variable 90	Ⓟ
Format für Display Nr.1	V_ANZMSK1	Variable 91	Ⓟ
Editorzahl für Display Nr.1	V_INPDAT1	Variable 92	Ⓟ
Format für Display Nr.1	V_INPMSK1	Variable 93	Ⓟ
Textnummer, Zahl oder ASCII-Code für Anzeige des Display Nr.2	V_ANZNR2	Variable 94	Ⓟ
Format für Display Nr.2	V_ANZMSK2	Variable 95	Ⓟ
Editorzahl für Display Nr.2	V_INPDAT2	Variable 96	Ⓟ
Format für Display Nr.2	V_INPMSK2	Variable 97	Ⓟ
Textoffset für Sprachumschaltung	V_TXTOFFS	Variable 98	Ⓟ

▮ Tabelle 91 – Systemvariablen zur Displayprogrammierung

Bitte beachten Sie, daß die tatsächliche Funktion der Displaybefehle abhängig von den verwendeten Anzeigemodulen ist. Eine Einführung in die Display-Programmierung erhalten Sie in Kapitel 4.2 - Display-Programmierung (Seite 160), eine Befehlsübersicht unter Display und Texte (Seite 43).

## 6.15 Tastaturen

Für das MC200 System sind unterschiedliche Tastaturen bzw. Bedieneinheiten verfügbar. Da diese Tastaturen eine unterschiedliche Anzahl Tasten und eine abweichende Gestaltung aufweisen, ist die Zuordnung von Merkern zu der jeweiligen Tastatur nicht einheitlich.

Grundsätzlich sind folgende Datenbereiche innerhalb des MC200 Systems für die Tastatureinheit reserviert:

Merkerbereich	Funktion
Merker 2305-2432	Tastenmerker: wenn gesetzt, wurde die entsprechende Taste gedrückt. Achtung! Der jeweilige Merker muß vom SPS-Programm zurückgesetzt werden.
Merker 2945-3200	LED-Merker: wenn das entsprechende Bedienteil über eine Tastatur mit integrierten LEDs verfügt, wird hier der Zustand dieser LEDs gespeichert.

n Tabelle 92 – Tastenmerker (Gesamtübersicht)

Im folgenden wird nun die Zuordnung der Merker für die gegenwärtig erhältlichen Bedienteile der MC200 Familie dargestellt.

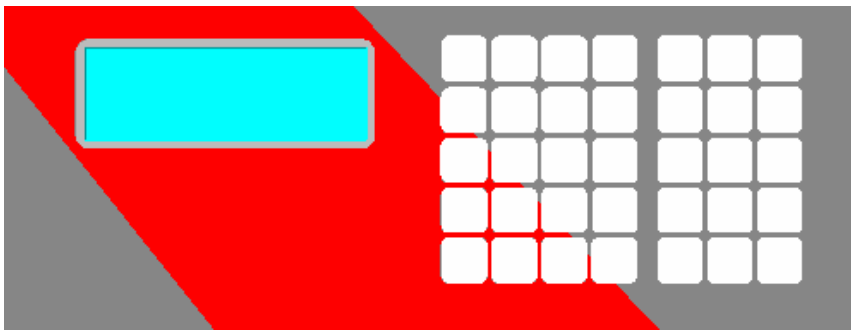
### n MC200BED (stehend, 20 Tasten)



Merker 2305	Merker 2306	Merker 2307	Merker 2308
Merker 2313	Merker 2314	Merker 2315	Merker 2316
Merker 2321	Merker 2322	Merker 2323	Merker 2324
Merker 2329	Merker 2330	Merker 2331	Merker 2332
Merker 2337	Merker 2338	Merker 2339	Merker 2340

n Abbildung 3 – MC200BED (stehend, 20 Tasten)

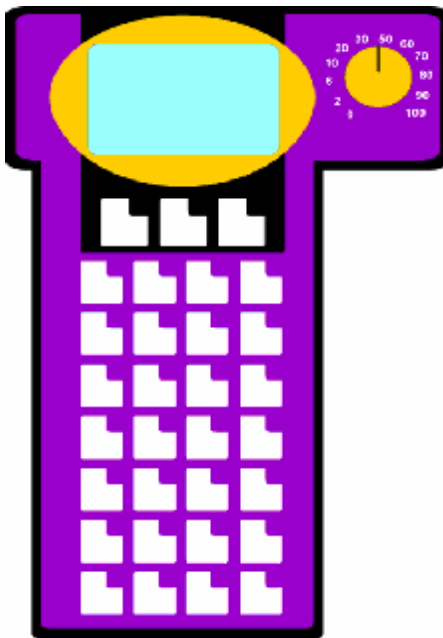
## n MC200BED (liegend, 36 Tasten)



M2305	M2306	M2307	M2308		M2309	M2310	M2311
M2313	M2314	M2315	M2316		M2317	M2318	M2319
M2321	M2322	M2323	M2324		M2325	M2326	M2327
M2329	M2330	M2331	M2332		M2333	M2334	M2335
M2337	M2338	M2339	M2340		M2341	M2342	M2343

n Abbildung 4 – MC200BED (liegend, 36 Tasten)

## n MC200HT



Taste 1	Taste 9		Taste 25
Taste 2	Taste 10	Taste 18	Taste 26
Taste 3	Taste 11	Taste 19	Taste 27
Taste 4	Taste 12	Taste 20	Taste 28
Taste 5	Taste 13	Taste 21	Taste 29
Taste 6	Taste 14	Taste 22	Taste 30
Taste 7	Taste 15	Taste 23	Taste 31
Taste 8	Taste 16	Taste 24	Taste 32

n Abbildung 5 – MC200HT

Die Zuordnung der Tasten zu den einzelnen Merkern finden Sie auf der nächsten Seite.

Taste	Tastenmerker	LED rot	LED grün	Blinken schnell	Blink. langsam
1	M2305	M2945	M3009	M3073	M3137
2	M2306	M2946	M3010	M3074	M3138
3	M2307	M2947	M3011	M3075	M3139
4	M2308	M2948	M3012	M3076	M3140
5	M2309	M2949	M3013	M3077	M3141
6	M2310	M2950	M3014	M3078	M3142
7	M2311	M2951	M3015	M3079	M3143
8	M2312	M2952	M3016	M3080	M3144
9	M2313	M2953	M3017	M3081	M3141
10	M2314	M2954	M3018	M3082	M3142
11	M2315	M2955	M3019	M3083	M3143
12	M2316	M2956	M3020	M3084	M3144
13	M2317	M2957	M3021	M3085	M3145
14	M2318	M2958	M3022	M3086	M3146
15	M2319	M2959	M3023	M3087	M3147
16	M2320	M2960	M3024	M3088	M3148
18	M2322	M2962	M3026	M3090	M3150
19	M2323	M2963	M3027	M3091	M3151
20	M2324	M2964	M3028	M3092	M3152
21	M2325	M2965	M3029	M3093	M3153
22	M2326	M2966	M3030	M3094	M3154
23	M2327	M2967	M3031	M3095	M3155
24	M2328	M2968	M3032	M3096	M3156
25	M2329	M2969	M3033	M3097	M3157
26	M2330	M2970	M3034	M3098	M3158
27	M2331	M2971	M3035	M3099	M3159
28	M2332	M2972	M3036	M3100	M3160
29	M2333	M2973	M3037	M3101	M3161
30	M2334	M2974	M3038	M3102	M3162
31	M2335	M2975	M3039	M3103	M3164
32	M2336	M2976	M3040	M3104	M3164

n Tabelle 93 – Tasten- und LED-Merker für das MC200HT

## 6.16 Elektrisches Handrad

Bei Verwendung eines MC200BDL oder MC200BED-ABA können bis zu zwei Impulsgeber an das Display angeschlossen werden. Die Anzahl der Impulse wird in Systemvariablen gespeichert. Sie können diese Werte im MC200 System frei verwenden.

Funktion	Name	Adresse	
Elektrisches Handrad 1	V_ELHAND1	Variable 117	Ⓟ
Elektrisches Handrad 2	V_ELHAND2	Variable 118	Ⓟ
Bewertungsfaktor elektrisches Handrad, mit diesem Wert wird der Impuls am Handrad multipliziert bevor er in V_ELHAND übertragen wird	V_ELHFAK	Variable 119	Ⓟ

■ Tabelle 94 – Systemmerker und –Variablen für das elektrische Handrad

## 6.17 Eingebaute serielle Schnittstelle

Jedes MC200 System verfügt über eine eingebaute serielle Schnittstelle. Normalerweise wird diese Schnittstelle zur Kommunikation mit einem angeschlossenen PC bzw. zur Wartung, Diagnose und Programmierung des Systems verwendet.

Sie können diese Schnittstelle aber auch verwenden, um programmgesteuert serielle Daten über die Schnittstelle an ein anderes, serielles Gerät zu senden. Wie im Kapitel 4.2 - Display-Programmierung (Seite 160) beschrieben, verwenden Sie hierzu die gleichen Befehle wie zur Ansteuerung des Displays.

Funktion	Name	Adresse	
Textnummer, Zahl oder ASCII-Code für die eingebaute serielle Schnittstelle (drucken)	V_VANZNR3	Variable 99	b
Format für die eingebaute serielle Schnittstelle	V_ANZMSK3	Variable 100	b
Aktivitätsmerker für eingebaute serielle Schnittstelle: wenn ein, werden gerade Daten ausgegeben	M_RS232	Merker 2088	y

n Tabelle 95 – Systemvariablen und –Merker für die eingebaute serielle Schnittstelle

## 6.18 Serielles Erweiterungsmodul

Für jedes serielle Erweiterungsmodul der MC200 Familie stehen vier Systemmerker- und Variablen sowie ein Parameter für die grundsätzliche Konfiguration des Moduls zur Verfügung. Bitte beachten Sie, daß Sie die Fehlermerker aus dem SPS-Programm heraus nach Bearbeitung der Störung wieder zurücksetzen müssen.

