

SIMATIC

Anweisungsliste (AWL) für S7-300/400

Referenzhandbuch

Diese Dokumentation ist Bestandteil des
Dokumentationspaketes mit der Bestellnummer:
6ES7810-4CA08-8AW1

Ausgabe 03/2006
A5E00706959-01

Vorwort	
Inhaltsverzeichnis	
<hr/>	
Bitverknüpfung	1
<hr/>	
Vergleicher	2
<hr/>	
Umwandler	3
<hr/>	
Zähler	4
<hr/>	
DB-Aufruf	5
<hr/>	
Sprünge	6
<hr/>	
Festpunkt-Funktionen	7
<hr/>	
Gleitpunkt-Funktionen	8
<hr/>	
Laden/Transferieren	9
<hr/>	
Programmsteuerung	10
<hr/>	
Schieben/Rotieren	11
<hr/>	
Zeiten	12
<hr/>	
Wortverknüpfung	13
<hr/>	
Akkumulator-Operationen	14
<hr/>	
Anhang	
<hr/>	
AWL-Operationen Übersicht	A
<hr/>	
Programmierbeispiele	B
<hr/>	
Parameterübergabe	C
<hr/>	
Index	

Sicherheitshinweise

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.



Gefahr

bedeutet, dass Tod oder schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.



Warnung

bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.



Vorsicht

mit Warndreieck bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Vorsicht

ohne Warndreieck bedeutet, dass Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Achtung

bedeutet, dass ein unerwünschtes Ergebnis oder Zustand eintreten kann, wenn der entsprechende Hinweis nicht beachtet wird.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zugehörige Gerät/System darf nur in Verbindung mit dieser Dokumentation eingerichtet und betrieben werden. Inbetriebsetzung und Betrieb eines Gerätes/Systems dürfen nur von **qualifiziertem Personal** vorgenommen werden. Qualifiziertes Personal im Sinne der sicherheitstechnischen Hinweise dieser Dokumentation sind Personen, die die Berechtigung haben, Geräte, Systeme und Stromkreise gemäß den Standards der Sicherheitstechnik in Betrieb zu nehmen, zu erden und zu kennzeichnen.

Bestimmungsgemäßer Gebrauch

Beachten Sie Folgendes:



Warnung

Das Gerät darf nur für die im Katalog und in der technischen Beschreibung vorgesehenen Einsatzfälle und nur in Verbindung mit von Siemens empfohlenen bzw. zugelassenen Fremdgeräten und -komponenten verwendet werden. Der einwandfreie und sichere Betrieb des Produktes setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung und Montage sowie sorgfältige Bedienung und Instandhaltung voraus.

Marken

Alle mit dem Schutzrechtsvermerk © gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

Haftungsausschluss

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten.

Vorwort

Zweck des Handbuchs

Dieses Handbuch unterstützt Sie bei der Erstellung von Anwenderprogrammen in der Programmiersprache AWL.

Es beschreibt die Sprachelemente der Programmiersprache AWL, ihre Syntax und Funktionsweise.

Erforderliche Grundkenntnisse

Dieses Handbuch richtet sich an Programmierer von S7-Programmen, Inbetriebsetzer und Servicepersonal.

Zum Verständnis des Handbuchs sind allgemeine Kenntnisse auf dem Gebiet der Automatisierungstechnik erforderlich.

Außerdem werden Kenntnisse über die Verwendung von Computern oder PC-ähnlichen Arbeitsmitteln (z. B. Programmiergeräten) unter den Betriebssystemen MS Windows 2000 Professional, MS Windows XP Professional oder MS Windows Server 2003 vorausgesetzt.

Gültigkeitsbereich des Handbuchs

Das Handbuch ist gültig für die Programmiersoftware STEP 7 ab Version 5.4.

Normerfüllung nach IEC 1131-3

AWL entspricht der in der Norm DIN EN-61131-3 (int. IEC 1131-3) festgelegten Sprache "Anweisungsliste" (engl. Instruction List), wobei hinsichtlich der Operationen wesentliche Unterschiede bestehen. Genaue Aussagen zur Normerfüllung finden Sie in der Normerfüllungstabelle in der NORM.TAB-Datei von STEP 7.

Dokumentationspakete zu STEP 7

Das vorliegende Handbuch zu AWL setzt theoretische Kenntnisse über S7-Programme voraus, die Sie in der Online-Hilfe zu STEP 7 nachlesen können. Da die Sprachpakete auf der Basissoftware STEP 7 aufsetzen, sollten Sie bereits Kenntnisse im Umgang mit der Basissoftware STEP 7 und deren Dokumentation haben.

Dieses Handbuch ist Bestandteil des Dokumentationspaketes "STEP 7 Referenzwissen".

Die folgende Tabelle zeigt die Dokumentation zu STEP 7 im Überblick:

Handbücher	Zweck	Bestellnummer
STEP 7-Grundwissen mit <ul style="list-style-type: none"> • Erste Schritte und Übungen mit STEP 7 • Programmieren mit STEP 7 • Hardware konfigurieren und Verbindungen projektieren mit STEP 7 • Von S5 nach S7, Umsteigerhandbuch 	Das Grundwissen für technisches Personal, das das Vorgehen zur Realisierung von Steuerungsaufgaben mit STEP 7 und S7-300/400 beschreibt.	6ES7810-4CA08-8AW1
STEP 7-Referenzwissen mit <ul style="list-style-type: none"> • Handbücher KOP/FUP/AWL für S7-300/400 • Standard- und Systemfunktionen für S7-300/400 Band 1 und Band 2 	Das Referenzwissen zum Nachschlagen, das die Programmiersprachen KOP, FUP und AWL sowie Standard- und Systemfunktionen ergänzend zum STEP 7-Grundwissen beschreibt.	6ES7810-4CA08-8AW1

Online-Hilfen	Zweck	Bestellnummer
Hilfe zu STEP 7	Das Grundwissen zum Programmieren und Hardware konfigurieren mit STEP 7 als Online-Hilfe	Bestandteil des Softwarepaketes STEP 7
Referenzhilfen zu AWL/KOP/FUP Referenzhilfe zu SFBs/SFCs Referenzhilfe zu Organisationsbausteinen	Kontextsensitives Referenzwissen	Bestandteil des Softwarepaketes STEP 7

Online-Hilfe

Ergänzend zum Handbuch erhalten Sie bei der Nutzung der Software detaillierte Unterstützung durch die in die Software integrierte Online-Hilfe.

Auf die Inhalte der Online-Hilfe können Sie wie folgt zugreifen:

- Kontext-sensitive Hilfe zum markierten Objekt über Menübefehl **Hilfe > Hilfe zum Kontext** über Funktionstaste F1 oder über Fragezeichen in der Funktionsleiste.
- Hilfe zu STEP 7 über den Menübefehl **Hilfe > Hilfethemen** oder die Schaltfläche "Hilfe zu STEP 7" im Hilfefenster der kontext-sensitiven Hilfe.
- Glossar für alle STEP 7-Applikationen über die Schaltfläche "**Glossar**".

Wenn Sie Informationen der Online-Hilfe lieber in gedruckter Form lesen möchten, können Sie einzelne Hilfethemen, Bücher oder die gesamte Hilfe auch ausdrucken.

Dieses Handbuch ist ein Auszug der "Hilfe zu AWL". Aufgrund der identischen Gliederungsstruktur von Handbuch und Online-Hilfe können Sie bequem zwischen Handbuch und Online-Hilfe wechseln.

Weitere Unterstützung

Bei Fragen zur Nutzung der im Handbuch beschriebenen Produkte, die Sie hier nicht beantwortet finden, wenden Sie sich bitte an Ihren Siemens-Ansprechpartner in den für Sie zuständigen Vertretungen und Geschäftsstellen.

Ihren Ansprechpartner finden Sie unter:

<http://www.siemens.com/automation/partner>

Den Wegweiser zum Angebot an technischen Dokumentationen für die einzelnen SIMATIC Produkte und Systeme finden Sie unter:

<http://www.siemens.de/simatic-tech-doku-portal>

Den Online-Katalog und das Online-Bestellsystem finden Sie unter:

<http://mall.automation.siemens.com/>

Trainingscenter

Um Ihnen den Einstieg in das Automatisierungssystem SIMATIC S7 zu erleichtern, bieten wir entsprechende Kurse an. Wenden Sie sich bitte an Ihr regionales Trainingscenter oder an das zentrale Trainingscenter in D 90327 Nürnberg.

Telefon: +49 (911) 895-3200.

Internet: <http://www.sitrain.com>

Technical Support

Sie erreichen den Technical Support für alle A&D-Produkte

Über das Web-Formular für den Support Request
<http://www.siemens.de/automation/support-request>

Telefon: + 49 180 5050 222

Fax: + 49 180 5050 223

Weitere Informationen zu unserem Technical Support finden Sie im Internet unter
<http://www.siemens.de/automation/service>

Service & Support im Internet

Zusätzlich zu unserem Dokumentations-Angebot bieten wir Ihnen im Internet unser komplettes Wissen online an.

<http://www.siemens.com/automation/service&support>

Dort finden Sie:

- den Newsletter, der Sie ständig mit den aktuellsten Informationen zu Ihren Produkten versorgt.
- die für Sie richtigen Dokumente über unsere Suche in Service & Support.
- ein Forum, in welchem Anwender und Spezialisten weltweit Erfahrungen austauschen.
- Ihren Ansprechpartner für Automation & Drives vor Ort.
- Informationen über Vor-Ort Service, Reparaturen, Ersatzteile. Vieles mehr steht für Sie unter dem Begriff "Leistungen" bereit.

Inhaltsverzeichnis

1	Bitverknüpfung.....	1-1
1.1	Bitverknüpfungsoperationen Übersicht.....	1-1
1.2	U Und.....	1-3
1.3	UN Und Nicht.....	1-4
1.4	O Oder.....	1-5
1.5	ON Oder Nicht.....	1-6
1.6	X Exklusiv Oder.....	1-7
1.7	XN Exklusiv Oder Nicht.....	1-8
1.8	O Und vor Oder.....	1-9
1.9	U(Und mit Verzweigung.....	1-10
1.10	UN(Und Nicht mit Verzweigung.....	1-11
1.11	O(Oder mit Verzweigung.....	1-12
1.12	ON(Oder Nicht mit Verzweigung.....	1-13
1.13	X(Exklusiv Oder mit Verzweigung.....	1-14
1.14	XN(Exklusiv Oder Nicht mit Verzweigung.....	1-15
1.15) Verzweigung schließen.....	1-16
1.16	= Zuweisung.....	1-18
1.17	R Rücksetze.....	1-19
1.18	S Setze.....	1-20
1.19	NOT Negiere VKE.....	1-21
1.20	SET Setze VKE (=1).....	1-22
1.21	CLR Rücksetze VKE (=0).....	1-23
1.22	SAVE Sichere VKE im BIE-Bit.....	1-24
1.23	FN Flanke Negativ.....	1-25
1.24	FP Flanke Positiv.....	1-27
2	Vergleicher.....	2-1
2.1	Vergleichsoperationen Übersicht.....	2-1
2.2	? I Vergleiche Ganzzahlen (16 Bit).....	2-2
2.3	? D Vergleiche Ganzzahlen (32 Bit).....	2-3
2.4	? R Vergleiche Gleitpunktzahlen (32 Bit).....	2-4
3	Umwandler.....	3-1
3.1	Umwandlungsoperationen Übersicht.....	3-1
3.2	BTI BCD wandeln in Ganzzahl (16 Bit).....	3-2
3.3	ITB Ganzzahl (16 Bit) wandeln in BCD.....	3-3
3.4	BTD BCD wandeln in Ganzzahl (32 Bit).....	3-4
3.5	ITD Ganzzahl (16 Bit) wandeln in Ganzzahl (32 Bit).....	3-5
3.6	DTB Ganzzahl (32 Bit) wandeln in BCD.....	3-6
3.7	DTR Ganzzahl (32 Bit) wandeln in Gleitpunktzahl (32 Bit, IEEE-FP).....	3-7
3.8	INVI 1-Komplement Ganzzahl (16 Bit).....	3-8
3.9	INVD 1-Komplement Ganzzahl (32 Bit).....	3-9
3.10	NEGI 2-Komplement Ganzzahl (16 Bit).....	3-10
3.11	NEGD 2-Komplement Ganzzahl (32 Bit).....	3-11
3.12	NEGR Negiere Gleitpunktzahl.....	3-12
3.13	TAW Tausche Reihenfolge der Bytes im AKKU 1-L (16 Bit).....	3-13
3.14	TAD Tausche Reihenfolge der Bytes im AKKU 1 (32 Bit).....	3-14

3.15	RND	Runden einer Gleitpunktzahl zur Ganzzahl	3-15
3.16	TRUNC	Runden einer Gleitpunktzahl durch Abschneiden.....	3-16
3.17	RND+	Runden einer Gleitpunktzahl zur nächsthöheren Ganzzahl.....	3-17
3.18	RND-	Runden einer Gleitpunktzahl zur nächstniederen Ganzzahl	3-18
4	Zähler		4-1
4.1		Zähloperationen Übersicht	4-1
4.2	FR	Freigabe Zähler.....	4-2
4.3	L	Lade aktuellen Zählwert als Ganzzahl in AKKU 1	4-3
4.4	LC	Lade aktuellen Zählwert als BCD in AKKU 1	4-5
4.5	R	Rücksetze Zähler	4-7
4.6	S	Setze Zählerstartwert	4-8
4.7	ZV	Zählen vorwärts.....	4-9
4.8	ZR	Zählen rückwärts.....	4-10
5	DB-Aufruf		5-1
5.1		Datenbausteinoperationen Übersicht	5-1
5.2	AUF	Datenbaustein öffnen	5-2
5.3	TDB	Tausche Global-DB und Instanz-DB	5-3
5.4	L DBLG	Lade Länge Global-DB in AKKU 1	5-4
5.5	L DBNO	Lade Nummer Global-DB in AKKU 1	5-5
5.6	L DILG	Lade Länge Instanz-DB in AKKU 1	5-6
5.7	L DINO	Lade Nummer Instanz-DB in AKKU 1	5-7
6	Sprünge		6-1
6.1		Sprungoperationen Übersicht.....	6-1
6.2	SPA	Springe absolut.....	6-3
6.3	SPL	Springleiste	6-4
6.4	SPB	Springe, wenn VKE = 1	6-6
6.5	SPBN	Springe, wenn VKE = 0.....	6-7
6.6	SPBB	Springe, wenn VKE = 1 und rette VKE ins BIE.....	6-8
6.7	SPBNB	Springe, wenn VKE = 0 und rette VKE ins BIE	6-9
6.8	SPBI	Springe, wenn BIE = 1.....	6-10
6.9	SPBIN	Springe, wenn BIE = 0.....	6-11
6.10	SPO	Springe, wenn OV = 1	6-12
6.11	SPS	Springe, wenn OS = 1	6-14
6.12	SPZ	Springe, wenn Ergebnis = 0	6-16
6.13	SPN	Springe, wenn Ergebnis <> 0.....	6-17
6.14	SPP	Springe, wenn Ergebnis > 0	6-18
6.15	SPM	Springe, wenn Ergebnis < 0.....	6-19
6.16	SPPZ	Springe, wenn Ergebnis >= 0.....	6-20
6.17	SPMZ	Springe, wenn Ergebnis <= 0	6-21
6.18	SPU	Springe, wenn Ergebnis ungültig	6-22
6.19	LOOP	Programmschleife.....	6-24

7	Festpunkt-Funktionen	7-1
7.1	Festpunkt-Funktionen Übersicht.....	7-1
7.2	Auswerten der Bits im Statuswort bei Festpunkt-Funktionen.....	7-2
7.3	+I Addiere AKKU 1 und 2 als Ganzzahl (16 Bit).....	7-3
7.4	-I Subtrahiere AKKU 1 von 2 als Ganzzahl (16 Bit).....	7-4
7.5	*I Multipliziere AKKU 1 und 2 als Ganzzahl (16 Bit).....	7-5
7.6	/I Dividiere AKKU 2 durch 1 als Ganzzahl (16 Bit)	7-6
7.7	+ Addiere Ganzzahlkonstante (16, 32 Bit).....	7-8
7.8	+D Addiere AKKU 1 und 2 als Ganzzahl (32 Bit).....	7-10
7.9	-D Subtrahiere AKKU 1 von 2 als Ganzzahl (32 Bit).....	7-11
7.10	*D Multipliziere AKKU 1 und 2 als Ganzzahl (32 Bit)	7-12
7.11	/D Dividiere AKKU 2 durch 1 als Ganzzahl (32 Bit).....	7-13
7.12	MOD Divisionsrest Ganzzahl (32 Bit).....	7-15
8	Gleitpunkt-Funktionen.....	8-1
8.1	Gleitpunkt-Funktionen Übersicht	8-1
8.2	Auswerten der Bits im Statuswort bei Gleitpunkt-Funktionen	8-2
8.3	Grundoperationen.....	8-3
8.3.1	+R Addiere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)	8-3
8.3.2	-R Subtrahiere AKKU 1 von 2 als Gleitpunktzahl (32 Bit)	8-5
8.3.3	*R Multipliziere AKKU 1 und 2 als Gleitpunktzahl (32 Bit).....	8-7
8.3.4	/R Dividiere AKKU 2 durch 1 als Gleitpunktzahl (32 Bit).....	8-8
8.3.5	ABS Absolutwert einer Gleitpunktzahl (32 Bit, IEEE-FP)	8-9
8.4	Erweiterte Operationen.....	8-10
8.4.1	SQR Bilden des Quadrats einer Gleitpunktzahl (32 Bit).....	8-10
8.4.2	SQRT Bilden der Quadratwurzel einer Gleitpunktzahl (32 Bit)	8-11
8.4.3	EXP Bilden des Exponentialwerts einer Gleitpunktzahl (32 Bit)	8-12
8.4.4	LN Bilden des natürlichen Logarithmus einer Gleitpunktzahl (32 Bit)	8-13
8.4.5	SIN Bilden des Sinus eines Winkels als Gleitpunktzahlen (32 Bit)	8-14
8.4.6	COS Bilden des Cosinus eines Winkels als Gleitpunktzahlen (32 Bit)	8-15
8.4.7	TAN Bilden des Tangens eines Winkels als Gleitpunktzahlen (32 Bit)	8-16
8.4.8	ASIN Bilden des Arcussinus einer Gleitpunktzahl (32 Bit).....	8-17
8.4.9	ACOS Bilden des Arcuscossinus einer Gleitpunktzahl (32 Bit).....	8-18
8.4.10	ATAN Bilden des Arcustangens einer Gleitpunktzahl (32 Bit).....	8-19
9	Laden/Transferieren.....	9-1
9.1	Lade- und Transferoperationen Übersicht.....	9-1
9.2	L Lade.....	9-2
9.3	L STW Lade Statuswort in AKKU 1	9-4
9.4	LAR1 Lade Adreßregister 1 mit Inhalt von AKKU 1	9-5
9.5	LAR1 <D> Lade Adreßregister 1 mit Pointer (32 Bit-Format)	9-6
9.6	LAR1 AR2 Lade Adreßregister 1 mit Inhalt von Adressregister 2.....	9-7
9.7	LAR2 Lade Adreßregister 2 mit Inhalt von AKKU 1	9-7
9.8	LAR2 <D> Lade Adreßregister 2 mit Ganzzahl (32 Bit)	9-8
9.9	T Transferiere	9-9
9.10	T STW Transferiere AKKU 1 in Statuswort	9-10
9.11	TAR Tausche Adreßregister 1 mit 2.....	9-11
9.12	TAR1 Transferiere Adreßregister 1 in AKKU 1	9-11
9.13	TAR1 <D> Transferiere Adreßregister 1 nach Zieladresse (32-Bit-Pointer)	9-12
9.14	TAR1 AR2 Transferiere Adreßregister 1 in Adreßregister 2	9-13
9.15	TAR2 Transferiere Adreßregister 2 in AKKU 1	9-13
9.16	TAR2 <D> Transferiere Adreßregister 2 nach Zieladresse (32-Bit-Pointer)	9-14

10	Programmsteuerung	10-1
10.1	Programmsteuerungsoperationen Übersicht	10-1
10.2	BE Bausteinende	10-2
10.3	BEB Bausteinende bedingt	10-3
10.4	BEA Bausteinende absolut	10-4
10.5	CALL Bausteinaufruf	10-5
10.6	FB aufrufen	10-8
10.7	FC aufrufen	10-10
10.8	SFB aufrufen	10-12
10.9	SFC aufrufen	10-14
10.10	Multiinstanz aufrufen	10-16
10.11	Baustein aus einer Bibliothek aufrufen	10-16
10.12	CC Bedingter Bausteinaufruf	10-17
10.13	UC Unbedingter Bausteinaufruf	10-18
10.14	Das Master Control Relay	10-19
10.15	Wichtige Hinweise zur Benutzung der MCR-Funktionalität	10-21
10.16	MCR(Sichere VKE im MCR-Stack, Beginn MCR-Bereich	10-22
10.17)MCR Beende MCR-Bereich	10-24
10.18	MCRA Aktiviere MCR-Bereich	10-25
10.19	MCRD Deaktiviere MCR-Bereich	10-26
11	Schieben/Rotieren	11-1
11.1	Schiebeoperationen	11-1
11.1.1	Schiebeoperationen Übersicht	11-1
11.1.2	SSI Schiebe Vorzeichen rechts Ganzzahl (16 Bit)	11-2
11.1.3	SSD Schiebe Vorzeichen rechts Ganzzahl (32 Bit)	11-4
11.1.4	SLW Schiebe links Wort (16 Bit)	11-6
11.1.5	SRW Schiebe rechts Wort (16 Bit)	11-8
11.1.6	SLD Schiebe links Doppelwort (32 Bit)	11-10
11.1.7	SRD Schiebe rechts Doppelwort (32 Bit)	11-12
11.2	Rotieroperationen	11-14
11.2.1	Rotieroperationen Übersicht	11-14
11.2.2	RLD Rotiere links Doppelwort (32 Bit)	11-14
11.2.3	RRD Rotiere rechts Doppelwort (32 Bit)	11-16
11.2.4	RLDA Rotiere Akku 1 links über A1-Anzeige (32 Bit)	11-18
11.2.5	RRDA Rotiere Akku 1 rechts über A1-Anzeige (32 Bit)	11-19
12	Zeiten	12-1
12.1	Zeitoperationen Übersicht	12-1
12.2	Speicherbereiche und Komponenten einer Zeit	12-2
12.3	FR Freigabe Timer	12-5
12.4	L Lade aktuellen Zeitwert als Ganzzahl in AKKU 1	12-7
12.5	LC Lade aktuellen Zeitwert als BCD in AKKU 1	12-9
12.6	R Rücksetze Timer	12-11
12.7	SI Zeit als Impuls	12-12
12.8	SV Zeit als verlängerter Impuls	12-14
12.9	SE Zeit als Einschaltverzögerung	12-16
12.10	SS Zeit als speichernde Einschaltverzögerung	12-18
12.11	SA Zeit als Ausschaltverzögerung	12-20