Funktion	Name	Adresse	
Modul 1: Neue serielle Daten empfangen	M_SERIN_1	Merker 2289	Ⓟ
Modul 2: Neue serielle Daten empfangen	M_SERIN_2	Merker 2290	Ⓟ
Modul 3: Neue serielle Daten empfangen	M_SERIN_3	Merker 2291	Ⓟ
Modul 4: Neue serielle Daten empfangen	M_SERIN_4	Merker 2292	Ⓟ
Modul 5: Neue serielle Daten empfangen	M_SERIN_5	Merker 2293	Ⓟ
Modul 6: Neue serielle Daten empfangen	M_SERIN_6	Merker 2294	Ⓟ
Modul 7: Neue serielle Daten empfangen	M_SERIN_7	Merker 2295	Ⓟ
Modul 8: Neue serielle Daten empfangen	M_SERIN_8	Merker 2296	Ⓟ
Modul 1: Anzahl Byte im Eingangspuffer	V_SERIN1	Variable 49	Ⓟ
Modul 2: Anzahl Byte im Eingangspuffer	V_SERIN2	Variable 50	Ⓟ
Modul 3: Anzahl Byte im Eingangspuffer	V_SERIN3	Variable 51	Ⓟ
Modul 4: Anzahl Byte im Eingangspuffer	V_SERIN4	Variable 52	Ⓟ
Modul 5: Anzahl Byte im Eingangspuffer	V_SERIN5	Variable 53	Ⓟ
Modul 6: Anzahl Byte im Eingangspuffer	V_SERIN6	Variable 54	Ⓟ
Modul 7: Anzahl Byte im Eingangspuffer	V_SERIN7	Variable 55	Ⓟ
Modul 8: Anzahl Byte im Eingangspuffer	V_SERIN8	Variable 56	Ⓟ
Modul 1: Empfangspuffer voll	M_SERIN_OV_1	Merker 3265	Ⓟ
Modul 2: Empfangspuffer voll	M_SERIN_OV_2	Merker 3266	Ⓟ
Modul 3: Empfangspuffer voll	M_SERIN_OV_3	Merker 3267	Ⓟ
Modul 4: Empfangspuffer voll	M_SERIN_OV_4	Merker 3268	Ⓟ
Modul 5: Empfangspuffer voll	M_SERIN_OV_5	Merker 3269	Ⓟ
Modul 6: Empfangspuffer voll	M_SERIN_OV_6	Merker 3270	Ⓟ
Modul 7: Empfangspuffer voll	M_SERIN_OV_7	Merker 3271	Ⓟ
Modul 8: Empfangspuffer voll	M_SERIN_OV_8	Merker 3272	Ⓟ
Modul 1: Sendepuffer voll	M_SEROUT_OV_1	Merker 3273	Ⓟ
Modul 2: Sendepuffer voll	M_SEROUT_OV_2	Merker 3274	Ⓟ
Modul 3: Sendepuffer voll	M_SEROUT_OV_3	Merker 3275	Ⓟ
Modul 4: Sendepuffer voll	M_SEROUT_OV_4	Merker 3276	Ⓟ
Modul 5: Sendepuffer voll	M_SEROUT_OV_5	Merker 3277	Ⓟ
Modul 6: Sendepuffer voll	M_SEROUT_OV_6	Merker 3278	Ⓟ
Modul 7: Sendepuffer voll	M_SEROUT_OV_7	Merker 3279	Ⓟ
Modul 8: Sendepuffer voll	M_SEROUT_OV_8	Merker 3280	Ⓟ



Funktion	Name	Adresse	
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung erstes serielles Modul	60	b
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung zweites serielles Modul	61	b
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung drittes serielles Modul	62	b
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung viertes serielles Modul	63	b
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung fünftes serielles Modul	64	b
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung sechstes serielles Modul	65	b
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung siebtes serielles Modul	66	b
Binärcodierte Parameter für das serielle Erweiterungsmodul	Parameterierung achttes serielles Modul	67	b
Daten für Ausgabe mit SETSER	V_SERNR	Variable 99	b
Formatvariable für Ausgabe mit SETSER	V_SERMSK	Variable 100	

n Tabelle 96 – Systemmerker und –Variablen zu den seriellen Erweiterungsmodulen

## 6.19 Übersicht Speichermanagement

### n MC200CPU (Typen B und C)

n 128 kB RAM

n 128 kB Flash-EPROM

Speicherbereich (hex)	Typ	Inhalt
00000 – 002FF	EPROM	Interrupt-Vektortabellen, Sprungtabellen
00300 – 003F8	EPROM	Universal Data Block (Speicher-Inhaltsverzeichnis)
003F9 – 0DFFF	EPROM	MC200CPU Betriebssystem
0E000 – 0E5FF	EPROM	Im Flash gespeicherte Achs- und Systemparameter
0E600 – 0F9FF	EPROM	MC200CPU Betriebssystem
0FA00 – 0FBFF	RAM	Internes Prozessor-RAM, enthält 80C166 Registerbänke, Stackspeicher und ähnliches
0FC00 – 0FCFF	RAM	SPS-Merker (4096 Merker zu je 1 Bit)
0FD00 – 0FDFF	RAM	Arbeitsspeicher des CPU-Moduls, enthält diverse Puffer für die Modulkommunikation, SPS-Statusinformationen usw.
0FE00 – 0FFFF	RAM	Internes Prozessor-RAM, enthält 80C166 Register
10000 – 13FFF	EPROM	Im Flash gespeichertes SPS-Programm (Page 1)
14000 – 17FFF	EPROM	Im Flash gespeichertes SPS-Programm (Page 2)
18000 – 1BF7F	EPROM	Im Flash gespeicherte SPS-Texte
1BF80 – 1BFFF	EPROM	Informationen zum im Flash gespeicherten SPS-Programm
1C000 – 1FF7F	EPROM	Im Flash gespeicherte Variablen (Silicon-Flash-Disk)
1FF80 – 1FFFF	EPROM	Interne Systeminformationen
20000 – 23FFF	RAM	SPS-Programm (Page 1)
24000 – 27FFF	RAM	SPS-Programm (Page 2)
28000 – 2BFFF	RAM	Reserviert für Erweiterungen
2C000 – 2FF7F	RAM	SPS-Texte
2FF80 – 2FFFF	RAM	Reserviert für Erweiterungen
30000 – 37FFF	RAM	SPS-Variablen (8192 Variablen zu je 32 Bit)
38000 – 396E9	RAM	Erweiterter Arbeitsspeicher des CPU-Moduls, enthält diverse Systeminformationen, Fließkommaberechnungstabellen usw.
396EA – 39769	RAM	Achs- und Systemparameter
3976A – 3BFFF	RAM	Diverse Systempuffer
3C000 – 3FF7F	RAM	MCSave™ Ereignisspeicher (4000 Ereignisse zu je 32 Bit)
3FF80 – 3FFFF	RAM	Checksummen und Konsistenzprüfungen

n Tabelle 97 – Speicherstruktur MC200CPU (Typen B und C)

## n MC200PROFI/B

n 256 kB RAM

n 128 kB Flash-EPROM

Speicherbereich (hex)	Typ	Inhalt
00000 – 002FF	EPROM	Interrupt-Vektortabellen, Sprungtabellen
00300 – 003F8	EPROM	Universal Data Block (Speicher-Inhaltsverzeichnis)
003F9 – 0DFFF	EPROM	MC200CPU Betriebssystem
0E000 – 0E5FF	EPROM	Im Flash gespeicherte Achs- und Systemparameter
0E600 – 0F9FF	EPROM	MC200CPU Betriebssystem
0FA00 – 0FBFF	RAM	Internes Prozessor-RAM, enthält 80C166 Registerbänke, Stackspeicher und ähnliches
0FC00 – 0FCFF	RAM	SPS-Merker (4096 Merker zu je 1 Bit)
0FD00 – 0FDFF	RAM	Arbeitsspeicher des CPU-Moduls, enthält diverse Puffer für die Modulkommunikation, SPS-Statusinformationen usw.
0FE00 – 0FFFF	RAM	Internes Prozessor-RAM, enthält 80C166 Register
10000 – 13FFF	EPROM	Im Flash gespeichertes SPS-Programm (Page 1)
14000 – 17FFF	EPROM	Im Flash gespeichertes SPS-Programm (Page 2)
18000 – 1BF7F	EPROM	Im Flash gespeicherte SPS-Texte
1BF80 – 1BFFF	EPROM	Informationen zum im Flash gespeicherten SPS-Programm
1C000 – 1FF7F	EPROM	Im Flash gespeicherte Variablen (Silicon-Flash-Disk)
1FF80 – 1FFFF	EPROM	Interne Systeminformationen
20000 – 27FFF	RAM	SPS-Variablen (8192 Variablen zu je 32 Bit)
28000 – 296E9	RAM	Erweiterter Arbeitsspeicher des CPU-Moduls, enthält diverse Systeminformationen, Fließkommaberechnungstabellen usw.
296EA – 29769	RAM	Achs- und Systemparameter
2976A – 2BFFF	RAM	Diverse Systempuffer
2C000 – 2FF7F	RAM	MCSave™ Ereignisspeicher (4000 Ereignisse zu je 32 Bit)
2FF80 – 2FFFF	RAM	Checksummen und Konsistenzprüfungen
30000 – 3FFFF	RAM	Profibus SPC3 DP-Slave RAM
40000 – 43FFF	RAM	SPS-Programm (Page 1)
44000 – 47FFF	RAM	SPS-Programm (Page 2)
48000 – 4BFFF	RAM	Reserviert für Erweiterungen
4C000 – 4FF7F	RAM	SPS-Texte
4FF80 – 4FFFF	RAM	Reserviert für Erweiterungen

n Tabelle 98 – Speicherstruktur MC200PROFI/B

## n Aufbau des Universal Data Block (UDB)

Die Position einzelner Informationen innerhalb der CPU-Module kann sich jederzeit ändern – das ist im Zuge der Fortentwicklung und Erweiterung der Module schlichtweg normal., stellt jedoch gerade für z.B. eine PC-Software, die direkt auf den Speicher der Module zugreift, ein Problem dar. Zur Lösung haben wir den UDB erfunden und in unsere CPU-Module integriert: an einer festen Position im Speicher (00300 hex) steht eine Art Inhaltsverzeichnis der wichtigsten Speicherstellen. Durch Auswerten dieses Inhaltsverzeichnisses kann dann ein stets korrekter Zugriff auf die richtigen Speicherzellen erfolgen.