13	Wortverknüpfung	13-1
13.1	Wortverknüpfungsoperationen Übersicht	13-1
13.2	UW UND-Wort (16 Bit).....	13-2
13.3	OW ODER-Wort (16 Bit).....	13-4
13.4	XOW EXKLUSIV-ODER-Wort (16 Bit)	13-6
13.5	UD UND-Doppelwort (32 Bit).....	13-8
13.6	OD ODER-Doppelwort (32 Bit).....	13-10
13.7	XOD EXKLUSIV-ODER-Doppelwort (32 Bit).....	13-12
14	Akkumulator-Operationen.....	14-1
14.1	Akkumulatoroperationen Übersicht	14-1
14.2	TAK Tausche AKKU 1 mit AKKU 2	14-2
14.3	PUSH CPU mit zwei Akkus	14-3
14.4	PUSH CPU mit vier Akkus.....	14-4
14.5	POP CPU mit zwei Akkus.....	14-5
14.6	POP CPU mit vier Akkus	14-6
14.7	ENT Enter AKKU-Stack	14-7
14.8	LEAVE Leave AKKU-Stack	14-8
14.9	INC Inkrementiere AKKU 1-L-L	14-8
14.10	DEC Dekrementiere AKKU 1-L-L	14-10
14.11	+AR1 Addiere AKKU 1 zum Adreßregister 1.....	14-11
14.12	+AR2 Addiere AKKU 1 zum Adreßregister 2.....	14-12
14.13	BLD Bildbefehl (Nulloperation)	14-14
14.14	NOP 0 Nulloperation.....	14-15
14.15	NOP 1 Nulloperation.....	14-15
A	AWL-Operationen Übersicht.....	A-1
A.1	AWL-Operationen sortiert nach deutscher Mnemonik (SIMATIC)	A-1
A.2	AWL-Operationen sortiert nach englischer Mnemonik (International)	A-6
B	Programmierbeispiele	B-1
B.1	Programmierbeispiele Übersicht	B-1
B.2	Bitverknüpfungsoperationen Beispiel	B-2
B.3	Zeitoperationen Beispiel	B-7
B.4	Zähl- und Vergleichsoperationen Beispiel.....	B-10
B.5	Arithmetische Operationen mit Ganzzahlen Beispiel	B-12
B.6	Wortverknüpfungsoperationen Beispiel.....	B-13
C	Parameterübergabe	C-1
Index	Index-1

1 Bitverknüpfung

1.1 Bitverknüpfungsoperationen Übersicht

Beschreibung

Bitverknüpfungsoperationen arbeiten mit den Zahlen "1" und "0". Diese Zahlen bilden die Basis des Dualsystems und werden "Binärziffern" oder kurz "Bits" genannt. Im Zusammenhang mit U, O, XO und Ausgängen steht eine "1" für "logisch JA" und eine "0" für "logisch NEIN".

Die Bitverknüpfungsoperationen interpretieren die Signalzustände "1" und "0" und verknüpfen sie entsprechend der Booleschen Logik. Die Verknüpfungen liefern ein Ergebnis von "1" oder "0", das sogenannte Verknüpfungsergebnis (VKE).

Für Verknüpfungsoperationen mit Bitoperanden gibt es folgende Grundoperationen:

- U Und
- UN Und Nicht
- O Oder
- ON Oder Nicht
- X Exklusiv Oder
- XN Exklusiv Oder Nicht

Mit folgenden Operationen können Sie Klammerausdrücke bilden:

- U(Und mit Verzweigung,
- UN(Und Nicht mit Verzweigung
- O(Oder mit Verzweigung,
- ON(Oder Nicht mit Verzweigung
- X(Exklusiv Oder mit Verzweigung
- XN(Exklusiv Oder Nicht mit Verzweigung
-) Verzweigung schließen

Mit folgenden Operationen können Sie eine Verknüpfungskette abschließen:

- = Zuweisung
- R Rücksetze
- S Setze

Mit folgenden Operationen können Sie das VKE verändern:

- NOT Negiere VKE
- SET Setze VKE (=1)
- CLR Rücksetze VKE (=0)
- SAVE Sichere VKE im BIE-Bit

Folgende Operationen reagieren auf einen Wechsel im VKE:

- FN Flanke Negativ (1 -> 0)
- FP Flanke Positiv (0 -> 1)

1.2 U Und

Format

U <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D, T, Z

Beschreibung

U fragt das adressierte Bit auf den Signalzustand "1" ab und führt eine UND-Verknüpfung des Abfrageergebnisses mit dem VKE durch.

Mit der Operation **UND** können Sie auch direkt das Statuswort abfragen. Verwenden Sie hierzu die folgenden Operanden: ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BIE.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	x	x	x	1

Beispiel

AWL-Programm	Relais-Schaltplan		
	Stromschiene		
U E 1.0	E 1.0 Signalzustand 1		Schließer
U E 1.1	E 1.1 Signalzustand 1		Schließer
= A 4.0	A 4.0 Signalzustand 1		Spule
Zeigt geschlossenen Schalter			

1.3 UN Und Nicht

Format

UN <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D, T, Z

Beschreibung

UN fragt das adressierte Bit auf den Signalzustand "0" ab und führt eine UND-Verknüpfung des Abfrageergebnisses mit dem VKE durch.

Mit der Operation **UND NICHT** können Sie auch direkt das Statuswort abfragen. Verwenden Sie hierzu die folgenden Operanden: ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BIE.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	x	x	x	1

Beispiel

AWL-Programm		Relais-Schaltplan	
			Stromschiene
U	E 1.0	E 1.0 Signalzustand 0	Schließer
UN	E 1.1	E 1.1 Signalzustand 1	Öffner
=	A 4.0	A 4.0 Signalzustand 0	Spule

1.4 O Oder

Format

O <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D, T, Z

Beschreibung

O fragt das adressierte Bit auf den Signalzustand "1" ab und führt eine ODER-Verknüpfung des Abfrageergebnisses mit dem VKE durch.

Mit der Operation **ODER** können Sie auch direkt das Statuswort abfragen. Verwenden Sie hierzu die folgenden Operanden: ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BIE.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	x	1

Beispiel

AWL-Programm	Relais-Schaltplan
O E 1.0	E 1.0 Signalzustand 1 Schließer
O E 1.1	E 1.1 Signalzustand 0 Schließer
= A 4.0	A 4.0 Signalzustand 1 Spule
	<p>▼ Zeigt geschlossenen Schalter an</p>

1.5 ON Oder Nicht

Format

ON <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D, T, Z

Beschreibung

ON fragt das adressierte Bit auf den Signalzustand "0" ab und führt eine ODER-Verknüpfung des Abfrageergebnisses mit dem VKE durch.

Mit der Operation **ODER NICHT** können Sie auch direkt das Statuswort abfragen. Verwenden Sie hierzu die folgenden Operanden: ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BIE.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	x	1

Beispiel

AWL-Programm		Relais-Schaltplan	
			Stromschiene
O	E 1.0	E 1.0 Signalzustand 0	Schließer
ON	E 1.1	E 1.1 Signalzustand 1	Öffner
=	A 4.0	A 4.0 Signalzustand 1	Spule

1.6 X Exklusiv Oder

Format

X <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D, T, Z

Beschreibung

X fragt das adressierte Bit auf den Signalzustand "1" ab und führt eine EXKLUSIV ODER-Verknüpfung des Abfrageergebnisses mit dem VKE durch.

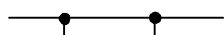

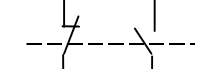

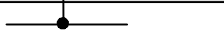
Sie können die Exklusiv-ODER-Funktion auch mehrfach nacheinander anwenden. Dann ist das gemeinsame Verknüpfungsergebnis "1", wenn eine ungerade Anzahl der abgefragten Operanden das Abfrageergebnis "1" liefert.

Mit der Operation **EXKLUSIV ODER** können Sie auch direkt das Statuswort abfragen. Verwenden Sie hierzu die folgenden Operanden: ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BIE.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	x	1

Beispiel

AWL-Programm		Relais-Schaltplan	
		Stromschiene	
X	E 1.0	Kontakt E 1.0	
X	E 1.1	Kontakt E 1.1	
=	A 4.0	A 4.0 Spule	
			

1.7 XN Exklusiv Oder Nicht

Format

XN <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D, T, Z

Beschreibung

XN fragt das adressierte Bit auf den Signalzustand "0" ab und führt eine EXKLUSIV ODER-Verknüpfung des Abfrageergebnisses mit dem VKE durch.

Mit der Operation **EXKLUSIV ODER NICHT** können Sie auch direkt das Statuswort abfragen. Verwenden Sie hierzu die folgenden Operanden: ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BIE.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	x	1

Beispiel

AWL-Programm		Relais-Schaltplan	
		Stromschiene	
X	E 1.0	Kontakt E 1.0	
XN	E 1.1	Kontakt E 1.1	
=	A 4.0	A 4.0 Spule	

1.8 O Und vor Oder

Format

O

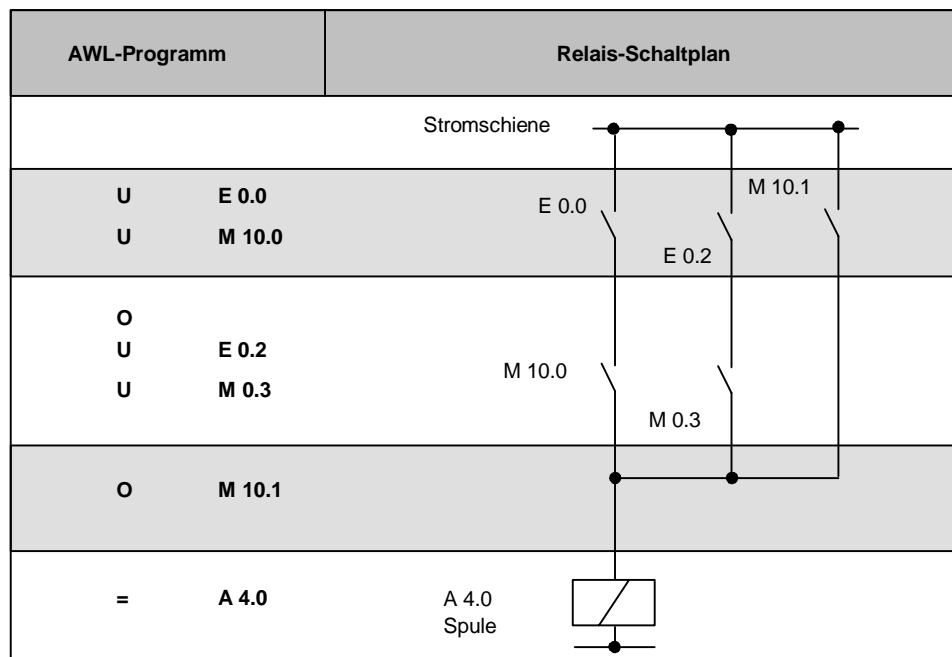
Beschreibung

Die Operation **O** führt nach der Regel UND vor ODER die Verknüpfung ODER auf UND-Verknüpfungen aus.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	x	1	-	x

Beispiel



1.9 U(Und mit Verzweigung

Format

U(

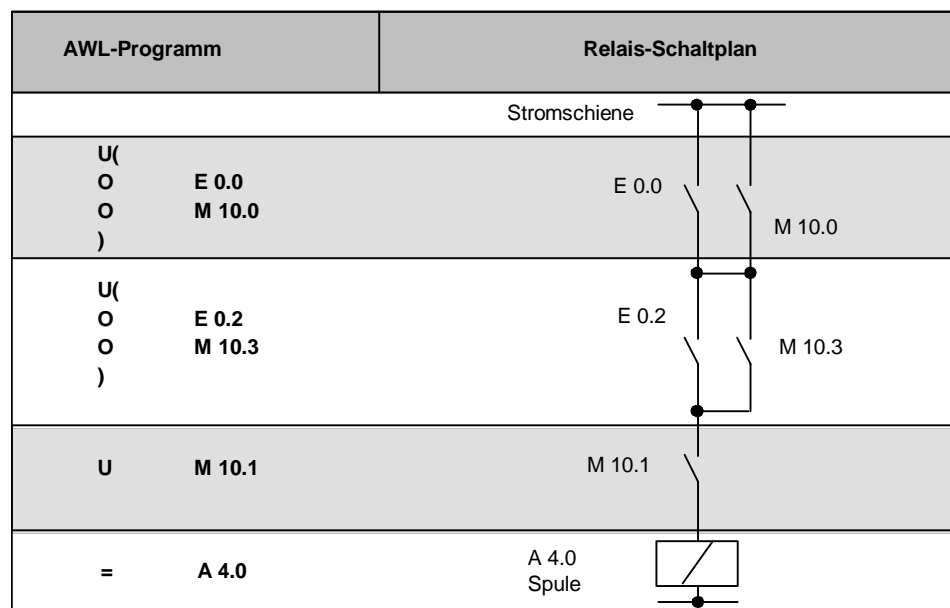
Beschreibung

U((UND mit Verzweigung) speichert die Bits VKE und OR sowie eine Operationskennung im Klammerstack. Der Klammerstack kann maximal 7 Einträge enthalten.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

Beispiel



1.10 UN(Und Nicht mit Verzweigung

Format

UN(

Beschreibung

UN((UND NICHT mit Verzweigung) speichert die Bits VKE und OR sowie eine Operationskennung im Klammerstack. Der Klammerstack kann maximal 7 Einträge enthalten.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

1.11 O(Oder mit Verzweigung

Format

O(

Beschreibung

O((ODER mit Verzweigung) speichert die Bits VKE und OR sowie eine Operationskennung im Klammerstack. Der Klammerstack kann maximal 7 Einträge enthalten.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

1.12 ON(Oder Nicht mit Verzweigung

Format

ON(

Beschreibung

ON((ODER NICHT mit Verzweigung) speichert die Bits VKE und OR sowie eine Operationskennung im Klammerstack. Der Klammerstack kann maximal 7 Einträge enthalten.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

1.13 X(Exklusiv Oder mit Verzweigung

Format

X(

Beschreibung

X((EXKLUSIV ODER mit Verzweigung) speichert die Bits VKE und OR sowie eine Operationskennung im Klammerstack. Der Klammerstack kann maximal 7 Einträge enthalten.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

1.14 XN(Exklusiv Oder Nicht mit Verzweigung

Format

XN(

Beschreibung

XN((EXKLUSIV ODER NICHT mit Verzweigung) speichert die Bits VKE und OR sowie eine Operationskennung im Klammerstack. Der Klammerstack kann maximal 7 Einträge enthalten.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

1.15) Verzweigung schließen

Format

)

Beschreibung

) (Verzweigung schließen) löscht einen Eintrag aus dem Klammerstack, stellt das Bit OR wieder her, verknüpft das im Stackeintrag enthaltene VKE mit dem aktuellen VKE entsprechend der Operationskennung und weist das Ergebnis dem VKE zu. Handelt es sich bei der Operationskennung um UND oder UND NICHT, wird zusätzlich das OR-Bit berücksichtigt.

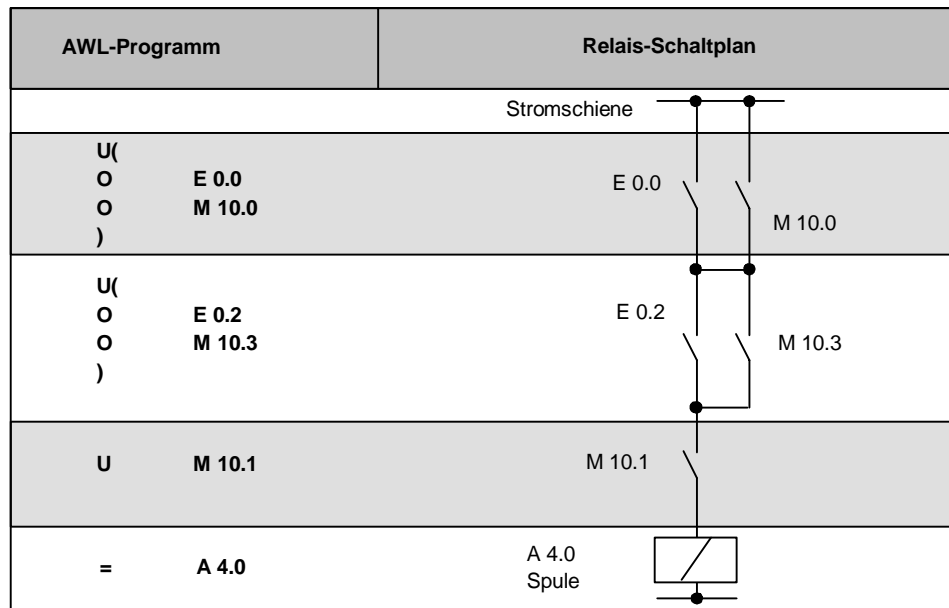
Operationen zum Öffnen von Verzweigungen:

- U(Und mit Verzweigung
- UN(Und Nicht mit Verzweigung
- O(Oder mit Verzweigung
- ON(Oder Nicht mit Verzweigung
- X(Exklusiv Oder mit Verzweigung
- XN(Exklusiv Oder Nicht mit Verzweigung

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	x	1	x	1

Beispiel



1.16 = Zuweisung

Format

= <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D

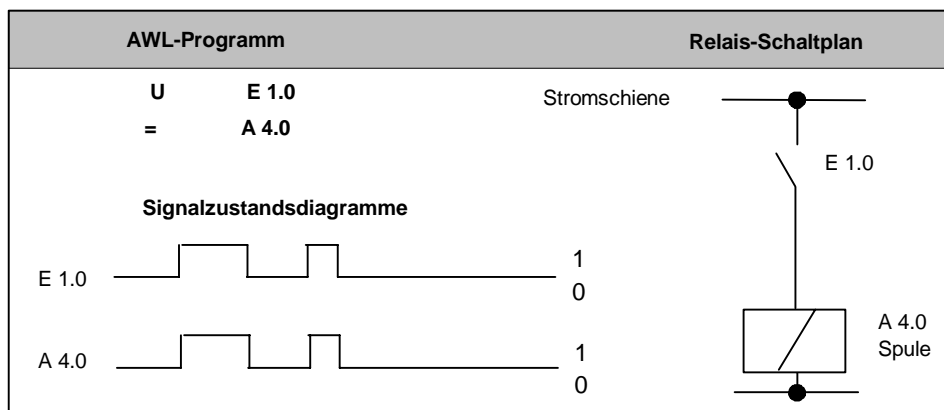
Beschreibung

= <Bit> schreibt bei eingeschaltetem Master Control Relay (MCR = 1) das VKE in das adressierte Bit. Bei MCR = 0 wird statt dem VKE der Wert "0" in das adressierte Bit geschrieben.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	-	0

Beispiel



1.17 R Rücksetze

Format

R <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D

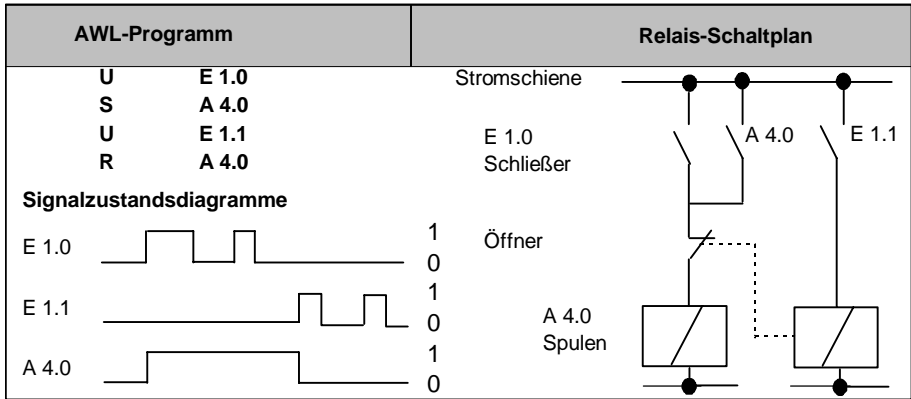
Beschreibung

R (Rücksetze Bit) schreibt bei VKE = 1 und bei eingeschaltetem Master Control Relay (MCR = 1) den Wert "0" in das adressierte Bit. Bei MCR = 0 wird das adressierte Bit nicht verändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	-	0

Beispiel



1.18 S Setze

Format

S <Bit>

Operand	Datentyp	Speicherbereich
<Bit>	BOOL	E, A, M, L, D

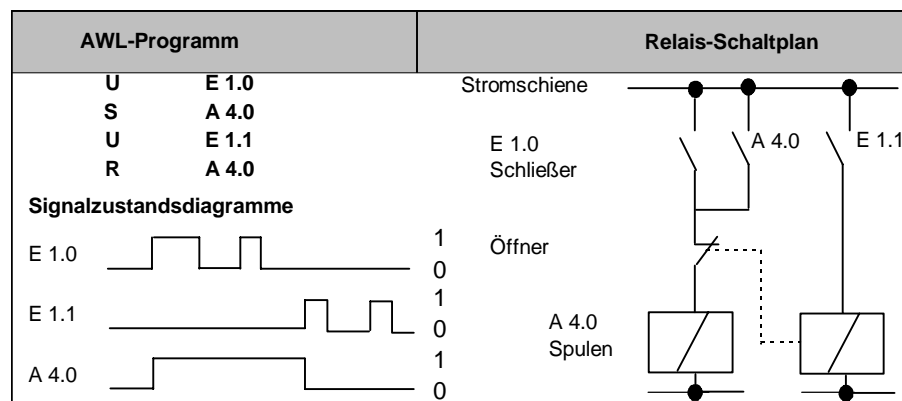
Beschreibung

S (Setze Bit) schreibt bei VKE = 1 und bei eingeschaltetem Master Control Relay (MCR = 1) den Wert "1" in das adressierte Bit. Bei MCR = 0 wird das adressierte Bit nicht verändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	-	0

Beispiel



1.19 NOT Negiere VKE

Format

NOT

Beschreibung

NOT negiert das VKE.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	1	x	-

1.20 SET Setze VKE (=1)

Format

SET

Beschreibung

SET setzt das VKE auf den Signalzustand "1".

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	1	0

Beispiel

AWL-Programm	Signalzustand	Verknüpfungsergebnis (VKE)
SET		1
= M 10.0	1	←
= M 15.1	1	
= M 16.0	1	
CLR		0
= M 10.1	0	←
= M 10.2	0	

1.21 CLR Rücksetze VKE (=0)

Format

CLR

Beschreibung

CLR setzt das VKE auf den Signalzustand "0".

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	0	0	0

Beispiel

AWL-Programm	Signalzustand	Verknüpfungsergebnis (VKE)
SET		1
= M 10.0	1	←
= M 15.1	1	
= M 16.0	1	
CLR		0
= M 10.1	0	←
= M 10.2	0	

1.22 SAVE Sichere VKE im BIE-Bit

Format

SAVE

Beschreibung

SAVE speichert das VKE im BIE-Bit. Das Erstabfragebit /ER wird dabei nicht zurückgesetzt.

Aus diesem Grund wird bei einer UND-Verknüpfung im nächsten Netzwerk der Zustand des BIE-Bits mitverknüpft.

Die Verwendung von **SAVE** und eine nachfolgende Abfrage des BIE-Bits im gleichen Baustein oder in unterlagerten Bausteinen wird nicht empfohlen, da das BIE-Bit durch zahlreiche dazwischen liegende Operationen verändert werden kann. Sinnvoll ist der Einsatz der Operation **SAVE** vor Verlassen eines Baustein, da damit der ENO-Ausgang (=BIE-Bit) auf den Wert des VKE-Bits gesetzt wird und Sie daran eine Fehlerbehandlung des Bausteins anschließen können.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	x	-	-	-	-	-	-	-	-

1.23 FN Flanke Negativ

Format

FN <Bit>

Operand	Datentyp	Speicherbereich	Speicherbereich
<Bit>	BOOL	E, A, M, L, D	Flankenmerker, speichert den vorherigen Signalzustand des VKE.

Beschreibung

FN <Bit> (Flanke Negativ) erkennt eine fallende Flanke, wenn das VKE von "1" auf "0" wechselt und zeigt dies mit VKE = 1 an.

Während eines jeden Programmzyklus wird der Signalzustand des VKE-Bits mit dem Signalzustand des VKE-Bits des vorherigen Zyklus verglichen, um Änderungen des Zustands festzustellen. Um den Vergleich ausführen zu können, muß der Zustand des vorherigen VKE-Bits in der Adresse des Flankenmerkers (<Bit>) gespeichert werden. Unterscheidet sich der aktuelle Signalzustand des VKE-Bits vom vorherigen Zustand ("1") (Erkennung einer fallenden Flanke), ist das VKE-Bit nach dieser Operation "1".

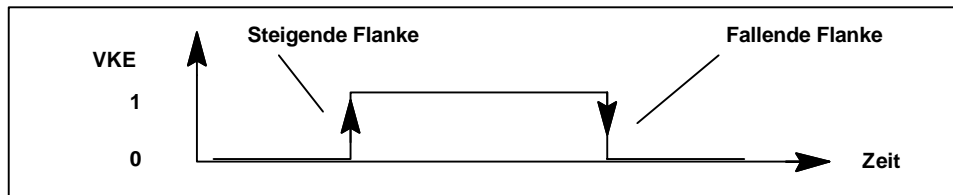
Hinweis

Die Operation ist nicht sinnvoll, falls das zu überwachende Bit im Prozeßabbild liegt. Denn die Lokaldaten eines Bausteins sind nur zu dessen Laufzeit gültig.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	x	1

Definition



Beispiel

Wenn das Automatisierungssystem eine negative Flanke an Kontakt E 1.0 erkennt, aktiviert es den Ausgang A 4.0 für einen OB1-Zyklus.

AWL-Programm		Signalzustandsdiagramme																		
U	E 1.0	E 1.0										1 0								
FN	M 1.0	M 1.0										1 0								
=	A 4.0	A 4.0										1 0								
Zyklus-Nr. des OB1:		<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="width: 20px; text-align: center;">1</td> <td style="width: 20px; text-align: center;">2</td> <td style="width: 20px; text-align: center;">3</td> <td style="width: 20px; text-align: center;">4</td> <td style="width: 20px; text-align: center;">5</td> <td style="width: 20px; text-align: center;">6</td> <td style="width: 20px; text-align: center;">7</td> <td style="width: 20px; text-align: center;">8</td> <td style="width: 20px; text-align: center;">9</td> </tr> </table>									1	2	3	4	5	6	7	8	9	
1	2	3	4	5	6	7	8	9												

1.24 FP Flanke Positiv

Format

FP <Bit>

Operand	Datentyp	Speicherbereich	Speicherbereich
<Bit>	BOOL	E, A, M, L, D	Flankenmerker, speichert den vorherigen Signalzustand des VKE.

Beschreibung

FP <Bit> (Flanke Positiv) erkennt eine steigende Flanke, wenn das VKE von "0" auf "1" wechselt und zeigt dies mit VKE = 1 an.

Während eines jeden Programmzyklus wird der Signalzustand des VKE-Bits mit dem Signalzustand des VKE-Bits des vorherigen Zyklus verglichen, um Änderungen des Zustands festzustellen. Um den Vergleich ausführen zu können, muß der vorherige VKE-Zustand in der Adresse des Flankenmerkers (<Bit>) gespeichert werden. Unterscheidet sich der aktuelle Signalzustand des VKE-Bits vom vorherigen Zustand ("0") (Erkennung einer steigenden Flanke), ist das VKE-Bit nach dieser Operation "1".

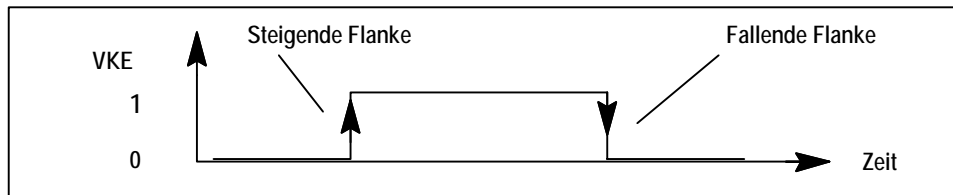
Hinweis

Die Operation ist nicht sinnvoll, falls das zu überwachende Bit im Prozeßabbild liegt. Denn die Lokaldaten eines Bausteins sind nur zu dessen Laufzeit gültig.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	x	x	1

Definition



Beispiel

Wenn das Automatisierungssystem eine steigende Flanke an Kontakt E 1.0 erkennt, aktiviert es den Ausgang A 4.0 für einen OB1-Zyklus.

AWL-Programm		Signalzustandsdiagramme																	
U	E 1.0	E 1.0																	
FP	M 1.0	M 1.0																	
=	A 4.0	A 4.0																	
Zyklus-Nr. des OB1:		<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="width: 20px; text-align: center;">1</td> <td style="width: 20px; text-align: center;">2</td> <td style="width: 20px; text-align: center;">3</td> <td style="width: 20px; text-align: center;">4</td> <td style="width: 20px; text-align: center;">5</td> <td style="width: 20px; text-align: center;">6</td> <td style="width: 20px; text-align: center;">7</td> <td style="width: 20px; text-align: center;">8</td> <td style="width: 20px; text-align: center;">9</td> </tr> </table>									1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9											

2 **Vergleicher**

2.1 **Vergleichsoperationen Übersicht**

Beschreibung

Verglichen werden die Werte von AKKU 2 und AKKU 1 entsprechend der folgenden Vergleichsarten:

== AKKU 2 ist gleich AKKU 1
<> AKKU 2 ist ungleich AKKU 1
> AKKU 2 ist größer als AKKU 1
< AKKU 2 ist kleiner als AKKU 1
>= AKKU 2 ist größer als oder gleich AKKU 1
<= AKKU 2 ist kleiner als oder gleich AKKU 1

Ergibt der Vergleich die Aussage "wahr", so ist das VKE der Operation "1", ansonsten "0". Die Statusbits A1 und A0 zeigen die Relation "kleiner als", "gleich" oder "größer als" an.

Folgende Vergleichsoperationen stehen Ihnen zur Verfügung:

- ? I Vergleiche Ganzzahlen (16 Bit)
- ? D Vergleiche Ganzzahlen (32 Bit)
- ? R Vergleiche Gleitpunktzahlen (32 Bit)

2.2 ? I Vergleiche Ganzzahlen (16 Bit)

Format

==I, <>I, >I, <I, >=I, <=I

Beschreibung

Die Operationen **Vergleiche Ganzzahlen (16 Bit)** vergleichen den Inhalt von AKKU2-L mit dem Inhalt von AKKU1-L. Die Inhalte von AKKU2-L und AKKU1-L werden als Ganzzahlen (16 Bit) ausgewertet. Das Ergebnis des Vergleichs wird vom VKE und den relevanten Bits des Statusworts angezeigt. VKE = 1 zeigt an, daß das Vergleichsergebnis wahr ist. VKE = 0 zeigt an, daß das Vergleichsergebnis falsch ist. Die Statusbits A1 und A0 zeigen die Relation "kleiner als", "gleich" oder "größer als" an.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	0	-	0	x	x	1

Werte des VKE

Ausgeführte Vergleichsoperation	VKE, wenn AKKU 2 > AKKU 1	VKE, wenn AKKU 2 = AKKU 1	VKE, wenn AKKU 2 < AKKU 1
==I	0	1	0
<>I	1	0	1
>I	1	0	0
<I	0	0	1
>=I	1	1	0
<=I	0	1	1

Beispiel

AWL	Erläuterung
L MW10	//Lade den Inhalt von MW10 (16 Bit-Ganzzahl).
L EW24	//Lade den Inhalt von EW24 (16 Bit-Ganzzahl).
>I	//Vergleiche, ob AKKU2-L (MW10) größer (>) ist als AKKU1-L (EW24).
= M 2.0	//VKE = 1, wenn MW10 > EW24.

2.3 ? D Vergleiche Ganzzahlen (32 Bit)

Format

==D, <>D, >D, <D, >=D, <=D

Beschreibung

Die Operationen **Vergleiche Ganzzahlen (32 Bit)** vergleichen den Inhalt von AKKU 2 mit dem Inhalt von AKKU 1. Die Inhalte von AKKU 2 und AKKU 1 werden als Ganzzahlen (32 Bit) ausgewertet. Das Ergebnis des Vergleichs wird vom VKE und den relevanten Bits des Statusworts angezeigt. VKE = 1 zeigt an, daß das Vergleichsergebnis wahr ist. VKE = 0 zeigt an, daß das Vergleichsergebnis falsch ist. Die Statusbits A1 und A0 zeigen die Relation "kleiner als", "gleich" oder "größer als" an.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	0	-	0	x	x	1

Werte des VKE

Ausgeführte Vergleichsoperation	VKE, wenn AKKU 2 > AKKU 1	VKE, wenn AKKU 2 = AKKU 1	VKE, wenn AKKU 2 < AKKU 1
==D	0	1	0
<>D	1	0	1
>D	1	0	0
<D	0	0	1
>=D	1	1	0
<=D	0	1	1

Beispiel

AWL	Erläuterung
L MD10	//Lade den Inhalt von MD10 (32 Bit-Ganzzahl).
L ED24	//Lade den Inhalt von ED24 (32 Bit-Ganzzahl).
>D	//Vergleiche, ob AKKU 2 (MD10) größer (>) ist als AKKU 1 (ED24).
= M 2.0	//VKE = 1, wenn MD10 > ED24.

2.4 ? R Vergleiche Gleitpunktzahlen (32 Bit)

Format

==R, <>R, >R, <R, >=R, <=R

Beschreibung

Die Operationen **Vergleiche Gleitpunktzahlen (32-Bit, IEEE-FP)** vergleichen den Inhalt von AKKU 2 mit dem Inhalt von AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Gleitpunktzahlen (32-Bit, IEEE-FP) ausgewertet. Das Ergebnis des Vergleichs wird vom VKE und den relevanten Bits des Statusworts angezeigt. VKE = 1 zeigt an, daß das Vergleichsergebnis wahr ist. VKE = 0 zeigt an, daß das Vergleichsergebnis falsch ist. Die Statusbits A1 und A0 zeigen die Relation "kleiner als", "gleich" oder "größer als" an.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	0	x	x	1

Werte des VKE

Ausgeführte Vergleichsoperation	VKE, wenn AKKU 2 > AKKU 1	VKE, wenn AKKU 2 = AKKU 1	VKE, wenn AKKU 2 < AKKU 1
==R	0	1	0
<>R	1	0	1
>R	1	0	0
<R	0	0	1
>=R	1	1	0
<=R	0	1	1

Beispiel

AWL	Erläuterung
L MD10	//Lade den Inhalt von MD10 (Gleitpunktzahl).
L 1.359E+02	//Lade die Konstante 1.359E+02.
>R	//Vergleiche, ob AKKU 2 (MD10) größer (>) ist als AKKU 1 (1.359E+02).
= M 2.0	//VKE = 1, wenn MD10 > 1.359E+02.

3 Umwandler

3.1 Umwandlungsoperationen Übersicht

Beschreibung

Mit den folgenden Operationen können Sie binär-codierte Dezimalzahlen und Ganzzahlen in andere Zahlenarten umwandeln:

- BTI BCD wandeln in Ganzzahl (16 Bit)
- ITB Ganzzahl (16 Bit) wandeln in BCD
- BTD BCD wandeln in Ganzzahl (32 Bit)
- ITD Ganzzahl (16 Bit) wandeln in Ganzzahl (32 Bit)
- DTB Ganzzahl (32 Bit) wandeln in BCD
- DTR Ganzzahl (32 Bit) wandeln in Gleitpunktzahl (32 Bit, IEEE-FP)

Mit den folgenden Operationen können Sie die Komplemente von Ganzzahlen bilden oder das Vorzeichen einer Gleitpunktzahl wechseln:

- INVI 1-Komplement Ganzzahl (16 Bit)
- INVD 1-Komplement Ganzzahl (32 Bit)
- NEGI 2-Komplement Ganzzahl (16 Bit)
- NEGD 2-Komplement Ganzzahl (32 Bit)
- NEGR Negiere Gleitpunktzahl (32 Bit, IEEE-FP)

Mit den folgenden Operationen können Sie die Reihenfolge der Bytes im niederwertigen Wort von AKKU 1 oder im gesamten Akkumulator umkehren:

- TAW Tausche Reihenfolge der Bytes im AKKU 1-L (16 Bit)
- TAD Tausche Reihenfolge der Bytes im AKKU 1 (32 Bit)

Mit den folgenden Operationen können Sie eine Gleitpunktzahl (32 Bit, IEEE-FP) in AKKU 1 in eine Ganzzahl (32 Bit) umwandeln. Die einzelnen Operationen unterscheiden sich in der Art des Rundens.

- RND Runden einer Gleitpunktzahl zur Ganzzahl
- TRUNC Runden einer Gleitpunktzahl durch Abschneiden
- RND+ Runden einer Gleitpunktzahl zur nächsthöheren Ganzzahl
- RND- Runden einer Gleitpunktzahl zur nächstniederen Ganzzahl

3.2 BTI BCD wandeln in Ganzzahl (16 Bit)

Format

BTI

Beschreibung

BTI (Dezimal-Dual-Umwandlung einer dreistelligen BCD-Zahl) wertet den Inhalt von AKKU1-L als eine dreistellige binär-codierte Dezimalzahl (BCD) aus und wandelt diese in eine Ganzzahl (16 Bit) um. Das Ergebnis wird in AKKU1-L gespeichert. AKKU1-H und AKKU 2 werden nicht verändert.

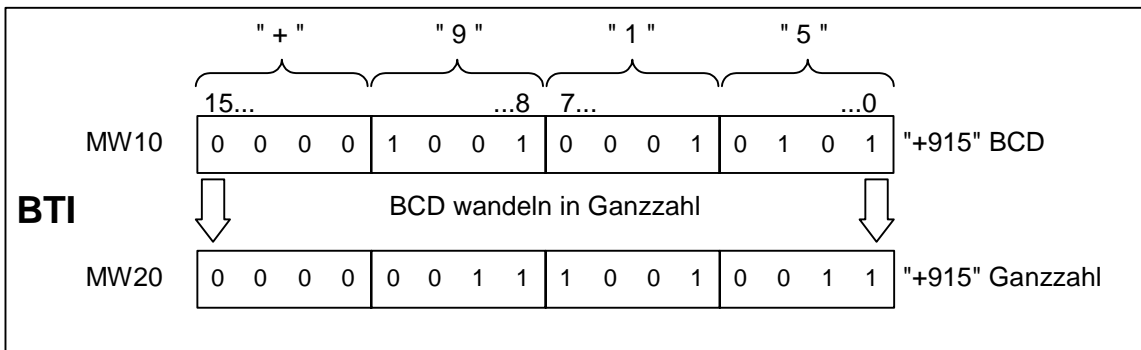
BCD-Zahl in AKKU1-L: Zulässig sind Werte von "-999" bis "+999". Bit 0 bis Bit 11 geben den Wert und Bit 15 das Vorzeichen (0 = positiv, 1= negativ) der BCD-Zahl an. Bit 12 bis Bit 14 werden bei der Umwandlung nicht verwendet. Wenn eine Dezimalziffer (eine 4-Bit-Tetrade in der BCD-Darstellung) im ungültigen Bereich von 10 bis 15 liegt, tritt während einer versuchten Umwandlung ein BCDF-Fehler auf. Im allgemeinen geht das Automatisierungssystem dann in den Betriebszustand STOP. Sie können jedoch mit Hilfe von OB121 eine andere Fehlerreaktion auf diesen Synchronfehler programmieren.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MW10	//Lade die BCD-Zahl in AKKU1-L.
BTI	//Wandle die BCD-Zahl in eine Ganzzahl um, speichere das Ergebnis in AKKU1-L.
T MW20	//Transferiere das Ergebnis (Ganzzahl, 16 Bit) nach MW20.



3.3 ITB Ganzzahl (16 Bit) wandeln in BCD

Format

ITB

Beschreibung

ITB (Dual-Dezimal-Umwandlung einer Ganzzahl, 16 Bit) wertet den Inhalt von AKKU1-L als eine Ganzzahl (16 Bit) aus und wandelt diese in eine dreistellige binär-codierte Dezimalzahl (BCD) um. Das Ergebnis wird in AKKU1-L gespeichert. Bit 0 bis Bit 11 geben den Wert der BCD-Zahl an. Bit 12 bis Bit 15 stellen den Vorzeichenstatus (0000 = positiv, 1111 = negativ) der BCD-Zahl dar. AKKU1-H und AKKU 2 werden nicht verändert.

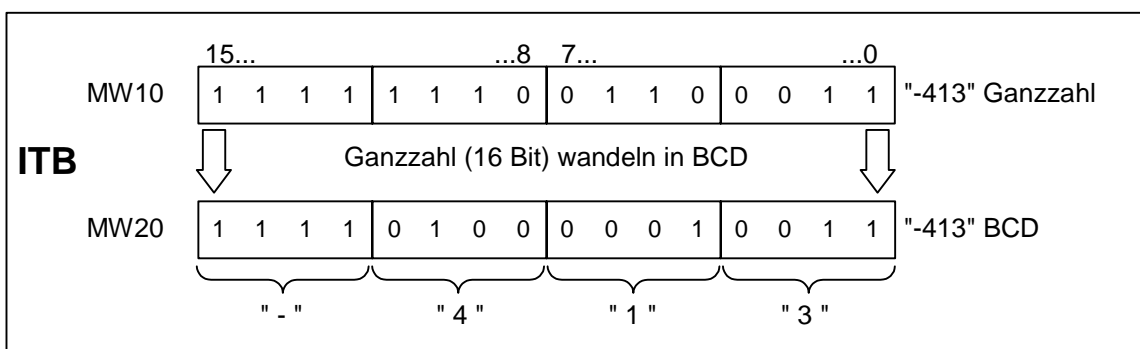
Die BCD-Zahl kann im Bereich von "-999" bis "+999" liegen. Befindet sich die Zahl außerhalb des zulässigen Bereichs, werden die Statusbits OV und OS auf "1" gesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
L MW10	//Lade die Ganzzahl in AKKU1-L.
ITB	//Wandle die Ganzzahl (16 Bit) in eine BCD-Zahl um, speichere das Ergebnis in AKKU1-L.
T MW20	//Transferiere das Ergebnis (BCD-Zahl) nach MW20.