Das Inhaltsverzeichnis hat folgenden, grundsätzlichen Aufbau:

- n Jeder Eintrag besteht aus 6 Byte
- n Die ersten 4 Byte enthalten die Speicheradresse
- n Die letzten 2 Byte enthalten die Länge der Informationen

### Beispiel

Um nun z.B. die Speicheradresse der Variable 1322 zu ermitteln ist wie folgt vorzugehen:

- n Lesen von 6 Byte ab der Speicheradresse 348 hex (UDB-Eintrag "SPS-Variablen Page 1", siehe Tabelle unten). Bei Version 2.22 der MC200CPU/C würden wir jetzt z.B. folgende Daten zurückerhalten (alle Werte hexadezimal): 00 00 03 00 00 40.
- n Die ersten 4 Byte enthalten die Speicheradresse der "SPS-Variablen Page 1" in LSB-MSB Schreibweise. Dies ergibt eine Adresse von 030000 hex.
- n Die letzten 2 Byte erhalten die Länge dieses Blocks, ebenfalls in LSB-MSB Schreibweise. Dies ergibt eine Länge von 4000 hex (oder 16384 dezimal) Byte.
- n Jetzt haben wir die Adresse der Variable 0, wir wollen aber die Variable 1322 auslesen. Deshalb addieren wir  $4 \times 1322 = 5288$  Byte zu unserer Startadresse hinzu und erhalten den Wert 314A8 hex als Zieladresse.
- n In den Speicherstellen 314A8 – 314AB hex ist der Wert der Variablen 1322 in LSB-MSB Schreibweise (32 Bit) abgelegt.

### Übersicht des UDB Speichers

Die Reihenfolge, in der die Informationen abgelegt sind, ist wie folgt definiert:

Adresse (hex)	Inhalt
00300	Letzte gültige Adresse des UDB: durch Auslesen dieser Speicherzelle kann die Länge des UDB ermittelt werden
00306	80C166 Registerbänke
0030C	MC100-kompatibler SPS-Haltepunkt (nicht verwendet)
00312	80C166 Stacks
00318	SPS-Programm (Page 1)
0031E	SPS-Programm (Page 2)
00324	Reserviert für Erweiterungen
0032A	SPS-Texte
00330	"Interne" Merker (Bereich 2048 – 4095)
00336	Statusinformationen digitale Ausgänge
0033C	Statusinformationen digitale Eingänge
00342	"Externe" Merker (Bereich 1 – 2047)

Adresse (hex)	Inhalt
00348	SPS-Variablen (Page 1, Bereich 1 – 4095)
0034E	SPS-Variablen (Page 2, Bereich 4096 – 8191)
00354	MCSave Ereignisdaten
0035A	MCSave Ereignisaktivierungs-Tabelle (Event-Filter)
00360	Display-Textpuffer (Display 1)
00366	Display-Textpuffer (Display 2)
0036C	Logisch/Physikalische Zuordnungstabelle Achsmodule
00372	Logisch/Physikalische Zuordnungstabelle digitale Eingänge
00378	Logisch/Physikalische Zuordnungstabelle digitale Ausgänge
0037E	Logisch/Physikalische Zuordnungstabelle Handbedienteile
00384	Logisch/Physikalische Zuordnungstabelle Busdezentralisierung
0038A	Logisch/Physikalische Zuordnungstabelle analoge Ein-/Ausgänge
00390	Logisch/Physikalische Zuordnungstabelle Zählermodule
00396	Gesamtanzahl der angeschlossenen Achsen
0039C	Programmierte Zielpositionen für 16 Achsen
003A2	Programmierte Sollgeschwindigkeit für 16 Achsen
003A8	Programmierte Beschleunigung für 16 Achsen
003AE	Aktuelle Schleppfehler für 16 Achsen
003B4	Aktuelle Ist-Geschwindigkeit für 16 Achsen
003BA	Stack-Speicher für 32 SPS-Haltepunkte
003C0	Zwangsmaske für digitale Eingänge (nicht verwendet)
003C6	Zusätzliche Informationen zum aktuellen SPS-Programm
003CC	Neustart-Informationen (bitcodiert)
003D2	Aktuell gewählte Display-Textseite
003D8	Logisch/Physikalische Zuordnungstabelle serielle Erweiterungsmodule
003DE	Reserviert für Erweiterungen
003E4	Reserviert für Erweiterungen
003EA	Reserviert für Erweiterungen
003F0	Reserviert für Erweiterungen

n Tabelle 99 – Übersicht UDB-Speicher

## n Raum für Ihre Notizen

# Kapitel 7 Beispiele

In diesem Abschnitt finden Sie einige praxisbezogene Beispiele zur Programmierung in der MC-1B Sprache. Alle hier aufgeführten Programme sind direkt und ohne jede Änderung lauffähig. Dennoch sind sie in erster Linie für Demonstrations- und Lernzwecke gedacht, und sollten in dieser Form nicht direkt in andere Programme übernommen werden.

Wir haben in dieser Dokumentation folgende Beispiele für Sie aufbereitet:

## n Achspositionierung

Zeigt in einem kurzen Programm das Einschalten der Leistungsteile, die Referenzfahrt, fahren in Grundstellung für zwei Achsen und dann eine interpolierte Bewegung.

## n Lauflicht

Demonstriert anschaulich den Unterschied zwischen der klassischen Schrittkettenprogrammierung und modernen Programmiermethoden wie Schleifen und indirekten Zugriffen anhand eines Lauflichtes.

## n Display-Programmierung

Zeigt einen Willkommensbildschirm und beinhaltet anschließend eine Menüführung zum Setzen oder Löschen von Ausgängen sowie zum Verändern einer Variable.

## n Mixed Mode (PC-Anbindung)

Erläutert anhand eines Beispiels die Möglichkeiten, die Sie zur Anbindung der Steuerung an eine PC-Oberfläche haben. Dieses Beispiel gilt natürlich nur dann, wenn Sie die eigentlichen Abläufe in MC-1B programmieren. Für eine vollständige Ablaufsteuerung über den PC lesen Sie bitte die entsprechenden VMC Dokumentationen.

**Alle hier abgedruckten Beispiele finden sich auch als vollständig lauffähige Projekte auf der VMC Workbench CD. Durchsuchen Sie einfach mit dem Window-Explorer Ihr VMC Workbench Verzeichnis nach dem Unterverzeichnis "Beispiele\MC-1B".**

## 7.1 Achspositionierung

```

//*****
/** Projekt:   Demo Achsen
/** Datei:    Main.MC
/** Dateityp: MC-1B Quellcode
/**
/** Erstellt am 24.02.99 um 22:01:50 von Mi cha
//*****

DEF_E      7, E_START          // SPS-Eingang: Bewegung starten
DEF_A      14, A_FERTIG       // SPS-Ausgang: Bewegung fertig
DEF_M      2340, M_START      // Bedienpult-Taste: START

// Dieses kleine Beispiel demonstriert, wie einfach die Programmierung
// von Bewegungen mit der MC-1B Sprache ist. In diesem Beispiel
// wurden der Einfachheit halber keine Fehlerüberwachungen mit
// eingebaut.

PWRDRV     1, 1                // Leistungsteil Achse 1 einschalten
PWRDRV     2, 1                // Leistungsteil Achse 2 einschalten

WarteLT:
NLAD_M     M_RFGON_A1         // Reglerfreigabe Achse noch nicht da?
NODER_M    M_RFGON_A2         // ...oder Achse 2?
SPRINGN    WarteLT           // Dann warten wir

STHOME     1, 0                // Referenzfahrt Achse 1 starten
GEHUPRI    WarteA1           // Unterprogramm wartet auf Achse 1

STHOME     2, 0                // Referenzfahrt Achse 2 starten
GEHUPRI    WarteA2           // Unterprogramm wartet auf Achse 2

Grundstellung:
LAD_VA     VARERG, 5000        // Zielposition laden
STPABS     1, VARERG           // Achse 1 auf Absolutposition 5000
LAD_VA     VARERG, -2000      // Zielposition laden
STPABS     2, VARERG           // Achse 2 auf Absolutposition -2000

GEHUPRI    WarteA             // Unterprogramm wartet auf beide Achsen

// Wir warten jetzt auf unser Startsignal, das entweder über
// einen SPS-Eingang oder über das Bedienpult kommen kann.

WarteStart:
LAD_E      E_START            // Eingang E_START eingeschaltet?
ODER_M     M_START            // ...oder Start-Taste gedrückt?
SPRINGN    WarteStart         // Nein, wir warten

AUS_A      A_FERTIG           // Ausgang "Bewegung fertig" löschen

LAD_VA     VARERG, 500         // Zielposition laden für Achse 1
STIABS     1, VARERG           // Linearinterpolation vorbereiten
LAD_VA     VARERG, 3000       // Zielposition laden
STIABS     2, VARERG           // Linearinterpolation für Achse 1+2 starten

GEHUPRI    WarteA             // Warten bis in Position

EIN_A      A_FERTIG           // SPS-Ausgang setzen: Bewegung fertig