3.4 BT D BCD wandeln in Ganzzahl (32 Bit)

Format

BT D

Beschreibung

BT D (Dezimal-Dual-Umwandlung einer siebenstelligen BCD-Zahl) wertet den Inhalt von AKKU 1 als eine siebenstellige binär-codierte Dezimalzahl (BCD) aus und wandelt diese in eine Ganzzahl (32 Bit) um. Das Ergebnis wird in AKKU 1 gespeichert. AKKU 2 wird nicht verändert.

BCD-Zahl in AKKU 1: Zulässig sind Werte von "-9999999" bis "+9999999". Bit 0 bis Bit 27 geben den Wert und Bit 31 das Vorzeichen (0 = positiv, 1 = negativ) der BCD-Zahl an. Bit 28 bis Bit 30 werden bei der Umwandlung nicht verwendet.

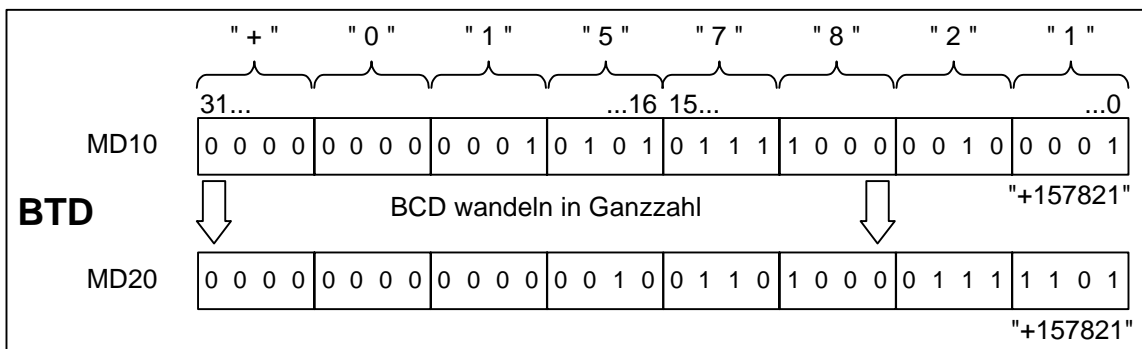
Wenn eine Dezimalziffer (eine 4-Bit-Tetrade in der BCD-Darstellung) im ungültigen Bereich von 10 bis 15 liegt, tritt während einer versuchten Umwandlung ein BCD-Fehler auf. Im allgemeinen geht das Automatisierungssystem dann in den Betriebszustand STOP. Sie können jedoch mit Hilfe von OB121 eine andere Fehlerreaktion auf diesen Synchronfehler programmieren.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MD10	//Lade die BCD-Zahl in AKKU 1.
BT D	//Wandle die BCD-Zahl in eine Ganzzahl um, speichere das Ergebnis in AKKU 1.
T MD20	//Transferiere das Ergebnis (Ganzzahl, 32 Bit) nach MD20.



3.5 ITD Ganzzahl (16 Bit) wandeln in Ganzzahl (32 Bit)

Format

ITD

Beschreibung

ITD (Umwandlung einer Ganzzahl, 16 Bit, in eine Ganzzahl, 32 Bit) wertet den Inhalt von AKKU1-L als Ganzzahl (16 Bit) aus und wandelt diese in eine Ganzzahl (32 Bit) um. Das Ergebnis wird in AKKU 1 gespeichert. AKKU 2 wird nicht verändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MW12	//Lade die Ganzzahl (16 Bit) in AKKU 1-L.
ITD	//Wandle die Ganzzahl (16 Bit) in eine Ganzzahl (32 Bit) um, speichere das Ergebnis in AKKU 1.
T MD20	//Transferiere das Ergebnis (Ganzzahl, 32 Bit) nach MD20.

Beispiel: MW12 = "-10" (Ganzzahl, 16 Bit):

Inhalt	AKKU1-H				AKKU1-L			
Bit	31 16	15 0
vor Ausführung von ITD	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
nach Ausführung von ITD	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 oder 1, Bits sind für die Umwandlung nicht erforderlich)							

3.6 DTB Ganzzahl (32 Bit) wandeln in BCD

Format

DTB

Beschreibung

DTB (Dual-Dezimal-Umwandlung einer Ganzzahl, 32 Bit) wertet den Inhalt von AKKU 1 als eine Ganzzahl (32 Bit) aus und wandelt diese in eine siebenstellige binär-codierte Dezimalzahl um. Das Ergebnis wird in AKKU 1 gespeichert. Bit 0 bis Bit 27 geben den Wert der BCD-Zahl an. Bit 28 bis Bit 31 stellen den Vorzeichenstatus der BCD-Zahl (0000 = positiv, 1111 = negativ) dar. AKKU 2 wird nicht verändert.

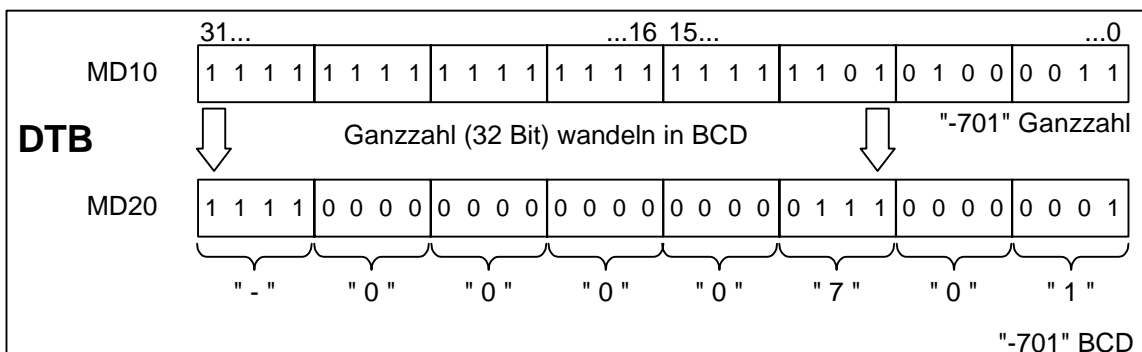
Die BCD-Zahl kann im Bereich zwischen "-9999999" und "+9999999" liegen. Wenn sich die Zahl nicht im zulässigen Bereich befindet, werden die Statusbits OV und OS auf "1" gesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
L MD10	//Lade die Ganzzahl (32 Bit) in AKKU 1.
DTB	//Wandle die Ganzzahl (32 Bit) in eine BCD-Zahl um, speichere das Ergebnis in AKKU 1.
T MD20	//Transferiere das Ergebnis (BCD-Zahl) nach MD20.



3.7 DTR Ganzzahl (32 Bit) wandeln in Gleitpunktzahl (32 Bit, IEEE-FP)

Format

DTR

Beschreibung

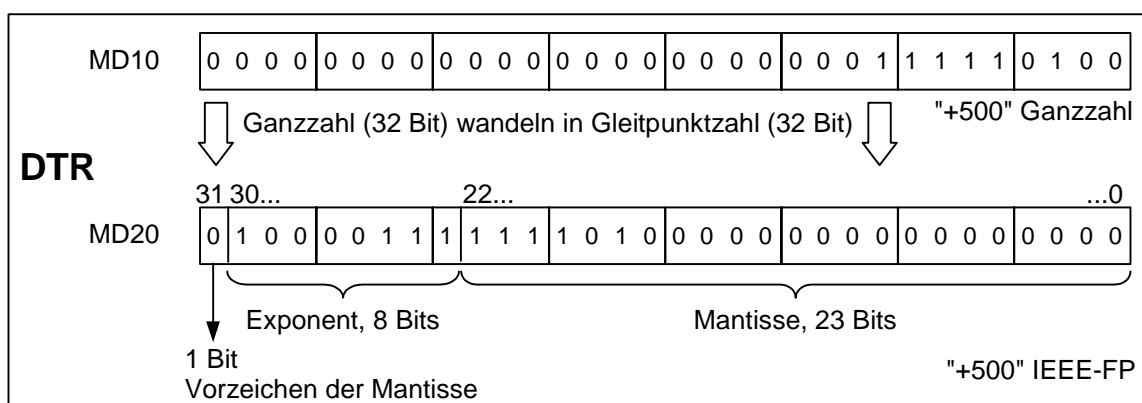
DTR (Umwandlung einer Ganzzahl, 32 Bit, in eine Gleitpunktzahl, 32 Bit, IEEE-FP) wertet den Inhalt von AKKU 1 als eine Ganzzahl (32 Bit) aus und wandelt diese in eine Gleitpunktzahl (32 Bit, IEEE-FP) um. Sofern erforderlich, rundet die Operation das Ergebnis (eine Ganzzahl, 32 Bit, hat eine höhere Genauigkeit als eine Gleitpunktzahl, 32 Bit, IEEE-FP). Das Ergebnis wird in AKKU 1 gespeichert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MD10	//Lade die Ganzzahl (32 Bit) in AKKU 1.
DTR	//Wandle die Ganzzahl (32 Bit) in eine Gleitpunktzahl (32-Bit, IEEE- FP) um, speichere das Ergebnis in AKKU 1.
T MD20	//Transferiere das Ergebnis (BCD-Zahl) nach MD20.



3.8 INVI 1-Komplement Ganzzahl (16 Bit)

Format

INVI

Beschreibung

INVI (1-Komplement Ganzzahl, 16 Bit) bildet das Einerkomplement des 16-Bit-Wertes in AKKU1-L. Beim Bilden des Einerkomplements werden die einzelnen Bits umgekehrt, d. h. die Nullen werden durch Einsen ersetzt und die Einsen durch Nullen. Das Ergebnis wird in AKKU1-L gespeichert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L EW8	//Lade den Wert in AKKU1-L.
INVI	//Bilde das Einerkomplement (16 Bit).
T MW10	//Transferiere das Ergebnis nach MW10.

Inhalt	AKKU1-L			
Bit	15 0
vor Ausführung von INVI	0110	0011	1010	1110
nach Ausführung von INVI	1001	1100	0101	0001

3.9 INVD 1-Komplement Ganzzahl (32 Bit)

Format

INVD

Beschreibung

INVD (1-Komplement Ganzzahl, 32 Bit) bildet das Einerkomplement des 32-Bit-Wertes in AKKU 1. Beim Bilden des Einerkomplements werden die einzelnen Bits umgekehrt, d. h. die Nullen werden durch Einsen ersetzt und die Einsen durch Nullen. Das Ergebnis wird in AKKU 1 gespeichert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L ED8	//Lade den Wert in AKKU 1.
INVD	//Bilde das Einerkomplement (32 Bit).
T MD10	//Transferiere das Ergebnis nach MD10.

Inhalt	AKKU1-H				AKKU1-L			
Bit	31 16	15 0
vor Ausführung von INVD	0110	1111	1000	1100	0110	0011	1010	1110
nach Ausführung von INVD	1001	0000	0111	0011	1001	1100	0101	0001

3.10 NEGI 2-Komplement Ganzzahl (16 Bit)

Format

NEGI

Beschreibung

NEGI (2-Komplement Ganzzahl, 16 Bit) bildet das Zweierkomplement des 16-Bit-Wertes in AKKU1-L. Beim Bilden des Zweierkomplements werden die einzelnen Bits umgekehrt, d. h. die Nullen werden durch Einsen ersetzt und die Einsen durch Nullen. Dann wird eine "1" addiert. Das Ergebnis wird in AKKU1-L gespeichert. Die Operation 2-Komplement Ganzzahl entspricht einer Multiplikation mit "-1". Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits:	A1	A0	OV	OS
Ergebnis = 0	0	0	0	-
-32768 <= Ergebnis <= -1	0	1	0	-
32767 >= Ergebnis >= 1	1	0	0	-
Ergebnis = 2768	0	1	1	1

Beispiel

AWL	Erläuterung
L EW8	//Lade den Wert in AKKU1-L.
NEGI	//Bilde das Zweierkomplement (16 Bit).
T MW10	//Transferiere das Ergebnis nach MW10.

Inhalt	AKKU1-L			
Bit	15 0
vor Ausführung von NEGI	0101	1101	0011	1000
nach Ausführung von NEGI	1010	0010	1100	1000

3.11 NEGD 2-Komplement Ganzzahl (32 Bit)

Format

NEGD

Beschreibung

NEGD (2-Komplement Ganzzahl, 32 Bit) bildet das Zweierkomplement des 32-Bit-Wertes in AKKU 1. Beim Bilden des Zweierkomplements werden die einzelnen Bits umgekehrt, d. h. die Nullen werden durch Einsen ersetzt und die Einsen durch Nullen. Dann wird eine "1" addiert. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation 2-Komplement Ganzzahl entspricht einer Multiplikation mit "-1". Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits:	A1	A0	OV	OS
Ergebnis = 0	0	0	0	-
-2.147.483.648 <= Ergebnis <= -1	0	1	0	-
2.147.483.647 >= Ergebnis >= 1	1	0	0	-
Ergebnis = 2 147 483 648	0	1	1	1

Beispiel

AWL	Erläuterung
L ED8	//Lade den Wert in AKKU 1.
NEGD	//Bilde das Zweierkomplement (32 Bit).
T MD10	//Transferiere das Ergebnis nach MD10.

Inhalt	AKKU1-H				AKKU1-L			
Bit	31 16	15 0
vor Ausführung von NEGD	0101	1111	0110	0100	0101	1101	0011	1000
nach Ausführung von NEGD	1010	0000	1001	1011	1010	0010	1100	1000

3.12 NEGR Negiere Gleitpunktzahl

Format

NEGR

Beschreibung

NEGR (Negiere Gleitpunktzahl, 32 Bit, IEEE-FP) negiert die Gleitpunktzahl (32 Bit, IEEE-FP) in AKKU 1. Die Operation kehrt den Status von Bit 31 in AKKU 1 um (Vorzeichen der Mantisse). Das Ergebnis wird in AKKU 1 gespeichert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L ED8	//Lade den Wert in AKKU 1 (Beispiel ED8 = 1.5E+02).
NEGR	//Negiere Gleitpunktzahl (32 Bit, IEEE-FP), speichere das Ergebnis in AKKU 1.
T MD10	//Transferiere das Ergebnis nach MD10 (Beispiel Ergebnis = -1.5E+02).

3.13 TAW Tausche Reihenfolge der Bytes im AKKU 1-L (16 Bit)

Format

TAW

Beschreibung

TAW kehrt die Reihenfolge der Bytes in AKKU1-L um. Das Ergebnis wird in AKKU1-L gespeichert. AKKU1-H und AKKU 2 werden nicht verändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L MW10	//Lade den Wert von MW10 in AKKU 1.	
TAW	//Kehre die Reihenfolge der Bytes in AKKU1-L um.	
T MW20	//Transferiere das Ergebnis nach MW20.	

Inhalt	AKKU1-H-H	AKKU1-H-L	AKKU1-L-H	AKKU1-L-L
vor der Ausführung von TAW	Wert A	Wert B	Wert C	Wert D
nach der Ausführung von TAW	Wert A	Wert B	Wert D	Wert C

3.14 TAD Tausche Reihenfolge der Bytes im AKKU 1 (32 Bit)

Format

TAD

Beschreibung

TAD kehrt die Reihenfolge der Bytes in AKKU 1 um. Das Ergebnis wird in AKKU 1 gespeichert. AKKU 2 wird nicht verändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L MD10	//Lade den Wert von MD10 in AKKU 1.	
TAD	//Kehre die Reihenfolge der Bytes in AKKU 1 um.	
T MD20	//Transferiere das Ergebnis nach MD20.	

Inhalt	AKKU1-H-H	AKKU1-H-L	AKKU1-L-H	AKKU1-L-L
vor der Ausführung von TAD	Wert A	Wert B	Wert C	Wert D
nach der Ausführung von TAD	Wert D	Wert C	Wert B	Wert A

3.15 RND Runden einer Gleitpunktzahl zur Ganzzahl

Format

RND

Beschreibung

RND (Umwandlung einer Gleitpunktzahl, 32 Bit, IEEE-FP, in eine Ganzzahl, 32 Bit) wertet den Inhalt von AKKU 1 als eine Gleitpunktzahl (32 Bit, IEEE-FP) aus. Die Operation wandelt die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um und rundet das Ergebnis zur nächsten Ganzzahl. Liegt der Bruch der umgewandelten Zahl genau zwischen einem geraden und einem ungeraden Ergebnis, rundet die Operation zum geraden Ergebnis. Liegt die Zahl außerhalb des zulässigen Bereichs, werden die Statusbits OV und OS auf "1" gesetzt.

Tritt ein Fehler auf (Vorliegen einer NaN oder einer Gleitpunktzahl, die nicht als Ganzzahl, 32 Bit, dargestellt werden kann), wird die Umwandlung nicht ausgeführt, sondern Überlauf angezeigt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	X	X	-	-	-	-

Beispiel

AWL	Erläuterung
L MD10	//Lade die Gleitpunktzahl (32 Bit, IEEE-FP) in AKKU1-L.
RND	//Wandle die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um und runde das Ergebnis. Speichere das Ergebnis in AKKU1.
T MD20	//Transferiere das Ergebnis (Ganzzahl, 32 Bit) nach MD20.

Wert vor der Umwandlung		Wert nach der Umwandlung
MD10 = "100.5"	=> RND =>	MD20 = "+100"
MD10 = "-100.5"	=> RND =>	MD20 = "-100"

3.16 TRUNC Runden einer Gleitpunktzahl durch Abschneiden

Format

TRUNC

Beschreibung

TRUNC (Umwandlung einer Gleitpunktzahl, 32 Bit, IEEE-FP, in eine Ganzzahl, 32 Bit) wertet den Inhalt von AKKU 1 als eine Gleitpunktzahl (32 Bit, IEEE-FP) aus. Die Operation wandelt die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um. Das Ergebnis besteht aus dem ganzzahligen Anteil der Gleitpunktzahl (IEEE-Rundungsmodus 'Round to Zero'). Liegt die Zahl außerhalb des zulässigen Bereichs, werden die Statusbits OV und OS auf "1" gesetzt. Das Ergebnis wird in AKKU 1 gespeichert.

Tritt ein Fehler auf (Vorliegen einer NaN oder einer Gleitpunktzahl, die nicht als Ganzzahl, 32 Bit, dargestellt werden kann), wird die Umwandlung nicht ausgeführt, sondern Überlauf angezeigt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
L MD10	//Lade die Gleitpunktzahl (32 Bit, IEEE-FP) in AKKU1-L.
TRUNC	//Wandle die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um und runde das Ergebnis. Speichere das Ergebnis in AKKU1.
T MD20	//Transferiere das Ergebnis (Ganzzahl, 32 Bit) nach MD20.

Wert vor der Umwandlung		Wert nach der Umwandlung
MD10 = "100.5"	=> TRUNC =>	MD20 = "+100"
MD10 = "-100.5"	=> TRUNC =>	MD20 = "-100"

3.17 RND+ Runden einer Gleitpunktzahl zur nächsthöheren Ganzzahl

Format

RND+

Beschreibung

RND+ (Umwandlung einer Gleitpunktzahl, 32 Bit, IEEE-FP, in eine Ganzzahl, 32 Bit) wertet den Inhalt von AKKU 1 als eine Gleitpunktzahl (32 Bit, IEEE-FP) aus. Die Operation wandelt die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um und rundet das Ergebnis zur kleinsten ganzen Zahl, die größer oder gleich der umgewandelten Gleitpunktzahl ist (IEEE-Rundungsmodus "Round to +infinity"). Liegt die Zahl außerhalb des zulässigen Bereichs, werden die Statusbits OV und OS auf "1" gesetzt. Das Ergebnis wird in AKKU 1 gespeichert.

Tritt ein Fehler auf (Vorliegen einer NaN oder einer Gleitpunktzahl, die nicht als Ganzzahl, 32 Bit, dargestellt werden kann), wird die Umwandlung nicht ausgeführt, sondern Überlauf angezeigt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
L MD10	//Lade die Gleitpunktzahl (32 Bit, IEEE-FP) in AKKU1-L.
RND	//Wandle die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um und runde das Ergebnis. Speichere das Ergebnis in AKKU1.
T MD20	//Transferiere das Ergebnis (Ganzzahl, 32 Bit) nach MD20.

Wert vor der Umwandlung		Wert nach der Umwandlung
MD10 = "100.5"	=> RND+ =>	MD20 = "+101"
MD10 = "-100.5"	=> RND+ =>	MD20 = "-100"

3.18 RND- Runden einer Gleitpunktzahl zur nächstniederen Ganzzahl

Format

RND-

Beschreibung

RND- (Umwandlung einer Gleitpunktzahl, 32 Bit, IEEE-FP, in eine Ganzzahl, 32 Bit) wertet den Inhalt von AKKU 1 als eine Gleitpunktzahl (32 Bit, IEEE-FP) aus. Die Operation wandelt die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um und rundet das Ergebnis zu der nächsten ganzen Zahl, die kleiner oder gleich der umgewandelten Gleitpunktzahl ist (IEEE-Rundungsmodus "Round to -infinity"). Liegt die Zahl außerhalb des zulässigen Bereichs, werden die Statusbits OV und OS auf "1" gesetzt. Das Ergebnis wird in AKKU 1 gespeichert.

Tritt ein Fehler auf (Vorliegen einer NaN oder einer Gleitpunktzahl, die nicht als Ganzzahl, 32 Bit, dargestellt werden kann), wird die Umwandlung nicht ausgeführt, sondern Überlauf angezeigt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
L MD10	//Lade die Gleitpunktzahl (32 Bit, IEEE-FP) in AKKU1-L.
RND-	//Wandle die Gleitpunktzahl (32 Bit, IEEE-FP) in eine Ganzzahl (32 Bit) um und runde das Ergebnis. Speichere das Ergebnis in AKKU1.
T MD20	//Transferiere das Ergebnis (Ganzzahl, 32 Bit) nach MD20.

Wert vor der Umwandlung		Wert nach der Umwandlung
MD10 = "100.5"	=> RND- =>	MD20 = "+100"
MD10 = "-100.5"	=> RND- =>	MD20 = "-101"

4 Zähler

4.1 Zähloperationen Übersicht

Beschreibung

Ein Zähler ist ein Funktionselement der Programmiersprache STEP 7. Zähler haben einen eigenen reservierten Speicherbereich in Ihrer CPU. Dieser Speicherbereich reserviert ein Wort von 16 Bit für jeden Zähler. Das Programmieren mit AWL unterstützt 256 Zähler. Die bei Ihrer CPU verfügbare Anzahl von Zählern entnehmen Sie bitte deren technischen Daten. Zähloperationen sind die einzigen Funktionen, die Zugriff auf den für Zähler reservierten Speicherbereich haben.

Folgende Zähloperationen stehen Ihnen zur Verfügung:

- FR Freigabe Zähler
- L Lade aktuellen Zählwert als Ganzzahl in AKKU 1
- LC Lade aktuellen Zählwert als BCD in AKKU 1
- R Rücksetze Zähler
- S Setze Zählerstartwert
- ZV Zählen vorwärts
- ZR Zählen rückwärts

4.2 FR Freigabe Zähler

Format

FR <Zähler>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zähler>	COUNTER	Z	Zähler; Bereich hängt von der CPU ab.

Beschreibung

FR <Zähler> löscht den Flankenmerker, der den adressierten Zähler auf Vorwärts- bzw. Rückwärtszählen setzt, wenn das VKE von "0" auf "1" wechselt. Die Freigabe des Zählers ist nicht erforderlich, wenn ein Zähler gesetzt werden soll oder wenn die normale Zählfunktion ausgeführt wird. Das heißt, trotz konstantem VKE von 1 an den Anweisungen **Setze Zählerstartwert**, **Zählen vorwärts** oder **Zählen rückwärts**, werden nach der Freigabe diese Operationen erneut ausgeführt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.0	//Frage den Signalzustand am Eingang E 2.0 ab.
FR Z3	//Gib den Zähler Z3 frei, wenn das VKE von "0" auf "1" wechselt.

4.3 L Lade aktuellen Zählwert als Ganzzahl in AKKU 1

Format

L <Zähler>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zähler>	COUNTER	Z	Zähler; Bereich hängt von der CPU ab.

Beschreibung

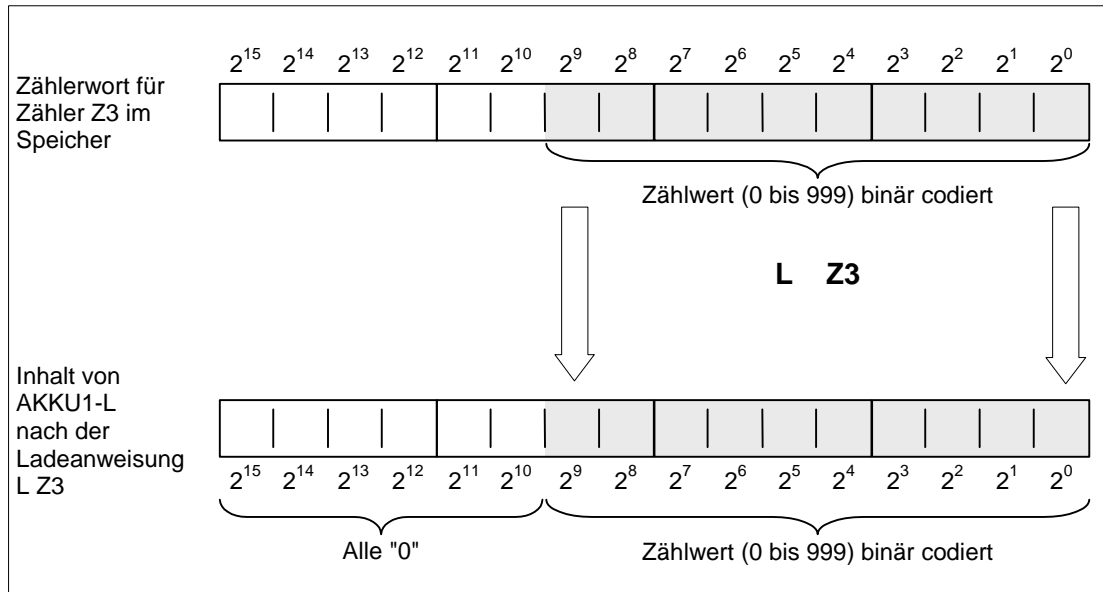
L <Zähler> lädt den aktuellen Zählwert des adressierten Zählers als Ganzzahl in AKKU1-L, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 gespeichert wurde.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L Z3	//Lade den Zählwert des Zählers Z3 im Binärformat in AKKU1-L.



4.4 LC Lade aktuellen Zählwert als BCD in AKKU 1

Format

LC <Zähler>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zähler>	COUNTER	Z	Zähler; Bereich hängt von der CPU ab.

Beschreibung

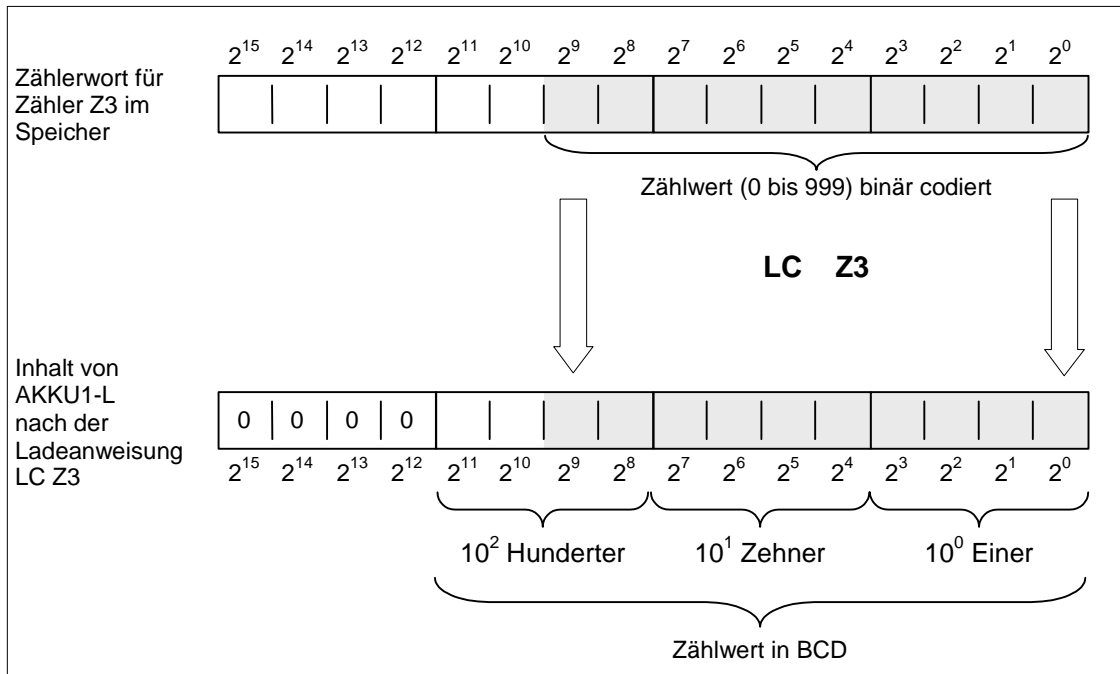
LC <Zähler> lädt den aktuellen Zählwert des adressierten Zählers als BCD-Zahl in AKKU 1, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 gespeichert wurde.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
LC Z3	//Lade den Zählwert des Zählers Z3 im BCD-Format in AKKU1-L.



4.5 R Rücksetze Zähler

Format

R <Zähler>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zähler>	COUNTER	Z	Zähler, der zurückgesetzt werden soll; Bereich hängt von der CPU ab.

Beschreibung

R <Zähler> lädt den Zählwert "0" in den adressierten Zähler, wenn das VKE = 1 ist.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.3	//Frage den Signalzustand am Eingang E 2.3 ab.
R Z3	//Setze den Zähler Z3 auf den Wert "0" zurück, wenn das VKE von "0" auf "1" wechselt.

4.6 S Setze Zählerstartwert

Format

S <Zähler>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zähler>	COUNTER	Z	Zähler, der voreingestellt werden soll; Bereich hängt von der CPU ab.

Beschreibung

S <Zähler> lädt den Zählwert aus AKKU1-L in den adressierten Zähler, wenn das VKE von "0" auf "1" wechselt. Der Zählwert in AKKU 1 muß als BCD-Zahl zwischen "0" und "999" vorliegen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.3	//Frage den Signalzustand am Eingang E 2.3 ab.
L C#3	//Lade den Zählwert 3 in AKKU1-L.
S Z1	//Setze den Zähler Z1 auf den Zählwert, wenn das VKE von "0" auf "1" wechselt.

4.7 ZV Zählen vorwärts

Format

ZV <Zähler>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zähler>	COUNTER	Z	Zähler; Bereich hängt von der CPU ab.

Beschreibung

ZV <Zähler> erhöht den Zählwert des adressierten Zählers um "1", wenn das VKE von "0" auf "1" wechselt und der Zählwert kleiner als "999" ist. Wenn der Zählwert den oberen Grenzwert "999" erreicht, wird der Wert nicht weiter erhöht. Weiteres Wechseln des VKE hat keine Auswirkungen. Das Überlaufbit (OV) wird nicht gesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.1	//Frage den Signalzustand am Eingang E 2.1 ab.
ZV Z3	//Inkrementiere den Zähler Z3 um den Wert 1, wenn das VKE von "0" auf "1" wechselt.

4.8 ZR Zählen rückwärts

Format

ZR <Zähler>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zähler>	COUNTER	Z	Zähler; Bereich hängt von der CPU ab.

Beschreibung

ZR <Zähler> vermindert den Zählwert des adressierten Zählers um "1", wenn das VKE von "0" auf "1" wechselt und der Zählwert größer als "0" ist. Wenn der Zählwert den unteren Grenzwert "0" erreicht, wird der Wert nicht weiter vermindert. Weiteres Wechseln des VKE hat keine Auswirkungen, weil der Zähler nicht mit negativen Werten arbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
L C#14	//Voreingestellter Wert des Zählers.
U E 0.1	//Voreingestellter Zähler nach der Erkennung der steigenden Flanke am Eingang E 0.1.
S Z1	//Lade den Zähler Z1 mit der Voreinstellung, sofern er freigegeben ist.
U E 0.0	//Vermindere um "1" bei jeder steigenden Flanke an E 0.0.
ZR Z1	//Dekrementiere den Zähler Z1 um "1", wenn das VKE abhängig von Eingang E 0.0 von "0" auf "1" wechselt.
UN Z1	//Null-Erkennung mit dem Bit Z1.
= A 0.0	//Wenn der Wert von Zähler Z1 "0" ist, ist A 0.0 = 1.

5 DB-Aufruf

5.1 Datenbausteinoperationen Übersicht

Beschreibung

Mit der Operation AUF (Öffne Datenbaustein) können Sie einen globalen Datenbaustein oder einen Instanz-Datenbaustein öffnen. Es kann gleichzeitig im Programm je ein globaler Datenbaustein und ein Instanz-Datenbaustein geöffnet sein.

Folgende Datenbausteinoperationen stehen Ihnen zur Verfügung:

- AUF Datenbaustein öffnen
- TDB Tausche Global-DB und Instanz-DB
- L DBLG Lade Länge Global-DB in AKKU 1
- L DBNO Lade Nummer Global-DB in AKKU 1
- L DILG Lade Länge Instanz-DB in AKKU 1
- L DINO Lade Nummer Instanz-DB in AKKU 1

5.2 AUF Datenbaustein öffnen

Format

AUF <Datenbaustein>

Operand	Datenbausteintyp	Quelladresse
<Datenbaustein>	DB, DI	1 bis 65535

Beschreibung

AUF <Datenbaustein> öffnet einen Datenbaustein als Global-Datenbaustein oder als Instanz-Datenbaustein. Es können jeweils ein Global-Datenbaustein und ein Instanz-Datenbaustein gleichzeitig geöffnet sein.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
AUF DB10	//Öffne den Datenbaustein DB10 als Global-Datenbaustein.
L DBW35	//Lade Datenwort DBW35 des geöffneten Datenbausteins in AKKU1-L.
T MW22	//Transferiere den Inhalt von AKKU1-L ins MW22.
AUF DI20	//Öffne den Datenbaustein DB20 als Instanz-Datenbaustein.
L DIB12	//Lade Datenbyte DIB12 des geöffneten Instanz-Datenbausteins in AKKU1-L-L.
T DBB37	//Transferiere den Inhalt von AKKU1-L-L nach Datenbyte DBB37 des geöffneten Global-Datenbausteins.

5.3 TDB Tausche Global-DB und Instanz-DB

Format

TDB

Beschreibung

TDB vertauscht die Datenbausteinregister. Ein Global-Datenbaustein wird so zum Instanz-Datenbaustein und umgekehrt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

5.4 L DBLG Lade Länge Global-DB in AKKU 1

Format

L DBLG

Beschreibung

L DBLG (Lade die Länge des Global-Datenbausteins) lädt die Länge des Global-Datenbausteins in AKKU 1, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 gespeichert wurde.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
AUF DB10	//Öffne den Datenbaustein DB10 als Global-Datenbaustein.	
L DBLG	//Lade die Länge des Global-Datenbausteins (Länge von DB10).	
L MD10	//Vergleichswert, ob der Datenbaustein lang genug ist.	
<D		
SPB ERRO	//Springe zur Sprungmarke ERRO, wenn die Länge des Datenbausteins kleiner ist als der Wert in MD10.	

5.5 L DBNO Lade Nummer Global-DB in AKKU 1

Format

L DBNO

Beschreibung

L DBNO (Lade die Nummer des Global-Datenbausteins) lädt die Nummer des geöffneten Global-Datenbausteins in AKKU1, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 gespeichert wurde.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

5.6 L DILG Lade Länge Instanz-DB in AKKU 1

Format

L DILG

Beschreibung

L DILG (Lade die Länge des Instanz-Datenbausteins) lädt die Länge des Instanz-Datenbausteins in AKKU 1, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 gespeichert wurde.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
AUF DI20	//Öffne den Datenbaustein DB20 als Instanz-Datenbaustein.
L DILG	//Lade die Länge des Instanz-Datenbausteins (Länge von DB20).
L MW10	//Vergleichswert, ob der Datenbaustein lang genug ist.
<I	
SPB ERRO	//Springe zur Sprungmarke ERRO, wenn die Länge des Datenbausteins kleiner ist als der Wert in MW10.

5.7 L DINO Lade Nummer Instanz-DB in AKKU 1

Format

L DINO

Beschreibung

L DINO (Lade die Nummer des Instanz-Datenbausteins) lädt die Nummer des geöffneten Instanz-Datenbausteins in AKKU 1, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 gespeichert wurde.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

6 Sprünge

6.1 Sprungoperationen Übersicht

Beschreibung

Sprungoperationen ermöglichen es, den Programmablauf zu unterbrechen, um die Bearbeitung an einem anderen Punkt wiederaufzunehmen. Mit der Operation LOOP können Sie einen Programmteil mehrmals aufrufen.

Als Operand einer Sprungoperation bzw. der Operation LOOP dient eine Sprungmarke. Die Sprungmarke kann aus bis zu vier Zeichen bestehen, von denen das erste Zeichen ein Buchstabe sein muß. Die Sprungmarke endet mit einem Doppelpunkt ":" und leitet die Anweisung in der Zeile ein.

Hinweis

Achten Sie bei Programmen für die S7-300-CPU's darauf, daß bei Sprungoperationen das Sprungziel immer der Beginn einer Verknüpfungskette ist (gilt nicht für die CPU 318-2). Das Sprungziel darf sich nicht innerhalb einer Verknüpfungskette befinden.

Mit den folgenden Operationen können Sie den Programmablauf unabhängig von einer Bedingung unterbrechen:

- SPA Springe absolut
- SPL Sprungleiste

Die folgenden Operationen unterbrechen den Programmablauf. Ihre Ausführung ist vom Verknüpfungsergebnis (VKE) abhängig, das die vorherige Anweisung gebildet hat:

- SPB Springe, wenn VKE = 1
- SPBN Springe, wenn VKE = 0
- SPBB Springe, wenn VKE = 1 und rette VKE ins BIE
- SPBNB Springe, wenn VKE = 0 und rette VKE ins BIE

Die folgenden Operationen unterbrechen den Programmablauf abhängig vom Signalzustand eines Bits im Statuswort:

- SPBI Springe, wenn BIE = 1
- SPBIN Springe, wenn BIE = 0
- SPO Springe, wenn OV = 1
- SPS Springe, wenn OS = 1

Die folgenden Operationen unterbrechen den Programmablauf abhängig vom Ergebnis einer vorhergehenden Operation:

- SPZ Springe, wenn Ergebnis = 0
- SPN Springe, wenn Ergebnis \neq 0
- SPP Springe, wenn Ergebnis > 0
- SPM Springe, wenn Ergebnis < 0
- SPPZ Springe, wenn Ergebnis \geq 0
- SPMZ Springe, wenn Ergebnis \leq 0
- SPU Springe, wenn Ergebnis ungültig

6.2 SPA Sprünge absolut

Format

SPA <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

SPA <Sprungmarke> unterbricht den linearen Programmablauf und springt unabhängig vom Inhalt des Statusworts an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
U	E 1.0	
U	E 1.2	
SPB	DELE	//Sprünge, wenn VKE = 1, zur Sprungmarke DELE.
L	MB10	
INC	1	
T	MB10	
SPA	FORW	//Absoluter Sprung zur Sprungmarke FORW.
DELE:	L 0	
	T MB10	
FORW:	U E 2.1	//Der Programmablauf wird nach dem Sprung zur Sprungmarke FORW hier fortgesetzt.

6.3 SPL Sprungleiste

Format

SPL <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

SPL <Sprungmarke> (Springe über Sprungleiste) ermöglicht das Programmieren von Fallunterscheidungen. Die Zielsprungleiste, die maximal 255 Einträge enthält, beginnt unmittelbar nach der Operation **SPL** und endet vor der Sprungmarke, die der Operand **SPL** angibt. Jedes Sprungziel besteht aus einer Operation **SPA**. Die Anzahl der Sprungziele (0 bis 255) wird dem AKKU1-L-L entnommen.

Solange der AKKU-Inhalt kleiner ist als die Anzahl der Sprungziele zwischen **SPL**-Anweisung und Sprungmarke, springt die Operation **SPL** auf eine der Operationen **SPA**. Wenn AKKU1-L-L = 0 ist, wird zur ersten Operation **SPA** gesprungen, ist AKKU1-L-L = 1, wird zur zweiten Operation **SPA** gesprungen usw. Ist die Anzahl der Sprungziele zu groß, springt die Operation **SPL** zur ersten Anweisung nach der letzten Operation **SPA** in der Zielleiste.

Die Zielsprungleiste muß aus Operationen **SPA** bestehen, die sich vor der Sprungmarke befinden, die vom Operand der Anweisung **SPL** angegeben wird. Andere Operationen innerhalb der Sprungleiste sind unzulässig.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	MB0	//Lade die Nummer des Sprungziels in AKKU1-L-L.
SPL	LSTX	//Sprungziel, wenn AKKU1-L-L > 3 ist.
SPA	SEG0	//Sprungziel, wenn AKKU1-L-L = 0 ist.
SPA	SEG1	//Sprungziel, wenn AKKU1-L-L = 1 ist.
SPA	COMM	//Sprungziel, wenn AKKU1-L-L = 2 ist.
SPA	SEG3	//Sprungziel, wenn AKKU1-L-L = 3 ist.
LSTX:	SPA	COMM
SEG0:	*	//Zulässige Anweisung.
	*	
	SPA	COMM
SEG1:	*	//Zulässige Anweisung.
	*	
	SPA	COMM
SEG3:	*	//Zulässige Anweisung.
	*	
	SPA	COMM
COMM:	*	
	*	

6.4 SPB Sprünge, wenn VKE = 1

Format

SPB <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Wenn VKE = 1, unterbricht **SPB <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms ab (Ein-, Zwei- oder Dreiwortanweisungen).

Wenn VKE = 0, wird der Sprung nicht ausgeführt. Das VKE wird auf "1" gesetzt, und der Programmablauf wird mit der folgenden Anweisung fortgesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	1	0

Beispiel

AWL	Erläuterung
U E 1.0	
U E 1.2	
SPB JOVR	//Sprünge, wenn VKE = 1, zur Sprungmarke JOVR.
L EW8	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
T MW22	
JOVR: U E 2.1	//Der Programmablauf wird nach dem Sprung zur Sprungmarke JOVR hier fortgesetzt.