```



```
SPRING Grundstellung // Zurück in die Ausgangsposition

WarteA1:
LAD_M M_INPOS_A1 // Achse 1 in Position?
SPRINGN WarteA1 // Nein, wir warten
UPREND // Unterprogramm Ende

WarteA2:
LAD_M M_INPOS_A2 // Achse 2 in Position?
SPRINGN WarteA2 // Nein, wir warten
UPREND // Unterprogramm Ende

WarteA:
LAD_M M_INPOS_A1 // Achse 1 in Position?
UND_M M_INPOS_A2 // ... und Achse 2 in Position?
SPRINGN WarteA // Nein, wir warten
UPREND // Unterprogramm Ende
```

Selbst eine komplizierte Achsbewegung wie die lineare Interpolation ist mit der MC-1B Sprache in nur 4 Zeilen zu programmieren: Nicht umsonst haben wir diese Sprache speziell für die Positionierung optimiert! Auch andere Formen der Bewegung, spezielle Messungen, Sonderfunktionen wie "fliegende Säge" oder "Punkteimung", lassen sich ohne sonderlichen Aufwand direkt aus der SPS realisieren.

## 7.2 Lauflicht

```

//*****
/** Projekt:   Demo Lauflicht
/** Datei:    Main.MC
/** Dateityp: MC-1B Quellcode
/**
/** Erstellt am 24.02.99 um 21:14:11 von Mi cha
//*****

// Dieses Programm schaltet ein simples Lauflicht mit Hilfe
// von 8 digitalen Eingängen und 8 digitalen Ausgängen,
// die miteinander 1:1 verbunden sind (d.h. Ausgang 1 auf
// Eingang 1, Ausgang 2 auf Eingang 2 usw).

// Wir zeigen dieses Lauflicht hier in zwei unterschiedlichen
// Programmieretechniken: einmal als klassische Schrittkette,
// und einmal als Schleife mit indirekter Ein-/Ausgangsadressierung,
// ähnlich wie man dies in einer Hochsprache programmieren würde.

// Zum Testen schreiben Sie in die Variable V_DEMO (V200) den
// Wert von 1 für die Schrittkette, oder den Wert von 2 für
// die indirekte Adressierung.

DEF_V      200, V_DEMO
DEF_V      201, V_SCHRITT
DEF_V      202, V_ZAEHLER
DEF_M      1, M_STATUS

LAD_VA     V_DEMO, 0           // Auswahl der Demo
LAD_VA     V_SCHRITT, 1       // Immer mit dem ersten Schritt beginnen...
LAD_VA     V_ZAEHLER, 1       // ...und auch den Zähler zurücksetzen
EIN_M      M_STATUS           // Statusmerker für Indirekt-Demo einschalten

Start:
VERG_VA    V_DEMO, 1          // Schrittdemo?
LAD_M      M_GLEICH           // Vergleichsergebnis prüfen
SPRINGJ    Schritte          // Demo aufrufen

VERG_VA    V_DEMO, 2          // Demo mit indirekter Adressierung?
LAD_M      M_GLEICH           // Vergleichsergebnis prüfen
SPRINGJ    Indirekt          // Demo aufrufen

SPRING     Start              // Warten

//*****
/** Teil 1: Lauflicht als Schrittkette
//*****

Schritte:
GEHUPRJ    Schritt1          // Nächster Schritt der Schrittkette
SPRING     Schritte          // Hauptprogrammschleife

Schritt1:
VERG_VA    V_SCHRITT, 1       // Bereits ausgeführt?
LAD_M      M_GLEICH           // Vergleichsergebnis prüfen
SPRINGN    Schritt2          // Ja, dann zum nächsten Schritt

EIN_A      1                  // Ausgang 1 einschalten

```

```

INC_V      V_SCHRI TT, 1      // Nächster Schritt
UPREND     // Unterprogramm Ende

Schritt 2: // Schritt 2: A2 einschalten, sobald E1 ein
VERG_VA    V_SCHRI TT, 2      // Bereits ausgeführt?
LAD_M      M_GLEICH           // Vergleichsergebnis prüfen
SPRINGN    Schritt3          // Ja, dann zum nächsten Schritt

LAD_E      1                  // Eingang 1 eingeschaltet?
UPRENDN    // Nein, Unterprogramm Ende
EIN_A      2                  // Ausgang 2 einschalten
INC_V      V_SCHRI TT, 1      // Nächster Schritt
UPREND     // Unterprogramm Ende

Schritt 3: // Schritt 3: A3 einschalten, sobald E2 ein
VERG_VA    V_SCHRI TT, 3
LAD_M      M_GLEICH
SPRINGN    Schritt4

LAD_E      2
UPRENDN
EIN_A      3
INC_V      V_SCHRI TT, 1
UPREND

Schritt 4: // Schritt 4: A4 einschalten, sobald E3 ein
VERG_VA    V_SCHRI TT, 4
LAD_M      M_GLEICH
SPRINGN    Schritt5

LAD_E      3
UPRENDN
EIN_A      4
INC_V      V_SCHRI TT, 1
UPREND

Schritt 5: // Schritt 5: A5 einschalten, sobald E4 ein
VERG_VA    V_SCHRI TT, 5
LAD_M      M_GLEICH
SPRINGN    Schritt6

LAD_E      4
UPRENDN
EIN_A      5
INC_V      V_SCHRI TT, 1
UPREND

Schritt 6: // Schritt 6: A6 einschalten, sobald E5 ein
VERG_VA    V_SCHRI TT, 6
LAD_M      M_GLEICH
SPRINGN    Schritt7

LAD_E      5
UPRENDN
EIN_A      6
INC_V      V_SCHRI TT, 1
UPREND

```

```

Schritt7:                                     // Schritt 7: A7 einschalten, sobald E1 ein
VERG_VA    V_SCHRI TT, 7
LAD_M      M_GLEI CH
SPRINGN    Schritt8

LAD_E      6
UPRENDN
EIN_A      7
INC_V      V_SCHRI TT, 1
UPREND

Schritt8:                                     // Schritt 8: A8 einschalten, sobald E7 ein
VERG_VA    V_SCHRI TT, 8
LAD_M      M_GLEI CH
SPRINGN    Schritt9

LAD_E      7
UPRENDN
EIN_A      8
INC_V      V_SCHRI TT, 1
UPREND

Schritt9:                                     // Schritt 9: Warten auf E8
VERG_VA    V_SCHRI TT, 9
LAD_M      M_GLEI CH
SPRINGN    Schritt10

LAD_E      8
UPRENDN
INC_V      V_SCHRI TT, 1
UPREND

Schritt10:                                   // Schritt 10: Alle Ausgänge wieder löschen
LAD_M      M_EI N                               // BES einschalten
AUS_A      8                                   // Alle Ausgänge löschen
AUS_A      7
AUS_A      6
AUS_A      5
AUS_A      4
AUS_A      3
AUS_A      2
AUS_A      1
LAD_VA    V_SCHRI TT, 1                       // Wieder mit Schritt 1 beginnen
UPREND                                         // Unterprogramm Ende

```

```

//*****
//* Teil 2: Lauflicht mit indirekter Adressierung
//*****

Indirekt:
GEHUPRJ   Ei nschal ten           // Unterprogramm aufrufen
SPRING    Indi rekt              // Schl ei fe

Ei nschal ten:
VERG_VA   V_ZAEHLER, 9           // Ausgang (Zähler) ei nschal ten,
// wenn Ei ngang (Zähler-1) ei n
LAD_M     M_GLEICH                // Berei ts alle 8 ei ngeschal tet?
SPRINGJ   Ausschal ten          // Dann jetz t wieder ausschal ten

VERG_VA   V_ZAEHLER, 1           // Erster Ausgang?
LAD_M     M_GLEICH                // Ergebni s prüf en
EIN_AI    V_ZAEHLER              // Dann gl eich ei nschal ten, ohne Ei ngang zu
prüf en
INC_V     V_ZAEHLER, 1           // Zähl er erhöh en
UPRENDJ   // Unterprogramm Ende

LAD_M     M_EIN                  // BES ei nschal ten
SUB_VA    V_ZAEHLER, 1           // Ei ngang (Zähl er-1) ermittle n
LAD_EI    VARERG                 // Ei ngang ei ngeschal tet?
UPRENDN   // Nei n, Unterprogramm Ende

EIN_AI    V_ZAEHLER              // Ausgang ei nschal ten
INC_V     V_ZAEHLER, 1           // Zähl er erhöh en
UPREND    // Unterprogramm Ende

Ausschal ten:
DEC_V     V_ZAEHLER, 1           // Alle Ausgänge ausschal ten
// Zähl er ei n herunter
AUS_AI    V_ZAEHLER              // Ausgang ausschal ten
VERG_VA   V_ZAEHLER, 1           // Noch nicht alle Ausgänge ausgeschal tet?
LAD_M     M_GROESSER            // Ergebni s prüf en
SPRINGJ   Ausschal ten          // Nei n, wei ter
UPREND    // Unterprogramm Ende

```

Dieses Beispiel zeigt deutlich, daß die modernen Programmiermöglichkeiten mit MC-1B auch andere Lösungen zulassen, als Sie von sonstigen SPS-Programmiersprachen gewöhnt sein mögen. Denken Sie also ruhig auch einmal "um die Ecke", denn, wie aus diesem Beispiel ersichtlich, benötigt die indirekte Programmiermethode gerade einmal etwa 1/3 des Programmcodes wie die klassische Schrittkette.