6.5 SPBN Sprünge, wenn VKE = 0

Format

SPBN <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Wenn VKE = 0, unterbricht **SPBN <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms ab (Ein-, Zwei- oder Dreiwortanweisungen).

Wenn VKE = 1, wird der Sprung nicht ausgeführt. Der Programmablauf wird mit der folgenden Anweisung fortgesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	1	0

Beispiel

AWL	Erläuterung	
U	E 1.0	
U	E 1.2	
SPBN	JOVR	//Sprünge, wenn VKE = 0, zur Sprungmarke JOVR.
L	EW8	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
T	MW22	
JOVR:	U	E 2.1 //Der Programmablauf wird nach dem Sprung zur Sprungmarke JOVR hier fortgesetzt.

6.6 SPBB Sprünge, wenn VKE = 1 und rette VKE ins BIE

Format

SPBB <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Wenn VKE = 1, unterbricht **SPBB <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms ab (Ein-, Zwei- oder Dreiwortanweisungen).

Wenn VKE = 0, wird der Sprung nicht ausgeführt. Das VKE wird auf "1" gesetzt und der Programmablauf wird mit der folgenden Anweisung fortgesetzt.

Unabhängig vom VKE wird bei der Operation **SPBB <Sprungmarke>** das VKE ins BIE kopiert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	x	-	-	-	-	0	1	1	0

Beispiel

AWL	Erläuterung	
U	E 1.0	
U	E 1.2	
SPB	JOVR	//Sprünge, wenn VKE = 1, zur Sprungmarke JOVR. Kopiere Inhalt des VKE-Bit ins BIE-Bit.
B		
L	EW8	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
T	MW22	
JOVR:	U	E 2.1 //Der Programmablauf wird nach dem Sprung zur Sprungmarke JOVR hier fortgesetzt.

6.7 SPBNB Springe, wenn VKE = 0 und rette VKE ins BIE

Format

SPBNB <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Wenn VKE = 0, unterbricht **SPBNB <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Wenn VKE = 1, wird der Sprung nicht ausgeführt. Das VKE wird auf "1" gesetzt, und der Programmablauf wird mit der folgenden Anweisung fortgesetzt.

Unabhängig vom VKE wird bei der Operation **SPBNB <Sprungmarke>** das VKE ins BIE kopiert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	x	-	-	-	-	0	1	1	0

Beispiel

AWL	Erläuterung
U E 1.0	
U E 1.2	
SPB JOVR	//Springe, wenn VKE = 0, zur Sprungmarke JOVR. Kopiere Inhalt des
NB	VKE-Bit ins BIE-Bit.
L EW	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht
	ausgeführt wird.
T MW22	
JOVR: U E 2.1	//Der Programmablauf wird nach dem Sprung zur Sprungmarke JOVR
	hier fortgesetzt.

6.8 SPBI Sprünge, wenn BIE = 1

Format

SPBI <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Ist das Statusbit BIE = 1, unterbricht **SPBI <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms ab (Ein-, Zwei- oder Dreiwortanweisungen).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

6.9 SPBIN Springe, wenn BIE = 0

Format

SPBIN <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Ist das Statusbit BIE = 0, unterbricht **SPBIN <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Die Sprungmarke kann aus bis zu vier Zeichen bestehen, von denen das erste Zeichen ein Buchstabe sein muß. Die Sprungmarke endet mit einem Doppelpunkt ":" und leitet die Anweisung in der Zeile ein. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

6.10 SPO Sprünge, wenn OV = 1

Format

SPO <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Ist das Statusbit OV = 1, unterbricht **SPO <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab. Bei einer kombinierten arithmetischen Operation muß nach jeder einzelnen arithmetischen Operation darauf geachtet werden, daß kein Überlauf auftritt, um sicherzustellen, daß jedes Zwischenergebnis innerhalb des zulässigen Bereichs liegt. Andernfalls ist die Operation SPS zu verwenden.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	MW10	
L	3	
*I		//Multiplikation des Inhalts von MW10 mit "3".
SPO	OVER	//Sprunge, wenn das Ergebnis den maximalen Bereich überschreitet (OV = 1).
T	MW10	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
U	M	
	4.0	
R	M	
	4.0	
SPA	NEXT	
OVER: UN	M	//Der Programmablauf wird nach dem Sprung zur Sprungmarke OVER hier fortgesetzt.
	4.0	
S	M	
	4.0	
NEXT: NOP 0		//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.11 SPS Sprünge, wenn OS = 1

Format

SPS <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Ist das Statusbit OS = 1, unterbricht **SPS <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	-	-	-	-

Beispiel

AWL	Erläuterung	
L	EW10	
L	MW12	
*I		
L	DBW25	
+I		
L	MW14	
-I		
SPS	OVER	//Sprunge, wenn Überlauf in einer der 3 vorhergehenden Operationen, OS = 1, auftritt (siehe Hinweis).
T	MW16	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
U	M 4.0	
R	M 4.0	
SPA	NEXT	
OVER:	UN	M 4.0 //Der Programmablauf wird nach dem Sprung zur Sprungmarke OVER hier fortgesetzt.
S	M 4.0	
NEXT:	NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

Hinweis

In diesem Fall dürfen Sie die Operation **SPO** nicht verwenden. Die Operation **SPO** würde nur den Überlauf der vorangehenden Operation **-I** abfragen.

6.12 SPZ Sprünge, wenn Ergebnis = 0

Format

SPZ <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Sind die Statusbits A1 = 0 und A0 = 0, unterbricht **SPZ <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MW10	
SRW 1	
SPZ ZERO	//Sprünge zur Sprungmarke ZERO, wenn das geschobene Bit = 0 ist.
L MW2	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
INC 1	
T MW2	
SPA NEXT	
ZERO: L MW4	//Der Programmablauf wird nach dem Sprung zur Sprungmarke ZERO hier fortgesetzt.
INC 1	
T MW4	
NEXT: NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.13 SPN Sprünge, wenn Ergebnis $\lt \gt 0$

Format

SPN <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Ist das Ergebnis, das von den Statusbits A1 und A0 angezeigt wird, größer oder kleiner als Null ($A1 = 0/A0 = 1$ oder $A1 = 1/A0 = 0$), unterbricht **SPN <Sprungmarke>** (Sprünge, wenn das Ergebnis $\lt \gt 0$) den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	EW8	
L	MW12	
XOW		
SPN	NOZE	//Sprünge, wenn der Inhalt von AKKU1-L ungleich Null ist.
UN	M 4.0	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
S	M 4.0	
SPA	NEXT	
NOZE:	UN	M 4.1 //Der Programmablauf wird nach dem Sprung zur Sprungmarke NOZE hier fortgesetzt.
S	M 4.1	
NEXT:	NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.14 SPP Sprünge, wenn Ergebnis > 0

Format

SPP <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Sind die Statusbits A1 = 1 und A0 = 0, unterbricht **SPP <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	EW8	
L	MW12	
-I		//Subtraktion des Inhalts von MW12 vom Inhalt von EW8.
SPP	POS	//Sprünge, wenn das Ergebnis > 0 (d. h. der Inhalt von AKKU 1 > 0) ist.
UN	M 4.0	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
S	M 4.0	
SPA	NEXT	
POS:	UN	M 4.1 //Der Programmablauf wird nach dem Sprung zur Sprungmarke POS hier fortgesetzt.
S	M 4.1	
NEXT:	NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.15 SPM Sprünge, wenn Ergebnis < 0

Format

SPM <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Sind die Statusbits A1 = 0 und A0 = 1, unterbricht **SPM <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird an dem Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	EW8	
L	MW12	
-I		//Subtraktion des Inhalt von MW12 vom Inhalt von EW8.
SPM	NEG	//Sprünge, wenn Ergebnis < 0 (d. h. der Inhalt von AKKU 1 < 0) ist.
UN	M 4.0	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
S	M 4.0	
SPA	NEXT	
NEG:	UN	M 4.1 //Der Programmablauf wird nach dem Sprung zur Sprungmarke NEG hier fortgesetzt.
S	M 4.1	
NEXT:	NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.16 SPPZ Sprünge, wenn Ergebnis ≥ 0

Format

SPPZ <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Ist das Ergebnis, das von den Statusbits A1 und A0 angezeigt wird, größer als oder gleich Null ($A1 = 0/A0 = 0$ oder $A1 = 1/A0 = 0$), unterbricht **SPPZ <Sprungmarke>** (Sprünge, wenn das Ergebnis ≥ 0) den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	EW8	
L	MW12	
-I		//Subtraktion des Inhalts von MW12 vom Inhalt von EW8.
SPPZ	REG0	//Sprünge, wenn das Ergebnis ≥ 0 (d. h. der Inhalt von AKKU 1 ≥ 0) ist.
UN	M 4.0	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
S	M 4.0	
SPA	NEXT	
REG0:	UN	M 4.1 //Der Programmablauf wird nach dem Sprung zur Sprungmarke REG0 hier fortgesetzt.
S	M 4.1	
NEXT:	NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.17 SPMZ Sprünge, wenn Ergebnis ≤ 0

Format

SPMZ <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Ist das Ergebnis, das von den Statusbits A1 und A0 angezeigt wird, kleiner als oder gleich Null ($A1 = 0/A0 = 0$ oder $A1 = 0/A0 = 1$), unterbricht **SPMZ <Sprungmarke>** (Sprünge, wenn das Ergebnis ≤ 0) den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	EW8	
L	MW12	
-I		//Subtraktion des Inhalts von MW12 vom Inhalt von EW8.
SPMZ	RGE0	//Sprünge, wenn das Ergebnis ≤ 0 (d. h. der Inhalt von AKKU 1 ≤ 0) ist.
UN	M 4.0	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
S	M 4.0	
SPA	NEXT	
RGE0:	UN	M 4.1 //Der Programmablauf wird nach dem Sprung zur Sprungmarke RGE0 hier fortgesetzt.
S	M 4.1	
NEXT:	NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.18 SPU Sprünge, wenn Ergebnis ungültig

Format

SPU <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

Sind die Statusbits $A1 = 1$ und $A0 = 1$, unterbricht **SPU <Sprungmarke>** den linearen Programmablauf und springt an das Sprungziel. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Das Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Sprunganweisung und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. $+32767$ Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Das Setzen der Statusbits $A1 = 1$ und $A0 = 1$ erfolgt bei

- Division durch Null oder
- Verwendung unzulässiger Operationen oder
- "ungültigem" Ergebnis eines Vergleichs von Gleitpunktzahlen, d. h. bei Verwendung eines ungültigen Formats.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung	
L	MD10	
L	ED2	
/D		//Division des Inhalts von MD10 durch den Inhalt von ED2.
SPU	ERRO	//Springe, wenn Division durch Null (d. h. ED2 = "0").
T	MD14	//Der Programmablauf wird hier fortgesetzt, wenn der Sprung nicht ausgeführt wird.
U	M 4.0	
R	M 4.0	
SPA	NEXT	
ERRO:	UN	M 4.0 //Der Programmablauf wird nach dem Sprung zur Sprungmarke ERRO hier fortgesetzt.
S	M 4.0	
NEXT:	NOP 0	//Der Programmablauf wird nach dem Sprung zur Sprungmarke NEXT hier fortgesetzt.

6.19 LOOP Programmschleife

Format

LOOP <Sprungmarke>

Operand	Beschreibung
<Sprungmarke>	Symbolischer Name des Sprungziels.

Beschreibung

LOOP <Sprungmarke> (Dekrementiere AKKU1-L und springe, wenn AKKU1-L <> 0) vereinfacht die Programmierung von Schleifen. Der Schleifenzähler ist eine vorzeichenlose Ganzzahl (16 Bit) und befindet sich in AKKU1-L. Die Anweisung springt an das angegebene Sprungziel. Der Sprung wird ausgeführt, solange der Inhalt von AKKU1-L ungleich "0" ist. Der lineare Programmablauf wird am Sprungziel fortgesetzt. Dieses Sprungziel wird durch eine Sprungmarke angegeben. Es kann sowohl vorwärts als auch rückwärts gesprungen werden. Sprünge können nur innerhalb eines Bausteins ausgeführt werden, d. h. die Operation Programmschleife und das Sprungziel müssen innerhalb desselben Bausteins liegen. Das Sprungziel darf innerhalb dieses Bausteins nur einmal vorhanden sein. Die maximale Sprungweite liegt bei -32768 bzw. +32767 Wörtern des Programmcodes. Die tatsächliche maximale Anzahl der Anweisungen, die übersprungen werden können, hängt von der Kombination der Anweisungen innerhalb des Programms (Ein-, Zwei- oder Dreiwortanweisungen) ab.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel zur Berechnung der Fakultät von 5 (5!)

AWL	Erläuterung	
L	L#1	//Lade die Ganzzahl-Konstante (32 Bit) in den AKKU 1.
T	MD20	//Transferiere den Inhalt von AKKU 1 in MD20 (Initialisierung).
L	5	//Lade die Anzahl der Schleifenzyklen in AKKU1-L.
NEXT:	T	MW10 //Sprungmarke = Anfang der Schleife / Transferiere AKKU1-L in Schleifenzähler.
L	MD20	
*	D	//Multipliziere aktuellen Inhalt von MD20 mit aktuellem Inhalt von MB10.
T	MD20	//Transferiere Ergebnis der Multiplikation in MD20.
L	MW10	//Lade den Inhalt des Schleifenzählers in AKKU 1.
LOO	NEXT	//Dekrementiere den Inhalt von AKKU 1 und springe zur Sprungmarke
P		NEXT, wenn AKKU1-L > 0 ist.
L	MW24	//Der Programmablauf wird nach dem Ende der Schleife hier fortgesetzt.
L	200	
>I		

7 Festpunkt-Funktionen

7.1 Festpunkt-Funktionen Übersicht

Beschreibung

Festpunkt-Funktionen verknüpfen den Inhalt von AKKU 1 und 2 miteinander. Das Ergebnis wird in AKKU 1 abgelegt. Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Mit den Festpunkt-Funktionen können Sie die folgenden Operationen mit **zwei Ganzzahlen** (16 Bit, 32 Bit) durchführen:

- +I Addiere AKKU 1 und 2 als Ganzzahl (16 Bit)
- -I Subtrahiere AKKU 1 von 2 als Ganzzahl (16 Bit)
- *I Multipliziere AKKU 1 und 2 als Ganzzahl (16 Bit)
- /I Dividiere AKKU 2 durch 1 als Ganzzahl (16 Bit)
- + Addiere Ganzzahlkonstante (16, 32 Bit)

- +D Addiere AKKU 1 und 2 als Ganzzahl (32 Bit)
- -D Subtrahiere AKKU 1 von 2 als Ganzzahl (32 Bit)
- *D Multipliziere AKKU 1 und 2 als Ganzzahl (32 Bit)
- /D Dividiere AKKU 2 durch 1 als Ganzzahl (32 Bit)
- MOD Divisionsrest Ganzzahl (32 Bit)

7.2 Auswerten der Bits im Statuswort bei Festpunkt-Funktionen

Beschreibung

Die Festpunkt-Funktionen beeinflussen die Bits A1, A0, OV und OS im Statuswort.

Die folgenden Tabellen zeigen den Signalzustand der Bits des Statusworts für die Ergebnisse von Operationen mit Festpunktzahlen (16 Bit, 32 Bit).

Gültiger Bereich	A1	A0	OV	OS
0 (Null)	0	0	0	*
16 Bit: $-32\,768 \leq \text{Ergebnis} < 0$ (negative Zahl) 32 Bit: $-2\,147\,483\,648 \leq \text{Ergebnis} < 0$ (negative Zahl)	0	1	0	*
16 Bit: $32\,767 \geq \text{Ergebnis} > 0$ (positive Zahl) 32 Bit: $2\,147\,483\,647 \geq \text{Ergebnis} > 0$ (positive Zahl)	1	0	0	*

* Das OS-Bit wird vom Ergebnis der Operation nicht beeinflusst.

Ungültiger Bereich	A1	A0	OV	OS
Unterschreitung bei Addition 16 Bit: Ergebnis = -65536 32 Bit: Ergebnis = -4 294 967 296	0	0	1	1
Unterschreitung bei Multiplikation 16 Bit: Ergebnis < -32 768 (negative Zahl) 32 Bit: Ergebnis < -2 147 483 648 (negative Zahl)	0	1	1	1
Überlauf bei Addition, Subtraktion 16 Bit: Ergebnis > 32 767 (positive Zahl) 32 Bit: Ergebnis > 2 147 483 647 (positive Zahl)	0	1	1	1
Überlauf bei Multiplikation, Division 16 Bit: Ergebnis > 32 767 (positive Zahl) 32 Bit: Ergebnis > 2 147 483 647 (positive Zahl)	1	0	1	1
Unterschreitung bei Addition, Subtraktion 16 Bit: Ergebnis < -32 768 (negative Zahl) 32 Bit: Ergebnis < -2 147 483 648 (negative Zahl)	1	0	1	1
Division durch 0	1	1	1	1

Operation	A1	A0	OV	OS
+D: Ergebnis = -4 294 967 296	0	0	1	1
/D oder MOD: Division durch 0	1	1	1	1

7.3 +I Addiere AKKU 1 und 2 als Ganzzahl (16 Bit)

Format

+I

Beschreibung

+I (Addiere Ganzzahlen, 16 Bit) addiert den Inhalt von AKKU1-L zum Inhalt von AKKU2-L und speichert das Ergebnis in AKKU1-L. Die Inhalte von AKKU1-L und AKKU2-L werden als Ganzzahlen (16 Bit) ausgewertet. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS, und OV werden als Ergebnis der Operation gesetzt. Bei Überlauf/Unterlauf ist das Ergebnis der Operation keine Ganzzahl (32 Bit), sondern eine Ganzzahl (16 Bit).

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Summe	=	0		0	0	0	-
-32768	<=	Summe <	0	0	1	0	-
32767	>=	Summe >	0	1	0	0	-
Summe	=	-65536		0	0	1	1
65534	>=	Summe >	32767	0	1	1	1
-65535	<=	Summe <	-32768	1	0	1	1

Beispiel

AWL	Erläuterung
L EW10	//Der Wert von EW10 wird in AKKU1-L geladen.
L MW14	//Lade den Inhalt von AKKU1-L in AKKU2-L. Lade den Wert von MW14 in AKKU1-L.
+I	//Addiere AKKU2-L und AKKU1-L, speichere das Ergebnis in AKKU1-L.
T DB1.DBW25	//Der Inhalt von AKKU1-L (Ergebnis) wird nach DBW25 von DB1 transferiert.

7.4 -I Subtrahiere AKKU 1 von 2 als Ganzzahl (16 Bit)

Format

-I

Beschreibung

-I (Subtrahiere Ganzzahlen, 16 Bit) subtrahiert den Inhalt von AKKU1-L vom Inhalt von AKKU2-L und speichert das Ergebnis in AKKU1-L. Die Inhalte von AKKU1-L und AKKU2-L werden als Ganzzahlen (16 Bit) ausgewertet. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS, und OV werden als Ergebnis der Operation gesetzt. Bei Überlauf/Unterlauf ist das Ergebnis der Operation keine Ganzzahl (32 Bit), sondern eine Ganzzahl (16 Bit).

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Differenz	=	0		0	0	0	-
-32768	<=	Differenz	< 0	0	1	0	-
32767	>=	Differenz	> 0	1	0	0	-
65535	>=	Differenz	> 32767	0	1	1	1
-65535	<=	Differenz	< -32768	1	0	1	1

Beispiel

AWL	Erläuterung
L EW10	//Der Wert von EW10 wird in AKKU1-L geladen.
L MW14	//Lade den Inhalt von AKKU1-L in AKKU2-L. Lade den Wert von MW14 in AKKU1-L.
-I	//Subtrahiere AKKU1-L von AKKU2-L, speichere das Ergebnis in AKKU1-L.
T DB1.DBW25	//Der Inhalt von AKKU1-L (Ergebnis) wird nach DBW25 von DB1 transferiert.

7.5 *I Multipliziere AKKU 1 und 2 als Ganzzahl (16 Bit)

Format

*I

Beschreibung

*I (Multipliziere Ganzzahlen, 16 Bit) multipliziert den Inhalt von AKKU2-L mit dem Inhalt von AKKU1-L. Die Inhalte von AKKU1-L und AKKU2-L werden als Ganzzahlen (16 Bit) ausgewertet. Das Ergebnis wird als Ganzzahl (32 Bit) in AKKU 1 gespeichert. Sind die Statusbits OV = 1 und OS = 1, liegt das Ergebnis außerhalb des Bereichs einer Ganzzahl (16 Bit).

Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Produkt	=	0		0	0	0	-
-32768	<=	Produkt <	0	0	1	0	-
32767	>=	Produkt >	0	1	0	0	-
1.073.741.824	>=	Produkt >	32767	1	0	1	1
-1.073.709.056	<=	Produkt <	-32768	0	1	1	1

Beispiel

AWL	Erläuterung
L EW10	//Der Wert von EW10 wird in AKKU1-L geladen.
L MW14	//Lade den Inhalt von AKKU1-L in AKKU 2-L. Lade den Wert von MW14 in AKKU1-L.
*I	//Multipliziere AKKU2-L und AKKU1-L, speichere das Ergebnis in AKKU 1.
T DB1.DBD25	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD25 in DB1 transferiert.