## 7.3 Displayprogrammierung

### n Hauptprogramm (Datei MAIN.MC)

```

//*****
/** Projekt:   Demo Display
/** Datei:    Main.MC
/** Dateityp: MC-1B Quellcode
/**
/** Erstellt am 24.02.99 um 16:05:03 von Micha
//*****

// Einige symbolische Definitionen

DEF_V    200, V_DEMO           // Variable zum Hoch-/Runterzählen
DEF_M    2329, M_KEY_1         // Taste 1
DEF_M    2330, M_KEY_2         // Taste 2
DEF_M    2332, M_KEY_END       // Taste ENDE oder STOP
DEF_M    2324, M_KEY_MINUS     // Taste Minus
DEF_M    2316, M_KEY_PLUS      // Taste Plus
DEF_M    2340, M_KEY_START     // Taste START

DEF_V    201, V_ZAEHLER        // Zähler für Lauflicht
DEF_V    202, V_RECHNEN        // Rechenvariable für Lauflicht
DEF_M    10, M_RICHTUNG        // Richtung für Lauflicht

// Als Erstes schreiben wir das Copyright in das Display

Reset:
LAD_DT   V_ANZMSK1, 1, 1, 20   // Text in Zeile 2
LAD_VA   V_ANZNR1, 3           // Text Nr. 3 wählen
SETDSP   1, 1                  // ausgeben

// Jetzt warten wir auf die Taste START, und
// präsentieren währenddessen ein kleines Lauflicht
// in der Zeile 1 und 3. Dafür initialisieren wir eine
// Zählervariable, die uns sagt, an welcher
// Stelle im Display wir uns gerade befinden.

LAD_VA   V_ZAEHLER, 0          // Erste Zeilenposition
LAD_VA   V_RECHNEN, 20         // Maximale Zeilenlänge
EIN_M    M_RICHTUNG           // Erstmal von links nach rechts

Start:
GEHUPRI  Lauflicht             // Unterprogramm "Lauflicht" aufrufen

// In Zeile 4 lassen wir den Text "Weiter-> Taste START"
// alle 0,32 Sekunden blinken.

LAD_DT   V_ANZMSK1, 3, 1, 20   // Text in Zeile 3
LAD_VA   V_ANZNR1, 4           // Normalerweise der Text "Warte..."
LAD_M    M_BLINK32             // alle 0,32s umschalten auf...
LAD_VA   V_ANZNR1, 1           // ...Leerzeichen
LAD_M    M_EIN                 // BES einschalten
SETDSP   1, 1                  // Text ausgeben

NLAD_M   M_KEY_START           // Taste START nicht gedrückt?
SPRINGJ  Start                 // Dann warten wir noch ein wenig

```

```

AUS_M      M_KEY_START          // Taste START zurücksetzen

Haupt:
LAD_DT     V_ANZMSK1, 0, 1, 20  // Text in Zeile 1
LAD_VA     V_ANZNR1, 5         // Normalerweise Text 5
LAD_A      1                   // Wenn Ausgang 1 eingeschaltet...
INC_V      V_ANZNR1, 1         // ... stattdessen Text 6
LAD_M      M_EIN               // BES einschalten
SETDSP     1, 1                // Text ausgeben

LAD_DT     V_ANZMSK1, 1, 1, 20  // Text in Zeile 2
LAD_VA     V_ANZNR1, 7         // Normalerweise Text 7
LAD_A      2                   // Wenn Ausgang 2 eingeschaltet...
INC_V      V_ANZNR1, 1         // ... stattdessen Text 8
LAD_M      M_EIN               // BES einschalten
SETDSP     1, 1                // Text ausgeben

LAD_DT     V_ANZMSK1, 2, 1, 20  // Text in Zeile 3
LAD_VA     V_ANZNR1, 9         // Immer Text 9
SETDSP     1, 1                // Text ausgeben

LAD_DT     V_ANZMSK1, 3, 1, 13  // Text in Zeile 4
LAD_VA     V_ANZNR1, 10        // Immer Text 10
SETDSP     1, 1                // Text ausgeben

LAD_DV     V_ANZMSK1, 3, 14, 7  // Text in Zeile 4 ab Pos. 14
LAD_VV     V_ANZNR1, 200        // Wir wollen V200 ausgeben
SETDSP     1, 1                // und 'raus damit

LAD_M      M_KEY_1             // Taste 1 gedrückt?
AUS_M      M_KEY_1             // Ja, Tastenmerker zurücksetzen
GEHUPRJ    A1Toggle           // Ausgang umschalten

LAD_M      M_KEY_2             // Taste 2 gedrückt?
AUS_M      M_KEY_2             // Ja, Tastenmerker zurücksetzen
GEHUPRJ    A2Toggle           // Ausgang umschalten

LAD_M      M_KEY_PLUS          // Taste Plus gedrückt?
AUS_M      M_KEY_PLUS          // Ja, Tastenmerker zurücksetzen...
INC_V      V_DEMO, 1           // ... und V_DEMO um 1 erhöhen

LAD_M      M_KEY_MINUS         // Taste Minus gedrückt?
AUS_M      M_KEY_MINUS         // Ja, Tastenmerker zurücksetzen...
DEC_V      V_DEMO, 1           // ... und V_DEMO um 1 'runterzählen

LAD_M      M_KEY_END           // Taste ENDE gedrückt?
AUS_M      M_KEY_END           // Ja, Tastenmerker zurücksetzen...
SPRINGJ    Reset              // ... und zum Anfang springen

SPRING     Haupt               // Hauptprogrammsequenz

// A1 Toggle schaltet den Ausgang 1 um

A1Toggle:
NLAD_A     1                   // Invertierten Zustand Ausgang 1 laden
MOD_A      1                   // und zurück zum Ausgang übertragen
UPREND     // Ende des Unterprogramms

```

```

// A2 Toggle schaltet den Ausgang 2 um

A2Toggle:
NLAD_A      2          // Invertierten Zustand Ausgang 2 laden
MOD_A      2          // und zurück zum Ausgang übertragen
UPREND      // Ende des Unterprogramms

// Das Unterprogramm "Lauflicht" stellt ein Lauflicht
// beim Start des Programms dar. Dieser Programmteil benutzt
// den internen Aufbau der Display-Variablen, um Variablen
// Textpositionen und -längen zu berechnen.

Lauflicht:
LAD_M      M_TIM_1      // Timer noch nicht abgelaufen?
UPRENDJ    // Dann zurück, kein Update des Lauflichts

LAD_M      M_RICHTUNG   // wenn Lauflicht nach rechts
LAD_DT     V_ANZMSK1, 0, 1, 1 // Text beginnt ganz links
LAD_VA     V_ANZNR1, 11 // Text 11 (Sternchen) wählen
UND_VA     V_ANZMSK1, 65535 // Länge des Textes ausmarkieren
LAD_VV     V_ANZMSK1, VARERG // und wieder in Maske zurückspeichern
MUL_VA     V_ZAEHLER, 65536 // Die Länge des Textes ist in diesem
// Fall variabel, also müssen wir es berechnen
ODER_VV    V_ANZMSK1, VARERG // Ergebnis in V_ANZMSK1 einblenden
LAD_VV     V_ANZMSK1, VARERG // und die Variable zurückladen
INC_V      V_ZAEHLER, 1 // Zähler um eins erhöhen

NLAD_M     M_RICHTUNG   // wenn Lauflicht nach links
LAD_DT     V_ANZMSK1, 0, 1, 20 // Text beginnt ganz links
LAD_VA     V_ANZNR1, 2 // Text 2 (Copyright) wählen
UND_VA     V_ANZMSK1, 255 // Länge und Position ausblenden
LAD_VV     V_ANZMSK1, VARERG // und wieder in Maske zurückspeichern
MUL_VA     V_ZAEHLER, 256 // Textposition berechnen
ODER_VV    V_ANZMSK1, VARERG // Ergebnis in V_ANZMSK1 einblenden
LAD_VV     V_ANZMSK1, VARERG // und wieder in Maske zurückspeichern
SUB_VV     V_RECHNEN, V_ZAEHLER // Länge des Textes berechnen
MUL_VA     VARERG, 65536 // In die oberen Bytes schieben
ODER_VV    V_ANZMSK1, VARERG // Ergebnis in V_ANZMSK1 einblenden
LAD_VV     V_ANZMSK1, VARERG // und wieder in Maske zurückspeichern
DEC_V      V_ZAEHLER, 1 // Zähler um eins runter

LAD_M      M_EIN        // BES zwangsweise einschalten
SETDSP     1, 1         // Text ausgeben

// Die Zeile unterhalb des Copyrights wollen wir auch
// mitlaufen lassen. Dafür addieren wir 32 (= 2 Zeilen)
// zur Formatierungsvariable hinzu und wählen - je
// nach Richtung - entweder Sternchen oder Leerzeilen
// aus.

ADD_VA     V_ANZMSK1, 32 // Auf Zeile 3, aber mit den
// Positionen und Längen, die oben
// angegeben sind
LAD_VV     V_ANZMSK1, VARERG // Zurückschreiben in V_ANZMSK
LAD_VA     V_ANZNR1, 1 // Standardtext ist Leerzeichen
LAD_M      M_RICHTUNG   // Wenn Richtung nach rechts...
LAD_VA     V_ANZNR1, 11 // ...dann stattdessen Sternchen

LAD_M      M_EIN        // BES zwangsweise einschalten

```



```

SETDSP      1, 1                // Text ausgeben

LAD_VA      V_TIM_1, 1         // Timer auf 100ms
EIN_M       M_TIM_1           // ... und starten

VERG_VA     V_ZAEHLER, 21      // obere Grenze erreicht?
LAD_M       M_GLEICH           // Ja, dann wollen wir jetzt
AUS_M       M_RICHTUNG         // in die andere Richtung scrollen
DEC_V       V_ZAEHLER, 1       // Wieder korrekten Wert einstellen

VERG_VA     V_ZAEHLER, 0       // untere Grenze erreicht?
LAD_M       M_KLEINER         // Ja, dann wollen wir jetzt
EIN_M       M_RICHTUNG         // in die andere Richtung scrollen
INC_V       V_ZAEHLER, 1       // Wieder korrekten Wert einstellen

UPREND                                           // Ende Unterprogramm Lauflicht

```

## n Textdefinitionen (Datei TEXT.MCT)

```

//*****
//* Projekt:   Demo Display
//* Dateityp:  MC-1B Textdatei
//*
//* Erstellt am 24.02.99 um 16:05:44 von Micha
//*****

// Textschablone für die Ausrichtung der Texte
//   "12345678901234567890"

TEXT
0001 "
0002 " MC-1B Display Demo "
0003 " (c) 1999 MICRO DESIGN"
0004 "Weiter-> Taste Start"
0005 "<1> Ausgang 1 ein  "
0006 "<1> Ausgang 1 aus  "
0007 "<2> Ausgang 2 ein  "
0008 "<2> Ausgang 2 aus  "
0009 "<+> <-> V200 ändern "
0010 "V200 aktuell:      "
0011 "*****"

```

Sobald Sie sich einmal an die Form der Textprogrammierung gewöhnt haben, werden Sie schnell erkennen, daß Sie so ohne Schwierigkeiten auch komplexe Menüführungen und Benutzereingaben realisieren können. Bitte beachten Sie: der Quellcode bei der Programmierung der Laufschrift ist sicherlich sehr komplex; doch er soll ihnen nur demonstrieren, daß die offene Architektur der MC-1 Sprache auch Eingriffe in den Funktionsablauf jenseits der definierten Befehle zuläßt. Um die Programmierung der Laufschrift zu verstehen, sollten Sie nochmal einen Blick in das 4.2 - Display-Programmierung (ab Seite 160) werfen.

## 7.4 Mixed Mode (PC-Anbindung)

```

//*****
/** Projekt:   Demo Mixed Mode
/** Datei:    Main.MC
/** Dateityp: MC-1B Quellcode
/**
/** Erstellt am 24.02.99 um 20:01:16 von Mi cha
//*****

// Dieses Programm bildet eine Schnittstelle zwischen
// PC und Steuerung: Der PC sendet Kommandos via
// vorher vereinbarten Codes in einer Variablen, und
// die Steuerung führt dann die entsprechenden Aktionen
// aus. Die Steuercodes werden hier als Konstanten definiert:

DEF_W      0, NICHTS
DEF_W      1, LT_EIN
DEF_W      2, LT_AUS
DEF_W      3, REFERENZ
DEF_W      4, START

DEF_W     -1, FEHLER_ACHSE
DEF_W     -2, FEHLER_AKTION

// Die Aktionen werden in Variable V_AKTION geschrieben,
// ggf. eine Zielposition in V_ZIELPOS. Sobald die
// Steuerung den Job erledigt hat, wird V_AKTION
// auf NICHTS gesetzt, wenn ein Fehler aufgetreten ist,
// wird die entsprechende Fehlerkonstante geschrieben.

DEF_V      200, V_AKTION
DEF_V      201, V_ZIELPOS

// Programmstart. Variablen initialisieren.

LAD_VA     V_AKTION, NICHTS
LAD_VA     V_ZIELPOS, 0

Haupt:
VERG_VA    V_AKTION, NICHTS           // Liegt ein Job an?
LAD_M      M_GLEICH                  // Wir testen, ob die Aktionsvariable 0 ist
SPRINGJ    Haupt                     // Ist sie, wirklich nichts zu tun

VERG_VA    V_AKTION, LT_EIN          // Leistungsteil einschalten?
LAD_M      M_GLEICH                  // Vergleichsergebnis abfragen
GEHUPRJ    U_LT_EIN                  // Ja, Unterprogramm aufrufen

VERG_VA    V_AKTION, LT_AUS          // Leistungsteil ausschalten?
LAD_M      M_GLEICH                  // Vergleichsergebnis abfragen
GEHUPRJ    U_LT_AUS                  // Ja, Unterprogramm aufrufen

VERG_VA    V_AKTION, REFERENZ        // Referenzfahrt durchführen?
LAD_M      M_GLEICH                  // Vergleichsergebnis abfragen
GEHUPRJ    U_REFERENZ                // Ja, Unterprogramm aufrufen

VERG_VA    V_AKTION, START           // Achse starten?
LAD_M      M_GLEICH                  // Vergleichsergebnis abfragen
GEHUPRJ    U_START                   // Ja, Unterprogramm aufrufen

```

```

LAD_M      M_GROESSER           // Wenn ein ungültiger Code für
LAD_VA     V_AKTION, FEHLER_AKTION // die Aktion angegeben wurde, dann
                                                // geben wir einen Fehler zurück.
SPRING     Haupt                // Weiter mit der Hauptschleife

// Unterprogramm U_LT_EIN schaltet das Leistungsteil ein

U_LT_EIN:
LAD_M      M_RFGON_A1           // LT schon eingeschaltet?
LAD_VA     V_AKTION, FEHLER_AKTION // Fehlermeldung zurückgeben
UPRENDJ                                         // Unterprogramm Ende

LAD_M      M_EIN                // BES wieder einschalten
CLRERR     1                    // Etwaige Fehlermeldungen löschen
PWRDRV     1, 1                 // LT einschalten

LAD_VA     V_TIM_1, 50          // Zeitüberwachung 5 Sekunden
EIN_M      M_TIM_1              // Timer starten

U_LT_EIN_WARTEN:
NLAD_M     M_TIM_1              // Solange der Timer läuft...
ODER_M     M_RFGON_A1           // ...und das LT nicht bereit meldet
SPRINGN    U_LT_EIN_WARTEN      // ...warten wir

LAD_M      M_RFGON_A1           // LT jetzt ein?
LAD_VA     V_AKTION, NICHTS      // Ja, erfolgreiche Aktion melden
UPRENDJ                                         // Unterprogramm Ende

LAD_M      M_EIN                // BES wieder einschalten
LAD_VA     V_AKTION, FEHLER_ACHSE // Fehler zurückmelden
UPREND                                           // Unterprogramm Ende

// Unterprogramm U_LT_AUS schaltet das Leistungsteil aus

U_LT_AUS:
NLAD_M     M_RFGON_A1           // LT bereits ausgeschaltet?
LAD_VA     V_AKTION, FEHLER_AKTION // Fehlermeldung zurückgeben
UPRENDJ                                         // Unterprogramm Ende

LAD_M      M_EIN                // BES wieder einschalten
CLRERR     1                    // Etwaige Fehlermeldungen löschen
PWRDRV     1, 0                 // LT ausschalten

LAD_VA     V_TIM_1, 50          // Zeitüberwachung 5 Sekunden
EIN_M      M_TIM_1              // Timer starten

U_LT_AUS_WARTEN:
NLAD_M     M_TIM_1              // Solange der Timer läuft
NODER_M    M_RFGON_A1           // ...oder das LT noch ein ist
SPRINGN    U_LT_AUS_WARTEN      // ...warten wir

NLAD_M     M_RFGON_A1           // LT jetzt aus?
LAD_VA     V_AKTION, NICHTS      // Ja, erfolgreiche Aktion melden
UPRENDJ                                         // Unterprogramm Ende

LAD_M      M_EIN                // BES wieder einschalten
LAD_VA     V_AKTION, FEHLER_ACHSE // Fehler zurückmelden
UPREND                                           // Unterprogramm Ende

```

```

// Unterprogramm U_REFERENZ führt die Referenzfahrt durch

U_REFERENZ:
NLAD_M      M_RFGON_A1          // LT nicht eingeschaltet?
NODER_M     M_INPOS_A1         // ...oder Achse nicht im Stillstand?
LAD_VA      V_AKTION, FEHLER_AKTION // Dann Fehler zurückmelden
UPRENDJ

STHOME      1, 0                // Referenzfahrt starten

LAD_VA      V_TIM_1, 200        // Zeitüberwachung 20 Sekunden
EIN_M       M_TIM_1            // Timer starten

U_REFERENZ_WARTEN:
NLAD_M      M_TIM_1            // Solange der Timer läuft
NODER_M     M_INPOS_A1         // ...und die Achse noch fährt
SPRINGN     U_REFERENZ_WARTEN  // ...warten wir

LAD_M       M_INPOS_A1         // Achse jetzt in Position?
LAD_VA      V_AKTION, NICHTS    // Ja, erfolgreiche Aktion melden
UPRENDJ

LAD_M       M_EIN              // BES wieder einschalten
LAD_VA      V_AKTION, FEHLER_ACHSE // Fehler zurückmelden
UPREND

// Das Unterprogramm U_LT_START startet die Achse auf
// die in V_ZIELPOS angegebene Position

U_START:
NLAD_M      M_RFGON_A1          // LT nicht eingeschaltet?
NODER_M     M_INPOS_A1         // ...oder Achse nicht im Stillstand?
LAD_VA      V_AKTION, FEHLER_AKTION // Dann Fehler zurückmelden
UPRENDJ

STPABS      1, V_ZIELPOS        // Achse auf Position starten

LAD_VA      V_TIM_1, 200        // Zeitüberwachung 20 Sekunden
EIN_M       M_TIM_1            // Timer starten

U_START_WARTEN:
NLAD_M      M_TIM_1            // Solange der Timer läuft
NODER_M     M_INPOS_A1         // ...und die Achse noch fährt
SPRINGN     U_START_WARTEN    // ...warten wir

LAD_M       M_INPOS_A1         // Achse jetzt in Position?
LAD_VA      V_AKTION, NICHTS    // Ja, erfolgreiche Aktion melden
UPRENDJ

LAD_M       M_EIN              // BES wieder einschalten
LAD_VA      V_AKTION, FEHLER_ACHSE // Fehler zurückmelden
UPREND

```

Sie sehen, daß eine PC-Anbindung im Mixed Mode mit der MC-1 Sprache und den MICRO DESIGN PC-Werkzeugen sehr einfach ist. Achten Sie jedoch immer auf mögliche Asynchronitäten, denn natürlich arbeiten der PC und die Steuerung nicht mit der gleichen Geschwindigkeit. Reagieren Sie im PC-Programm stets auf Rückgabewerte aus der Steuerung, und strukturieren Sie den Ablauf so sorgfältig wie möglich.

n Raum für Ihre Notizen

## n Raum für Ihre Notizen

# Anhang A Wahrheitstabelle

In diesem Anhang finden Sie eine Übersicht, wie sich logische Verknüpfungen – UND, ODER, NODER, NUND oder XODER – auf das Ergebnis auswirken.