7.6 /I Dividiere AKKU 2 durch 1 als Ganzzahl (16 Bit)

Format

/I

Beschreibung

/I (Dividiere Ganzzahlen, 16 Bit) dividiert den Inhalt von AKKU2-L durch den Inhalt von AKKU1-L. Die Inhalte von AKKU1-L und AKKU2-L werden als Ganzzahlen (16 Bit) ausgewertet. Das Ergebnis wird in AKKU 1 gespeichert und besteht aus zwei Ganzzahlen (16 Bit), dem Quotienten und dem Divisionsrest. Der Quotient wird in AKKU1-L gespeichert und der Divisionsrest in AKKU1-H. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Quotient = 0				0	0	0	-
-32768	<=	Quotient <	0	0	1	0	-
32767	>=	Quotient >	0	1	0	0	-
Quotient	=	32768		1	0	1	1
Division durch Null				1	1	1	1

Beispiel

AWL	Erläuterung	
L	EW10	//Der Wert von EW10 wird in AKKU1-L geladen.
L	MW14	//Lade den Inhalt von AKKU1-L in AKKU2-L. Lade den Wert von MW14 in AKKU1-L.
/I		//Dividiere AKKU2-L durch AKKU1-L, speichere das Ergebnis in AKKU 1: AKKU1-L: Quotient, AKKU1-H: Divisionsrest
T	MD20	//Der Inhalt von AKKU 1 (Ergebnis) wird nach MD20 transferiert.

Beispiel: "13 dividiert durch 4"

Inhalt von AKKU2-L vor der Operation (EW10):	"13"
Inhalt von AKKU1-L vor der Operation (MW14):	"4"
Operation /I (AKKU2-L / AKKU1-L):	"13/4"
Inhalt von AKKU1-L nach der Operation (Quotient):	"3"
Inhalt von AKKU1-H nach der Operation (Divisionsrest):	"1"

7.7 + Addiere Ganzzahlkonstante (16, 32 Bit)

Format

+ <Ganzzahlkonstante>

Operand	Datentyp	Beschreibung
<Ganzzahlkonstante>	Konstante, (16 bzw. 32 Bit)	Konstante, die addiert werden soll

Beschreibung

+ <Ganzzahlkonstante> addiert die Ganzzahlkonstante zum Inhalt von AKKU 1 und speichert das Ergebnis in AKKU 1. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

+ <Ganzzahlkonstante, 16 Bit> addiert eine Ganzzahlkonstante (16 Bit) (in dem Bereich von -32768 bis +32767) zum Inhalt von AKKU1-L und speichert das Ergebnis in AKKU1-L.

+ <Ganzzahlkonstante, 32 Bit> addiert eine Ganzzahlkonstante (32 Bit) (in dem Bereich von -2.147.483.648 bis 2.147.483.647) zum Inhalt von AKKU 1 und speichert das Ergebnis in AKKU 1.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel 1

AWL	Erläuterung
L EW10	//Lade den Wert von EW10 in AKKU1-L.
L MW14	//Lade den Inhalt von AKKU1-L in AKKU2-L. Lade den Wert von MW14 in AKKU1-L.
+I	//Addiere AKKU2-L und AKKU1-L, speichere das Ergebnis in AKKU1-L.
+ 25	//Addiere AKKU1-L und 25, speichere das Ergebnis in AKKU1-L.
T DB1.DBW25	//Transferiere den Inhalt von AKKU1-L (Ergebnis) nach DBW25 von DB 1.

Beispiel 2

AWL	Erläuterung
L EW12	
L EW14	
+ 100	//Addiere AKKU1-L und 100, speichere das Ergebnis in AKKU 1-L.
>I	//Ist AKKU 2 > AKKU 1 bzw. gilt EW 12 > (EW14 + 100),
SPB NEXT	//dann springe zur Sprungmarke NEXT.

Beispiel 3

AWL	Erläuterung
L MD20	
L MD24	
+D	//Addiere AKKU 1 und AKKU 2, speichere das Ergebnis in AKKU 1.
+ L#-200	//Addiere AKKU1 und -200, speichere das Ergebnis in AKKU 1.
T MD28	

7.8 +D Addiere AKKU 1 und 2 als Ganzzahl (32 Bit)

Format

+D

Beschreibung

+D (Addiere Ganzzahlen, 32 Bit) addiert den Inhalt von AKKU 1 zum Inhalt von AKKU 2 und speichert das Ergebnis in AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Ganzzahlen (32 Bit) ausgewertet. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Summe	=	0		0	0	0	-
-2.147.483.648	<=	Summe <	0	0	1	0	-
2.147.483.647	>=	Summe >	0	1	0	0	-
Summe	=	-4.294.967.296		0	0	1	1
4.294.967.294	>=	Summe >	2.147.483.647	0	1	1	1
-4.294.967.295	<=	Summe <	-2.147.483.648	1	0	1	1

Beispiel

AWL	Erläuterung
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
+D	//Addiere AKKU 2 und AKKU 1, speichere das Ergebnis in AKKU 1.
T DB1.DBD25	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD25 von DB1 transferiert.

7.9 -D Subtrahiere AKKU 1 von 2 als Ganzzahl (32 Bit)

Format

-D

Beschreibung

-D (Subtrahiere Ganzzahlen, 32 Bit) subtrahiert den Inhalt von AKKU 1 vom Inhalt von AKKU 2 und speichert das Ergebnis in AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Ganzzahlen (32 Bit) ausgewertet. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Differenz	=	0		0	0	0	-
-2.147.483.648	<=	Differenz < 0		0	1	0	-
2.147.483.647	>=	Differenz > 0		1	0	0	-
4.294.967.295	>=	Differenz > 2.147.483.647		0	1	1	1
-4.294.967.295	<=	Differenz < -2.147.483.648		1	0	1	1

Beispiel

AWL	Erläuterung
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
-D	//Subtrahiere AKKU 1 von AKKU 2, speichere das Ergebnis in AKKU 1.
T DB1.DBD25	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD25 in DB1 transferiert.

7.10 *D Multipliziere AKKU 1 und 2 als Ganzzahl (32 Bit)

Format

*D

Beschreibung

*D (Multipliziere Ganzzahlen, 32 Bit) multipliziert den Inhalt von AKKU 1 mit dem Inhalt von AKKU 2. Die Inhalte von AKKU 1 und AKKU 2 werden als Ganzzahlen (32 Bit) ausgewertet. Das Ergebnis wird als Ganzzahl (32 Bit) in AKKU 1 gespeichert. Sind die Statusbits OV1 = 1 und OS = 1, liegt das Ergebnis außerhalb des Bereichs einer Ganzzahl (32 Bit).

Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Produkt	=	0		0	0	0	-
-2.147.483.648	<=	Produkt	<	0	0	1	-
2.147.483.647	>=	Produkt	>	0	1	0	-
Produkt	>	2.147.483.647		1	0	1	1
Produkt	<	-2.147.483.648		0	1	1	1

Beispiel

AWL	Erläuterung
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
*D	//Multipliziere AKKU 2 und AKKU 1, speichere das Ergebnis in AKKU 1.
T DB1.DBD25	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD25 in DB1 transferiert.

7.11 /D Dividiere AKKU 2 durch 1 als Ganzzahl (32 Bit)

Format

/D

Beschreibung

/D (Dividiere Ganzzahlen, 32 Bit) dividiert den Inhalt von AKKU 2 durch den Inhalt von AKKU1. Die Inhalte von AKKU 1 und AKKU 2 werden als Ganzzahlen (32 Bit) ausgewertet. Das Ergebnis wird in AKKU 1 gespeichert. Das Ergebnis enthält nur den Quotienten, nicht den Divisionsrest (mit der Operation **MOD** erhalten Sie den Divisionsrest).

Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Quotient = 0				0	0	0	-
-2147483648	<=	Quotient <	0	0	1	0	-
2147483647	>=	Quotient >	0	1	0	0	-
Quotient	=	2147483648		1	0	1	1
Division durch Null				1	1	1	1

Beispiel

AWL	Erläuterung
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
/D	//Dividiere AKKU 2 durch AKKU 1, speichere das Ergebnis (Quotient) in AKKU 1.
T MD20	//Der Inhalt von AKKU 1 (Ergebnis) wird nach MD20 transferiert.

Beispiel: "13 dividiert durch 4"

Inhalt von AKKU 2 vor der Operation (ED10):	"13"
Inhalt von AKKU 1 vor der Operation (MD14):	"4"
Operation /D (AKKU 2 / AKKU 1):	"13/4"
Inhalt von AKKU 1 nach der Operation (Quotient):	"3"

7.12 MOD Divisionsrest Ganzzahl (32 Bit)

Format

MOD

Beschreibung

MOD (Divisionsrest von Ganzzahlen, 32 Bit) dividiert den Inhalt von AKKU 2 durch den Inhalt von AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Ganzzahlen (32 Bit) ausgewertet. Das Ergebnis wird in AKKU 1 gespeichert. Das Ergebnis enthält nur den Divisionsrest, nicht den Quotienten (mit der Operation **/D** erhalten Sie den Quotienten).

Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Bilden der Statusbits				A1	A0	OV	OS
Rest	=	0		0	0	0	-
-2147483648	<=	Rest	<	0	0	1	-
2147483647	>=	Rest	>	0	1	0	-
Division durch Null				1	1	1	1

Beispiel

AWL	Erläuterung
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
MOD	//Dividiere AKKU 2 durch AKKU 1, speichere das Ergebnis (Divisionsrest) in AKKU 1.
T MD20	//Der Inhalt von AKKU 1 (Ergebnis) wird nach MD20 transferiert.

Beispiel: "13 dividiert durch 4"

Inhalt von AKKU 2 vor der Operation (ED10):	"13"
Inhalt von AKKU 1 vor der Operation (MD14):	"4"
Operation /D (AKKU 2 / AKKU 1):	"13/4"
Inhalt von AKKU 1 nach der Operation (Divisionsrest):	"1"

8 Gleitpunkt-Funktionen

8.1 Gleitpunkt-Funktionen Übersicht

Beschreibung

Gleitpunkt-Funktionen verknüpfen den Inhalt von AKKU 1 und 2 miteinander. Das Ergebnis wird in AKKU 1 abgelegt. Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Gleitpunkt-Zahlen gehören zum Datentyp REAL ("R"). Mit den Gleitpunkt-Funktionen können Sie die folgenden Operationen mit **zwei Gleitpunktzahlen** (32 Bit, IEEE-FP) ausführen:

- +R Addiere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
- -R Subtrahiere AKKU 1 von 2 als Gleitpunktzahl (32 Bit)
- *R Multipliziere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
- /R Dividiere AKKU 2 durch 1 als Gleitpunktzahl (32 Bit)

Folgende Operationen können Sie mit **einer Gleitpunktzahl** (32 Bit, IEEE-FP) ausführen:

- ABS Absolutwert einer Gleitpunktzahl (32 Bit, IEEE-FP)
- SQR Bilden des Quadrats einer Gleitpunktzahl (32 Bit)
- SQRT Bilden der Quadratwurzel einer Gleitpunktzahl (32 Bit)
- EXP Bilden des Exponentialwerts einer Gleitpunktzahl (32 Bit)
- LN Bilden des natürlichen Logarithmus einer Gleitpunktzahl (32 Bit)
- SIN Bilden des Sinus eines Winkels als Gleitpunktzahlen (32 Bit)
- COS Bilden des Cosinus eines Winkels als Gleitpunktzahlen (32 Bit)
- TAN Bilden des Tangens eines Winkels als Gleitpunktzahlen (32 Bit)
- ASIN Bilden des Arcussinus einer Gleitpunktzahl (32 Bit)
- ACOS Bilden des Arcuscosinus einer Gleitpunktzahl (32 Bit)
- ATAN Bilden des Arcustangens einer Gleitpunktzahl (32 Bit)

8.2 Auswerten der Bits im Statuswort bei Gleitpunkt-Funktionen

Beschreibung

Die Gleitpunkt-Funktionen beeinflussen die Bits A1, A0, OV und OS im Statuswort.

Die folgenden Tabellen zeigen den Signalzustand der Bits im Statuswort für die Ergebnisse von Operationen mit Gleitpunktzahlen (32 Bit).

Gültiger Bereich	A1	A0	OV	OS
+0, -0 (Null)	0	0	0	*
$-3,402823E+38 < \text{Ergebnis} < -1,175494E-38$ (negative Zahl)	0	1	0	*
$+1,175494E-38 < \text{Ergebnis} < 3,402824E+38$ (positive Zahl)	1	0	0	*

* Das OS-Bit wird vom Ergebnis der Operation nicht beeinflusst.

Ungültiger Bereich	A1	A0	OV	OS
Unterschreitung $-1,175494E-38 < \text{Ergebnis} < -1,401298E-45$ (negative Zahl)	0	0	1	1
Unterschreitung $+1,401298E-45 < \text{Ergebnis} < +1,175494E-38$ (positive Zahl)	0	0	1	1
Überlauf Ergebnis $< -3,402823E+38$ (negative Zahl)	0	1	1	1
Überlauf Ergebnis $> 3,402823E+38$ (positive Zahl)	1	0	1	1
keine gültige Gleitpunktzahl oder unzulässige Operation (Eingangswert außerhalb des gültigen Wertebereichs)	1	1	1	1

8.3 Grundoperationen

8.3.1 +R Addiere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)

Format

+R

Beschreibung

+R (Addiere Gleitpunktzahlen, 32-Bit, IEEE-FP) addiert den Inhalt von AKKU 1 zum Inhalt von AKKU 2 und speichert das Ergebnis in AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Gleitpunktzahlen (32-Bit, IEEE-FP) ausgewertet. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-unendlich	0	1	1	1	Überlauf
-qNaN	1	1	1	1	

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
AUF DB10	
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
+R	//Addiere AKKU 2 und AKKU 1, speichere das Ergebnis in AKKU 1.
T DBD25	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD25 in DB10 transferiert.

8.3.2 -R Subtrahiere AKKU 1 von 2 als Gleitpunktzahl (32 Bit)

Format

-R

Beschreibung

-R (Subtrahiere Gleitpunktzahlen, 32-Bit, IEEE-FP) subtrahiert den Inhalt von AKKU 1 vom Inhalt von AKKU 2 und speichert das Ergebnis in AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Gleitpunktzahlen (32-Bit, IEEE-FP) ausgewertet. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-unendlich	0	1	1	1	Überlauf
-qNaN	1	1	1	1	

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
AUF DB10	
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
-R	//Subtrahiere AKKU 2 von AKKU 1, speichere das Ergebnis in AKKU 1.
T DBD25	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD25 in DB10 transferiert.

8.3.3 *R Multipliziere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)

Format

*R

Beschreibung

*R (Multipliziere Gleitpunktzahlen, 32-Bit, IEEE-FP) multipliziert den Inhalt von AKKU 2 mit dem Inhalt von AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Gleitpunktzahlen (32-Bit, IEEE-FP) ausgewertet. Das Ergebnis wird als Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1 gespeichert. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-unendlich	0	1	1	1	Überlauf
-qNaN	1	1	1	1	

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
OPN DB10	
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
*R	//Multipliziere AKKU 2 und AKKU 1, speichere das Ergebnis in AKKU 1.
T DBD25	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD25 in DB10 transferiert.

8.3.4 /R Dividiere AKKU 2 durch 1 als Gleitpunktzahl (32 Bit)

Format

/R

Beschreibung

/R (Dividiere Gleitpunktzahlen, 32-Bit, IEEE-FP) dividiert den Inhalt von AKKU 2 durch den Inhalt von AKKU 1. Die Inhalte von AKKU 1 und AKKU 2 werden als Gleitpunktzahlen (32-Bit, IEEE-FP) ausgewertet. Die Operation wird ausgeführt, ohne das VKE zu berücksichtigen oder zu beeinflussen. Die Statusbits A1, A0, OS und OV werden als Ergebnis der Operation gesetzt.

Bei CPUs mit zwei Akkus bleibt der Inhalt von AKKU 2 unverändert.

Bei CPUs mit vier Akkus werden die Inhalte von AKKU 3 in AKKU 2, und von AKKU 4 in AKKU 3 kopiert. Der Inhalt von AKKU 4 bleibt unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-unendlich	0	1	1	1	Überlauf
-qNaN	1	1	1	1	

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	x	-	-	-	-

Beispiel

AWL	Erläuterung
AUF DB10	
L ED10	//Der Wert von ED10 wird in AKKU 1 geladen.
L MD14	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD14 in AKKU 1.
/R	//Dividiere AKKU 2 durch AKKU 1, speichere das Ergebnis in AKKU 1.
T DBD20	//Der Inhalt von AKKU 1 (Ergebnis) wird nach DBD20 in DB10 transferiert.

8.3.5 ABS Absolutwert einer Gleitpunktzahl (32 Bit, IEEE-FP)

Format

ABS

Beschreibung

ABS (Absolutwert einer Gleitpunktzahl, 32-Bit, IEEE-FP) bildet den Absolutwert einer Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L ED8	//Lade den Wert in AKKU 1 (Beispiel: ED8 = -1.5E+02).
ABS	//Bilde den Absolutwert, speichere das Ergebnis in AKKU 1.
T MD10	//Transferiere das Ergebnis nach MD10 (Beispiel: Ergebnis = 1.5E+02).

8.4 Erweiterte Operationen

8.4.1 SQR Bilden des Quadrats einer Gleitpunktzahl (32 Bit)

Format

SQR

Beschreibung

SQR (Bilden des Quadrats einer Gleitpunktzahl, 32-Bit, IEEE-FP) berechnet das Quadrat einer Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
AUF DB17	//Schlage Datenbaustein DB17 auf.
L DBD0	//Der Wert aus Datendoppelwort DBD0 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
SQR	//Berechne das Quadrat der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation SQR kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation SQR ein Fehler aufgetreten ist.
OK: T DBD4	//Transferiere das Ergebnis aus AKKU 1 in das Datendoppelwort DBD4.

8.4.2 SQRT Bilden der Quadratwurzel einer Gleitpunktzahl (32 Bit)

Format

SQRT

Beschreibung

SQRT (Bilden der Quadratwurzel einer Gleitpunktzahl, 32-Bit, IEEE-FP) berechnet die Quadratwurzel einer Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Das Ergebnis wird in AKKU 1 gespeichert. Der Eingangswert muß größer oder gleich Null sein. Das Ergebnis ist dann positiv. Einzige Ausnahme: die Quadratwurzel von -0 ist -0. Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und die CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
SQRT	//Berechne die Quadratwurzel der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation SQRT kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation SQRT ein Fehler aufgetreten ist.
OK: T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.3 EXP Bilden des Exponentialwerts einer Gleitpunktzahl (32 Bit)

Format

EXP

Beschreibung

EXP (Bilden des Exponentialwerts einer Gleitpunktzahl, 32-Bit, IEEE-FP) berechnet den Exponentialwert (Exponentialwert zur Basis e) einer Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
EXP	//Berechne den Exponentialwert der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1 zur Basis e. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation EXP kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation EXP ein Fehler aufgetreten ist.
OK: T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.4 LN Bilden des natürlichen Logarithmus einer Gleitpunktzahl (32 Bit)

Format

LN

Beschreibung

LN (Bilden des natürlichen Logarithmus einer Gleitpunktzahl, 32-Bit, IEEE-FP) berechnet den natürlichen Logarithmus (Logarithmus zur Basis e) einer Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Das Ergebnis wird in AKKU 1 gespeichert. Der Eingangswert muß größer Null sein. Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-unendlich	0	1	1	1	Überlauf
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
LN	//Berechne den natürlichen Logarithmus der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation LN kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation LN ein Fehler aufgetreten ist.
OK: T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.5 SIN Bilden des Sinus eines Winkels als Gleitpunktzahlen (32 Bit)

Format

SIN

Beschreibung

SIN (Bilden des Sinus von Winkeln als Gleitpunktzahlen, 32-Bit, IEEE-FP) berechnet den Sinus von einem Winkel, der im Bogenmaß angegeben wird. Der Winkel muß als Gleitpunktzahl in AKKU 1 vorliegen. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Überlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
SIN	//Berechne den Sinus der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.6 COS Bilden des Cosinus eines Winkels als Gleitpunktzahlen (32 Bit)

Format

COS

Beschreibung

COS (Bilden des Cosinus von Winkeln als Gleitpunktzahlen, 32-Bit, IEEE-FP) berechnet den Cosinus von einem Winkel, der im Bogenmaß angegeben wird. Der Winkel muß als Gleitpunktzahl in AKKU 1 vorliegen. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Überlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
COS	//Berechne den Cosinus der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.7 TAN Bilden des Tangens eines Winkels als Gleitpunktzahlen (32 Bit)

Format

TAN

Beschreibung

TAN (Bilden des Tangens von Winkeln als Gleitpunktzahlen, 32-Bit, IEEE-FP) berechnet den Tangens von einem Winkel, der im Bogenmaß angegeben wird. Der Winkel muß als Gleitpunktzahl in AKKU 1 vorliegen. Das Ergebnis wird in AKKU 1 gespeichert. Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+unendlich	1	0	1	1	Überlauf
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Unterlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-unendlich	0	1	1	1	Überlauf
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
TAN	//Berechne den Tangens der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation TAN kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation TAN ein Fehler aufgetreten ist.
OK: T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.8 ASIN Bilden des Arcussinus einer Gleitpunktzahl (32 Bit)

Format

ASIN

Beschreibung

ASIN (Bilden des Arcussinus einer Gleitpunktzahl, 32-Bit, IEEE-FP) berechnet den Arcussinus einer Gleitpunktzahl in AKKU 1. Zulässiger Wertebereich für den Eingangswert:

$$-1 \leq \text{Eingangswert} \leq +1$$

Das Ergebnis ist ein Winkel, der im Bogenmaß angegeben wird. Der Wert liegt in dem folgenden Bereich:

$$-\pi / 2 \leq \text{Arcussinus (AKKU 1)} \leq +\pi / 2, \text{ mit } \pi = 3,14159\dots$$

Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Überlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
ASIN	//Berechne den Arcussinus der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation ASIN kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation ASIN ein Fehler aufgetreten ist.
OK: T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.9 ACOS Bilden des Arcuscosinus einer Gleitpunktzahl (32 Bit)

Format

ACOS

Beschreibung

ACOS (Bilden des Arcuscosinus einer Gleitpunktzahl, 32-Bit, IEEE-FP) berechnet den Arcuscosinus einer Gleitpunktzahl in AKKU 1. Zulässiger Wertebereich für den Eingangswert:

$$-1 \leq \text{Eingangswert} \leq +1$$

Das Ergebnis ist ein Winkel, der im Bogenmaß angegeben wird. Der Wert liegt in dem folgenden Bereich:

$$0 \leq \text{Arcuscosinus (AKKU 1)} \leq \pi, \text{ mit } \pi = 3,14159\dots$$

Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Überlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
ACOS	//Berechne den Arcuscosinus der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation ACOS kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation ACOS ein Fehler aufgetreten ist.
OK: T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

8.4.10 ATAN Bilden des Arcustangens einer Gleitpunktzahl (32 Bit)

Format

ATAN

Beschreibung

ATAN (Bilden des Arcustangens einer Gleitpunktzahl, 32-Bit, IEEE-FP) berechnet den Arcustangens einer Gleitpunktzahl in AKKU 1. Das Ergebnis ist ein Winkel, der im Bogenmaß angegeben wird. Der Wert liegt in dem folgenden Bereich:

$$-\pi / 2 \leq \text{Arcustangens (AKKU 1)} \leq +\pi / 2, \text{ mit } \pi = 3,14159\dots$$

Die Operation beeinflusst die Bits A1, A0, OV und OS des Statusworts.