Verknüpfung	Parameter 1	Parameter 2	Ergebnis
ODER	Aus	Aus	Aus
ODER	Ein	Aus	Ein
ODER	Aus	Ein	Ein
ODER	Ein	Ein	Ein
NODER (Nicht-Oder)	Aus	Aus	Ein
NODER (Nicht-Oder)	Aus	Ein	Aus
NODER (Nicht-Oder)	Ein	Aus	Ein
NODER (Nicht-Oder)	Ein	Ein	Ein
NUND (Nicht-Und)	Aus	Aus	Aus
NUND (Nicht-Und)	Ein	Aus	Ein
NUND (Nicht-Und)	Aus	Ein	Aus
NUND (Nicht-Und)	Ein	Ein	Aus
UND	Aus	Aus	Aus
UND	Ein	Aus	Aus
UND	Aus	Ein	Aus
UND	Ein	Ein	Ein
XODER (Exklusiv-Oder)	Aus	Aus	Aus
XODER (Exklusiv-Oder)	Ein	Aus	Ein
XODER (Exklusiv-Oder)	Aus	Ein	Ein
XODER (Exklusiv-Oder)	Ein	Ein	Aus

n Tabelle 100 – Wahrheitstabelle

# Anhang B Display-Zeichentabelle

Folgender Zeichensatz wird von Displays der MC200 Familie verwendet:

	..0	..1	..2	..3	..4	..5	..6	..7	..8	..9
00.	█	☺	☹	♠	♣	♠	♣	+	█	☺
01.	█	☺	♀	♂	♠	♣	♠	♣	♠	!!
02.	9	6	■	♣	♠	♣	♠	♣	♠	♠
03.	▲	▼		!	"	#	\$	%	&	'
04.	(	)	*	+	,	-	.	/	0	1
05.	2	3	4	5	6	7	8	9	:	;
06.	<	=	>	?	@	A	B	C	D	E
07.	F	G	H	I	J	K	L	M	N	O
08.	P	Q	R	S	T	U	V	W	X	Y
09.	Z	[	\	]	^	_	`	a	b	c
10.	d	e	f	g	h	i	j	k	l	m
11.	n	o	p	q	r	s	t	u	v	w
12.	x	y	z	<		>	~	Δ	∇	ü
13.	é	è	ë	ä	å	ç	ê	ë	è	ï
14.	î	í	Ë	Ä	É	Æ	Ë	Ö	Ö	Ò
15.	Ô	Ù	Û	Ö	Ü	ƒ	£	¥	℞	ƒ
16.	á	í	ó	ú	ñ	ñ	∞	∞	∞	∞
17.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
18.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
19.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
20.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
21.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
22.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
23.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
24.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
25.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

n Abbildung 6 – MC200 Display Zeichentabelle



# Anhang C Benötigte PC-Software

Zur Entwicklung von SPS-Programmen mit MC-1B benötigen Sie die VMC Workbench, die Sie kostenfrei mit einer Bestellung erhalten. Hier wird kurz aufgezeigt, welche Programme der VMC Workbench Sie für welche Aufgaben verwenden können:

## Entwicklung des SPS-Programms

VMC Workbench Studio

## Test und Fehlersuche im SPS-Programm

VMC Workbench Studio

VMC Spy++

MCSave™

## Profibus Fehlerdiagnose

VMC Profibus Spy

## Inbetriebnahme und Parametrierung der Achsen

VMC Workbench Studio

VMC SAT™

VMC Spy++

## Programmierung einer Oberfläche mit dem MC200HT

VMC Workbench Studio

VMC Display Editor

## Fernwartung

VMC Connection Manager

VMC Workbench Studio

VMC Spy++

VMC Display Emulation

## Sonstige Hilfsprogramme

VMC Betriebssystem Assistent

VMC Emergency Recovery System

VMC Modem Assistent

## n Dokumentationen

Zu allen Programmen der VMC Workbench erhalten Sie eine Online-Dokumentation, wenn Sie im Hilfemenü "?" auf den Menüpunkt "Index" klicken, oder einfach die F1-Taste drücken.

## n Installation der Software

Legen Sie die VMC Workbench CD-ROM in das CD-Laufwerk Ihres Computers ein. Das Installationsprogramm startet automatisch. Wählen Sie "VMC installieren", und folgen Sie den Anweisungen auf dem Bildschirm.

# Anhang D Typcodes MC200 Familie

Jedes Modul der MC200 Familie wird durch einen eindeutigen Typcode gekennzeichnet. Folgende Tabelle zeigt eine Übersicht aller gegenwärtig gültigen Typcodes.

Modul	Typcode Hex	Typcode Dezimal
MC200CPU/B	0x80	128
MC200PROFI/B	0x81	129
MC200PROFI/A	0x82	130
MC200CPU/C	0x83	131
MC200CAN/B	0x84	132
MC200CAN/A	0x85	133
Reserviert für zukünftige CPU-Module	0x86	134
Reserviert für zukünftige CPU-Module	0x87	135
MC200E16	0x88	136
MC200A16	0x89	137
MC200A8E8	0x8A	138
MC200AIO	0x8B	139
MC200BED	0x8C	140
MC200HT	0x8D	141
MC200BDZ	0x8E	142
MC200CNT	0x8F	143
MC200MOC	0x90	144
MC200SM2	0x91	145
MC200SM1	0x92	146
Reserviert für zukünftige Achscontroller-Module	0x93	147
Reserviert für zukünftige Achscontroller-Module	0x94	148
Reserviert für zukünftige Achscontroller-Module	0x95	149
Reserviert für zukünftige Achscontroller-Module	0x96	150
Reserviert für zukünftige Achscontroller-Module	0x97	151
MC200SER	0x98	152
MC200SLE	0x99	153
MC200EPR	0x9A	154
MC200CON	0x9B	155

n Tabelle 101 – Typcodes MC200 Familie

# Anhang E Abbildungen und Tabellen

## n Tabellen

n Tabelle 1 – Definitionsbefehle .....	32
n Tabelle 2 – Compileranweisungen .....	32
n Tabelle 3 – Merkerbefehle .....	33
n Tabelle 4 – Ein-/ Ausgangsbefehle .....	35
n Tabelle 5 – Analog I/O .....	35
n Tabelle 6 – Variablenbefehle .....	39
n Tabelle 7 – Variablenvergleichsbefehle .....	39
n Tabelle 8 – Programmablaufbefehle .....	40
n Tabelle 9 – Positionierbefehle .....	42
n Tabelle 10 – Display und Texte .....	43
n Tabelle 11 – Datenkonvertierungsbefehle .....	43
n Tabelle 12 – Serielle Anbindung .....	44
n Tabelle 13 – Systembefehle .....	45
n Tabelle 14 – Makrobefehle .....	45
n Tabelle 15 – Funktionen für FLASH.....	64
n Tabelle 16 – Systemvariablen für FLASH.....	64
n Tabelle 17 – Gültige Werte für die Formatmaske bei LAD_DT.....	72
n Tabelle 18 – Gültige Werte für die Formatmaske bei LAD_DV .....	73
n Tabelle 19 – LAD_MV Übersicht .....	77
n Tabelle 20 – LAD_VM Übersicht .....	84
n Tabelle 21 – Gültige Werte für die Display-Auswahl mit SETDSP .....	104
n Tabelle 22 – Funktionen für SETDSP .....	104
n Tabelle 23 - Lautsprecherfunktionen.....	106
n Tabelle 24 – Gültige Werte für die Display-Auswahl bei SETEDI .....	110
n Tabelle 25 – Funktionen für SETEDI .....	110
n Tabelle 26 – Gültige Tastaturbelegungen .....	111
n Tabelle 27 – Sonderfunktionen für SETFUN.....	112
n Tabelle 28 – Funktionen für SETSER.....	116
n Tabelle 29 – Variablenvergleichsmerker bei VERG_II .....	140
n Tabelle 30 – Variablenvergleichsmerker bei VERG_IV .....	141
n Tabelle 31 – Variablenvergleichsmerker bei VERG_VA.....	142
n Tabelle 32 – Variablenvergleichsmerker bei VERG_VI .....	143
n Tabelle 33 – Variablenvergleichsmerker bei VERG_VV.....	144
n Tabelle 34 – Verknüpfungen für die bedingte Compilierung.....	151
n Tabelle 35 – Merker für Meßfunktion .....	157
n Tabelle 36 – Formatvariable V_ANZMSK .....	163
n Tabelle 37 – Analoge Ein- und Ausgänge .....	164
n Tabelle 38 – Merker für analoge Werteaktualisierung .....	165
n Tabelle 39 – Umsetzung analoge auf digitale Werte .....	165
n Tabelle 40 – Timer-Makros .....	167
n Tabelle 41 – Puffergrößen serielles Modul.....	171

n	Tabelle 42 – Puffergrößen seriellles Modul .....	174
n	Tabelle 43 - Systemparameter .....	178
n	Tabelle 44 – Achsparameter Servomotorcontroller .....	185
n	Tabelle 45 – Achsparameter Schrittmotorcontroller .....	189
n	Tabelle 46 – Parameter seriellles Modul .....	191
n	Tabelle 47 – Systemmerker des Bitergebnisspeichers .....	194
n	Tabelle 48 – Ergebnismerker und -Variablen .....	195
n	Tabelle 49 – Systemvariablen und Parameter interne Systemdaten .....	197
n	Tabelle 50 – Systemmerker und –Variablen SPS-Tasks .....	198
n	Tabelle 51 – Systemmerker und –Variablen RS485 Kommunikation .....	199
n	Tabelle 52 – Systemparameter mit Informationen zu angeschlossenen Modulen .....	201
n	Tabelle 53 – Systemmerker und –Variablen Achsenstatus Achse 1 .....	203
n	Tabelle 54 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 1 .....	203
n	Tabelle 55 – Systemmerker und –Variablen Achsenstatus Achse 2 .....	204
n	Tabelle 56 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 2 .....	204
n	Tabelle 57 – Systemmerker und –Variablen Achsenstatus Achse 3 .....	205
n	Tabelle 58 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 3 .....	205
n	Tabelle 59 – Systemmerker und –Variablen Achsenstatus Achse 4 .....	206
n	Tabelle 60 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 4 .....	206
n	Tabelle 61 – Systemmerker und –Variablen Achsenstatus Achse 5 .....	207
n	Tabelle 62 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 5 .....	207
n	Tabelle 63 – Systemmerker und –Variablen Achsenstatus Achse 6 .....	208
n	Tabelle 64 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 6 .....	208
n	Tabelle 65 – Systemmerker und –Variablen Achsenstatus Achse 7 .....	209
n	Tabelle 66 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 7 .....	209
n	Tabelle 67 – Systemmerker und –Variablen Achsenstatus Achse 8 .....	210
n	Tabelle 68 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 8 .....	210
n	Tabelle 69 – Systemmerker und –Variablen Achsenstatus Achse 9 .....	211
n	Tabelle 70 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 9 .....	211
n	Tabelle 71 – Systemmerker und –Variablen Achsenstatus Achse 10.....	212
n	Tabelle 72 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 10.....	212
n	Tabelle 73 – Systemmerker und –Variablen Achsenstatus Achse 11.....	213
n	Tabelle 74 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 11.....	213
n	Tabelle 75 – Systemmerker und –Variablen Achsenstatus Achse 12.....	214
n	Tabelle 76 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 12.....	214
n	Tabelle 77 – Systemmerker und –Variablen Achsenstatus Achse 13.....	215
n	Tabelle 78 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 13.....	215
n	Tabelle 79 Systemmerker und –Variablen Achsenstatus Achse 14 .....	216
n	Tabelle 80 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 14.....	216
n	Tabelle 81 – Systemmerker und –Variablen Achsenstatus Achse 15.....	217
n	Tabelle 82 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 15.....	217
n	Tabelle 83 – Systemmerker und –Variablen Achsenstatus Achse 16.....	218
n	Tabelle 84 – Erweiterte Parameter zum Lesen des Achsenstatus Achse 16.....	218
n	Tabelle 85 – Systemmerker und -Variablen für die Meßfunktionen.....	219
n	Tabelle 86 – Systemmerker und –Variablen für die SPS-Timer .....	221
n	Tabelle 87 – Blinkmerker .....	222

<b>n</b> Tabelle 88 – Systemmerker und –Variablen zum Zählermodul .....	223
<b>n</b> Tabelle 89 – Systemmerker und –Variablen zu den analogen Ein-/Ausgängen .....	224
<b>n</b> Tabelle 90 – Systemmerker und –Variablen zur EAU-T Emulation .....	225
<b>n</b> Tabelle 91 – Systemvariablen zur Displayprogrammierung .....	226
<b>n</b> Tabelle 92 – Tastenmerker (Gesamtübersicht) .....	227
<b>n</b> Tabelle 93 – Tasten- und LED-Merker für das MC200HT .....	229
<b>n</b> Tabelle 94 – Systemmerker und –Variablen für das elektrische Handrad .....	230
<b>n</b> Tabelle 95 – Systemvariablen und –Merker für die eingebaute serielle Schnittstelle .....	231
<b>n</b> Tabelle 96 – Systemmerker und –Variablen zu den seriellen Erweiterungsmodulen .....	233
<b>n</b> Tabelle 97 – Speicherstruktur MC200CPU (Typen B und C) .....	234
<b>n</b> Tabelle 98 – Speicherstruktur MC200PROFI/B .....	235
<b>n</b> Tabelle 99 – Übersicht UDB-Speicher .....	237
<b>n</b> Tabelle 100 – Wahrheitstabelle .....	255
<b>n</b> Tabelle 101 – Typcodes MC200 Familie .....	258

## **n** Abbildungen

<b>n</b> Abbildung 1 – Typischer Ablauf eines SPS-Programms in MC-1B .....	25
<b>n</b> Abbildung 2 – Aufbau Messfunktion .....	156
<b>n</b> Abbildung 3 – MC200BED (stehend, 20 Tasten) .....	227
<b>n</b> Abbildung 4 – MC200BED (liegend, 36 Tasten) .....	228
<b>n</b> Abbildung 5 – MC200HT .....	228
<b>n</b> Abbildung 6 – MC200 Display Zeichentabelle .....	256

# Anhang F Häufig gestellte Fragen

In diesem Abschnitt des Anhangs beantworten wir einige Fragen zur MC-1B Programmierung und zum MC200 Systems, die uns immer wieder gestellt werden.

## n Zum Thema SPS-Programmierung

? In meinem SPS-Programm schalte ich an einer bestimmten Stelle einen Ausgang ein (oder starte eine Achse), der Befehl wird aber nie ausgeführt. Was mache ich falsch?

! Im Prinzip machen Sie überhaupt nichts falsch, vermutlich sind Sie aber ein Opfer der MC-1B Programmstruktur geworden: Jeder Befehl wird nur dann ausgeführt, wenn das Bitergebnis einschaltet ist. Bei ausgeschaltetem Bitergebnis wird der Befehl ignoriert. Setzen Sie einmal mit dem VMC Workbench Studio einen Haltepunkt auf den Befehl, der nicht ausgeführt wird. Sobald das Programm an dieser Stelle anhält, muß die grüne Anzeige für das Bitergebnis im VMC Workbench Studio eingeschaltet sein. Unser Tip: formulieren Sie die Abfrage, die direkt vor diesem Befehl steht, so um, daß dabei das Bitergebnis eingeschaltet wird, oder verwenden Sie den Merker "Immer ein" (M\_EIN), um das Bitergebnis zwangsweise einzuschalten.

? Während des Programmablaufs geht die Steuerung in Not-Aus, und im VMC Workbench Studio blinkt eine rote Lampe mit dem Text: "Opcode" oder "Stackfehler".

! Wenn Sie diesen Fehler erhalten, dann rufen Sie vermutlich mit einem der GEHUPR-Befehle ein Unterprogramm auf, welches Sie dann nicht korrekt mit einem der UPREND-Befehle beenden, sondern vielleicht mit einem SPRING-Befehl zurück ins Hauptprogramm verzweigen. Dadurch wird das Unterprogramm niemals abgeschlossen, und die Steuerung merkt sich die Daten eines jeden Aufrufs. Wenn dies einige Male wiederholt wird, reicht der interne Speicher der Steuerung für die Verwaltung der Unterprogramme nicht mehr aus, und Sie erhalten eben diesen Fehler. Kontrollieren Sie bitte, ob Sie alle Unterprogramme korrekt mit einem der UPREND-Befehle abschließen.

## n Zum Thema Achsen

? Egal wie hoch ich mit SETVEL die Sollgeschwindigkeit für die Achse setze, sie fährt immer mit der gleichen (langsamen) Geschwindigkeit.

! Vermutlich haben Sie in den Achsparametern eine maximale Geschwindigkeit für die betroffene Achse eingestellt. Überprüfen Sie die Einstellung Ihrer Achsparameter, und versuchen Sie es erneut.

? Egal wie hoch ich mit SETRMP den Wert für die Beschleunigungsrampe setze, es verändert sich nichts im Beschleunigungsverhalten der Achse.

! Es gilt das Gleiche wie bei SETVEL: Vermutlich haben Sie in den Achsparametern eine Begrenzung für die maximale Rampe eingetragen. Erhöhen Sie den entsprechenden Parameter, um höhere Rampenwerte zu ermöglichen.

? Von mir mühsam eingestellte Achsparameter gehen plötzlich ohne besonderen Grund verloren.

- !** Achsparameter werden normalerweise immer nur im batteriegepufferten RAM-Speicher der Steuerung abgelegt. Diese Daten werden jedoch automatisch bei jedem Steuerungsneustart mit den Werten aus dem Flash-EPROM überschrieben. Ein Steuerungsneustart wird dann ausgeführt, wenn Sie z.B. ein SPS-Programm übertragen, die Spannung aus- und wieder einschalten oder ein Betriebssystem übertragen. Speichern Sie deshalb die Achsparameter mit der Inbetriebnahme-Software immer im Flash-EPROM der Steuerung ab.

## n Raum für Ihre Notizen