Die Inhalte von AKKU 2 (und bei CPUs mit vier Akkus auch die Inhalte von AKKU 3 und AKKU 4) bleiben unverändert.

Ergebnis

Ergebnis in AKKU 1	A1	A0	OV	OS	Anmerkung
+qNaN	1	1	1	1	
+normalisiert	1	0	0	-	
+denormalisiert	0	0	1	1	Überlauf
+Null	0	0	0	-	
-Null	0	0	0	-	
-denormalisiert	0	0	1	1	Unterlauf
-normalisiert	0	1	0	-	
-qNaN	1	1	1	1	

Beispiel

AWL	Erläuterung
L MD10	//Der Wert aus Merkerdoppelwort MD10 wird in AKKU 1 geladen. (Dieser Wert muss Gleitpunktformat haben.)
ATAN	//Berechne den Arcustangens der Gleitpunktzahl (32-Bit, IEEE-FP) in AKKU 1. Lege das Ergebnis in AKKU 1 ab.
UN OV	//Frage Bit OV im Statuswort auf "0" ab.
SPB OK	//Falls bei der Operation ATAN kein Fehler aufgetreten ist, springe zur Marke OK.
BEA	//BEA, falls bei der Operation ATAN ein Fehler aufgetreten ist.
OK: T MD20	//Transferiere das Ergebnis aus AKKU 1 in das Merkerdoppelwort MD20.

9 Laden/Transferieren

9.1 Lade- und Transferoperationen Übersicht

Beschreibung

Die Lade- und Transferoperationen ermöglichen es Ihnen, den Informationsaustausch zwischen Ein- oder Ausgabebaugruppen und Speicherbereichen oder zwischen Speicherbereichen zu programmieren. Die CPU führt diese Operationen in jedem Zyklus als unbedingte Operationen aus, d. h. sie werden vom Verknüpfungsergebnis einer Operation nicht beeinflusst.

Folgende Lade- und Transferoperationen stehen Ihnen zur Verfügung:

- L Lade
- L STW Lade Statuswort in AKKU 1
- LAR1 Lade Adreßregister 1 mit Inhalt von AKKU 1
- LAR1 <D> Lade Adreßregister 1 mit Pointer (32 Bit-Format)
- LAR1 AR2 Lade Adreßregister 1 mit Inhalt von Adressregister 2
- LAR2 Lade Adreßregister 2 mit Inhalt von AKKU 1
- LAR2 <D> Lade Adreßregister 2 mit Ganzzahl (32 Bit)

- T Transferiere
- T STW Transferiere AKKU 1 in Statuswort
- TAR Tausche Adreßregister 1 mit 2
- TAR1 Transferiere Adreßregister 1 in AKKU 1
- TAR1 <D> Transferiere Adreßregister 1 nach Zieladresse (32-Bit-Pointer)
- TAR1 AR2 Transferiere Adreßregister 1 in Adreßregister 2
- TAR2 Transferiere Adreßregister 2 in AKKU 1
- TAR2 <D> Transferiere Adreßregister 2 nach Zieladresse (32-Bit-Pointer)

9.2 L Lade

Format

L <Operand>

Operand	Datentyp	Speicherbereich	Quelladresse
<Operand>	BYTE	E, A, PE, M, L, D, Pointer, Parameter	0...65535
	WORD		0...65534
	DWORD		0...65532

Beschreibung

L <Operand> lädt den Inhalt des adressierten Bytes, Wortes oder Doppelwortes in AKKU 1, nachdem zuvor der alte Inhalt von AKKU 1 in AKKU 2 gespeichert wurde und AKKU 1 auf "0" zurückgesetzt wurde.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L EB10	//Lade Eingangsbyte EB10 in AKKU1-L-L.
L MB120	//Lade Merkerbyte MB120 in AKKU1-L-L.
L DBB12	//Lade Datenbyte DBB12 in AKKU1-L-L.
L DIW15	//Lade Instanzdatenwort DIW15 in AKKU1-L.
L LD252	//Lade Lokaldaten-Doppelwort LD252 in AKKU 1.
L P# E 8.7	//Lade den Pointer in AKKU1
L OTTO	//Lade den Parameter "OTTO" in AKKU1
L P# ANNA	//Lade den Pointer auf den angegebenen Parameter in AKKU1 (Dieser Befehl lädt den relativen Adressoffset des angegebenen Parameters. Um in multiinstanzfähigen FBs den absoluten Offset im Instanz-Datenbaustein zu ermitteln, muss zu diesem Wert noch der Inhalt des AR2-Registers addiert werden.

Inhalt von Akkumulator 1

Inhalt von AKKU 1	AKKU1-H-H	AKKU1-H-L	AKKU1-L-H	AKKU1-L-L
vor Ausführung der Ladeoperation	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
nach Ausführung von L MB10 (L <Byte>)	00000000	00000000	00000000	<MB10>
nach Ausführung von L MW10 (L <Wort>)	00000000	00000000	<MB10>	<MB11>
nach Ausführung von L MD10 (L <Doppelwort>)	<MB10>	<MB11>	<MB12>	<MB13>
nach Ausführung von L P# ANNA (im FB)	<86>	<Bit-Offset von ANNA relativ zum FB-Anfang>. Um in multiinstanzfähigen FBs den absoluten Offset im Instanz-Datenbaustein zu ermitteln, muß zu diesem Wert noch der Inhalt des AR2-Registers addiert werden.		
nach Ausführung von L P# ANNA (im FC)	<eine bereichsübergreifende Adresse des Datums, das an ANNA übergeben wird>			
	X = "1" oder "0"			

9.3 L STW Lade Statuswort in AKKU 1

Format

L STW

Beschreibung

L STW (Operation L mit dem Operand STW) lädt AKKU 1 mit dem Inhalt des Statusworts. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Hinweis

Bei den CPUs der Familie S7-300 werden die Bits des Statusworts /ER, STA und OR nicht durch die Anweisung **L STW** geladen. Lediglich Bit 1, 4, 5, 6, 7 und 8 werden an die entsprechenden Bitpositionen des niederwertigen Worts von AKKU 1 geladen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L STW	//Lade den Inhalt des Statusworts in AKKU 1.

Der Inhalt von AKKU 1 nach der Ausführung von **L STW** lautet:

Bit	31-9	8	7	6	5	4	3	2	1	0
Inhalt:	0	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER

9.4 LAR1 Lade Adreßregister 1 mit Inhalt von AKKU 1

Format

LAR1

Beschreibung

LAR1 lädt das Adreßregister AR1 mit dem Inhalt von AKKU 1 (32 Bit-Pointer). AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

9.5 LAR1 <D> Lade Adreßregister 1 mit Pointer (32 Bit-Format)

Format

LAR1 <D>

Operand	Datentyp	Speicherbereich	Quelladresse
<D>	DWORD Pointerkonstante	D, M, L	0...65532

Beschreibung

LAR1 <D> lädt das Adreßregister AR1 mit dem Inhalt des adressierten Doppelworts <D> oder einer Pointerkonstante. AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel: Direkte Adressen

AWL	Erläuterung
LAR1 DBD 20	//Lade AR 1 mit dem Pointer in Datendoppelwort DBD20.
LAR1 DID 30	//Lade AR1 mit dem Pointer in Instanzdoppelwort DID30.
LAR1 LD 180	//Lade AR1 mit dem Pointer in Lokaldaten-Doppelwort LD180.
LAR1 MD 24	//Lade AR1 mit dem Pointer in Merkerdoppelwort MD24.

Beispiel: Pointerkonstante

AWL	Erläuterung
LAR1 P#M100.0	//Lade AR1 mit einer 32-Bit Pointerkonstante.

9.6 LAR1 AR2 Lade Adreßregister 1 mit Inhalt von Adressregister 2

Format

LAR1 AR2

Beschreibung

LAR1 AR2 (Operation LAR1 mit dem Operand AR2) lädt das Adreßregister AR1 mit dem Inhalt von Adreßregister AR2. AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

9.7 LAR2 Lade Adreßregister 2 mit Inhalt von AKKU 1

Format

LAR2

Beschreibung

LAR2 lädt das Adreßregister AR2 mit dem Inhalt von AKKU 1 (32 Bit-Pointer).

AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

9.8 LAR2 <D> Lade Adreßregister 2 mit Ganzzahl (32 Bit)

Format

LAR2 <D>

Operand	Datentyp	Speicherbereich	Quelladresse
<D>	DWORD Pointerkonstante	D, M, L	0...65532

Beschreibung

LAR2 <D> lädt das Adreßregister AR 2 mit dem Inhalt des adressierten Doppelworts <D> oder einer Pointerkonstante. AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel: Direkte Adressen

AWL	Erläuterung
LAR2 DBD 20	//Lade AR 2 mit Pointer in Datendoppelwort DBD20.
LAR2 DID 30	//Lade AR 2 mit dem Pointer in Instanzdoppelwort DID30.
LAR2 LD 180	//Lade AR 2 mit dem Pointer in Lokaldaten-Doppelwort LD180.
LAR2 MD 24	//Lade AR 2 mit dem Inhalt des direkt adressierten Merkerdoppelwort MD24.

Beispiel: Pointerkonstante

AWL	Erläuterung
LAR2 P#M100.0	//Lade AR 2 mit einer 32-Bit Pointerkonstante.

9.9 T Transferiere

Format

T <Operand>

Operand	Datentyp	Speicherbereich	Quelladresse
<Operand>	BYTE	E, A, PA, M, L, D	0...65535
	WORD		0...65534
	DWORD		0...65532

Beschreibung

T <Operand> transferiert (kopiert) bei eingeschaltetem Master Control Relay (MCR = 1) den Inhalt von AKKU 1 in die Zieladresse. Bei MCR = 0 wird der Wert "0" an die Zieladresse geschrieben. Die Anzahl der Bytes, die aus dem AKKU 1 kopiert werden, hängt von der Größe ab, die in der Zieladresse angegeben ist. AKKU 1 speichert die Daten auch nach dem Transfervorgang. Das Transferieren in den direkten Peripheriebereich (Speicherbereich PA) transferiert auch den Inhalt von AKKU 1 oder "0" (bei MCR = 0) an die entsprechende Adresse im Prozeßabbild der Ausgänge (Speicherbereich A). Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
T AB10	//Transferiere den Inhalt von AKKU1-L-L in das Ausgangsbyte AB10.
T MW14	//Transferiere den Inhalt von AKKU1-L in das Merkerwort MW14.
T DBD2	//Transferiere den Inhalt von AKKU 1 in das Datendoppelwort DBD2.

9.10 T STW Transferiere AKKU 1 in Statuswort

Format

T STW

Beschreibung

T STW (Operation T mit dem Operand STW) transferiert Bit 0 bis Bit 8 von AKKU 1 in das Statuswort.

Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	x	x	x	x	x	x	x	x	x

Beispiel

AWL	Erläuterung
T STW	//Transferiere Bit 0 bis Bit 8 von AKKU 1 in das Statuswort.

Die Bits in AKKU 1 enthalten folgende Statusbits:

Bit	31-9	8	7	6	5	4	3	2	1	0
Inhalt:	*)	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER

*) Bits werden nicht transferiert.

9.11 TAR Tausche Adreßregister 1 mit 2

Format

TAR

Beschreibung

TAR (Tausche Adreßregister) tauscht die Inhalte der Adreßregister AR 1 und AR 2. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Der Inhalt von Adreßregister AR 1 wird verschoben in Adreßregister AR 2 und der Inhalt von Adreßregister AR 2 wird verschoben in Adreßregister AR 1.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

9.12 TAR1 Transferiere Adreßregister 1 in AKKU 1

Format

TAR1

Beschreibung

TAR1 transferiert den Inhalt von AR1 nach AKKU 1 (32 Bit-Pointer). Der Inhalt von AKKU 1 wurde zuvor in AKKU 2 gespeichert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

9.13 TAR1 <D> Transferiere Adreßregister 1 nach Zieladresse (32-Bit-Pointer)

Format

TAR1 <D>

Operand	Datentyp	Speicherbereich	Quelladresse
<D>	DWORD	D, M, L	0...65532

Beschreibung

TAR1 <D> transferiert den Inhalt von Adreßregister AR 1 in das adressierte Doppelwort <D>. Als Zielbereiche sind Merkerdoppelwörter (MD), Lokaldaten-Doppelwörter (LD), Datendoppelwörter (DBD) und Instanzdoppelwörter (DID) möglich.

AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiele

AWL	Erläuterung
TAR1 DBD20	//Transferiere den Inhalt von AR 1 in das Datendoppelwort DBD20.
TAR1 DID30	//Transferiere den Inhalt von AR 1 in das Instanzdoppelwort DID30.
TAR1 LD18	//Transferiere den Inhalt von AR 1 in das Lokaldaten-Doppelwort LD18.
TAR1 MD24	//Transferiere den Inhalt von AR 1 in das Merkerdoppelwort MD24.

9.14 TAR1 AR2 Transferiere Adreßregister 1 in Adreßregister 2

Format

TAR1 AR2

Beschreibung

TAR1 AR2 (Operation TAR1 mit dem Operand AR2) transferiert den Inhalt von Adreßregister AR 1 in Adreßregister AR 2.

AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

9.15 TAR2 Transferiere Adreßregister 2 in AKKU 1

Format

TAR2

Beschreibung

TAR2 transferiert den Inhalt von Adreßregister AR 2 in AKKU 1 (32 Bit-Pointer). Der Inhalt von AKKU 1 wurde zuvor in AKKU 2 gespeichert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

9.16 TAR2 <D> Transferiere Adreßregister 2 nach Zieladresse (32-Bit-Pointer)

Format

TAR2 <D>

Operand	Datentyp	Speicherbereich	Quelladresse
<D>	DWORD	D, M, L	0...65532

Beschreibung

TAR2 <D> transferiert den Inhalt von Adreßregister AR 2 in das adressierte Doppelwort <D>. Als Zielbereiche sind Merkerdoppelwörter (MD), Lokaldaten-Doppelwörter (LD), Datendoppelwörter (DBD) und Instanzdoppelwörter (DID) möglich.

AKKU 1 und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiele

AWL	Erläuterung
TAR2 DBD20	//Transferiere den Inhalt von AR 2 in das Datendoppelwort DBD20.
TAR2 DID30	//Transferiere den Inhalt von AR 2 in das Instanzdoppelwort DID30.
TAR2 LD18	//Transferiere den Inhalt von AR 2 in das Lokaldaten-Doppelwort LD18.
TAR2 MD24	//Transferiere den Inhalt von AR 2 in das Merkerdoppelwort MD24.

10 Programmsteuerung

10.1 Programmsteuerungsoperationen Übersicht

Beschreibung

Folgende Operationen stehen Ihnen zur Programmsteuerung zur Verfügung:

- BE Bausteinende
 - BEB Bausteinende bedingt
 - BEA Bausteinende absolut
 - CALL Bausteinaufruf
 - CC Bedingter Bausteinaufruf
 - UC Unbedingter Bausteinaufruf
-
- FB aufrufen
 - FC aufrufen
 - SFB aufrufen
 - SFC aufrufen
 - Multiinstanz aufrufen
 - Baustein aus einer Bibliothek aufrufen
-
- Das Master Control Relay
 - Wichtige Hinweise zur Benutzung der MCR-Funktionalität
 - MCR(Sichere VKE im MCR-Stack, Beginn MCR-Bereich
 -)MCR Beende MCR-Bereich
 - MCRA Aktiviere MCR-Bereich
 - MCRD Deaktiviere MCR-Bereich

10.2 BE Bausteinende

Format

BE

Beschreibung

BE (Bausteinende) unterbricht den Programmablauf im aktuellen Baustein und springt zu dem Baustein, der den aktuellen Baustein aufgerufen hat. Der Programmablauf wird mit der ersten Anweisung nach dem Bausteinaufruf fortgesetzt. Der aktuelle Lokaldatenbereich wird freigegeben und der vorherige Lokaldatenbereich wird zum aktuellen Lokaldatenbereich. Die Datenbausteine, die zum Zeitpunkt des Aufrufens des Bausteins geöffnet waren, werden erneut geöffnet. Zusätzlich wird die MCR-Abhängigkeit des aufrufenden Bausteins wiederhergestellt, und das VKE wird vom aktuellen Baustein in den aufrufenden Baustein übertragen. Die Operation BE ist nicht von Bedingungen abhängig. Wird die Operation BE übersprungen, wird der aktuelle Programmablauf nicht beendet, sondern am Sprungziel innerhalb des Bausteins fortgesetzt.

Die Operation BE ist mit der S5-Software nicht identisch. Bei S7-Hardware hat die Operation BE die gleiche Funktionalität wie die S5-Operation BEA.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel

AWL	Erläuterung
U E 1.0	
SPB NEXT	//Springe zur Sprungmarke NEXT, wenn das VKE = 1 (E 1.0 = 1) ist.
L EW4	//Setze hier fort, wenn der Sprung nicht ausgeführt wird.
T EW10	
U E 6.0	
U E 6.1	
S M 12.0	
BE	//Bausteinende.
NEXT: NOP 0	//Setze hier fort, wenn der Sprung ausgeführt wird.

10.3 BEB Bausteinende bedingt

Format

BEB

Beschreibung

Wenn VKE = 1, dann unterbricht **BEB** (Bausteinende bedingt) den Programmablauf im aktuellen Baustein und springt zu dem Baustein, der den aktuellen Baustein aufgerufen hat. Der Programmablauf wird mit der ersten Anweisung nach dem Bausteinanruf fortgesetzt. Der aktuelle Lokaldatenbereich wird freigegeben und der vorherige Lokaldatenbereich wird zum aktuellen Lokaldatenbereich. Die Datenbausteine, die zum Zeitpunkt des Aufrufens des Bausteins geöffnet waren, werden erneut geöffnet. Die MCR-Abhängigkeit des aufrufenden Bausteins wird wiederhergestellt.

Das VKE (= 1) wird von dem Baustein, der beendet wurde, in den Baustein, der aufgerufen hat, übernommen. Wenn das VKE = 0 ist, dann wird die Operation BEB nicht ausgeführt. Das VKE wird auf "1" gesetzt, und der Programmablauf wird mit der folgenden Anweisung fortgesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	x	0	1	1	0

Beispiel

AWL	Erläuterung
U E 1.0	//Aktualisiere das VKE.
BEB	//Beende den Baustein, wenn das VKE = 1 ist.
L EW4	//Setze hier fort, wenn die Anweisung BEB nicht ausgeführt wird (VKE = 0).
T MW10	

10.4 BEA Bausteinende absolut

Format

BEA

Beschreibung

BEA (Bausteinende absolut) unterbricht den Programmablauf im aktuellen Baustein und springt zu dem Baustein, der den aktuellen Baustein aufgerufen hat. Der Programmablauf wird mit der ersten Anweisung nach dem Bausteinaufruf fortgesetzt. Der aktuelle Lokaldatenbereich wird freigegeben und der vorherige Lokaldatenbereich wird zum aktuellen Lokaldatenbereich. Die Datenbausteine, die zum Zeitpunkt des Aufrufens des Bausteins geöffnet waren, werden erneut geöffnet. Zusätzlich wird die MCR-Abhängigkeit des aufrufenden Bausteins wiederhergestellt, und das VKE wird vom aktuellen Baustein in den aufrufenden Baustein übertragen. Die Operation BEA ist von keinen Bedingungen abhängig. Wird die Operation BEA übersprungen, wird der aktuelle Programmablauf nicht beendet, sondern am Sprungziel innerhalb des Bausteins fortgesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel

AWL	Erläuterung
U E 1.0	
SPB NEXT	//Springe zur Sprungmarke NEXT, wenn das VKE = 1 (E 1.0 = 1) ist.
L EW4	//Setze hier fort, wenn der Sprung nicht ausgeführt wird.
T EW10	
U E 6.0	
U E 6.1	
S M 12.0	
BEA	//Bausteinende absolut.
NEXT: NOP 0	//Setze hier fort, wenn der Sprung ausgeführt wird.

10.5 CALL Bausteinanruf

Format

CALL <Kennung des Codebausteins>

Beschreibung

CALL <Kennung des Codebausteins> dient zum Aufruf von Funktionen (FCs) und Funktionsbausteinen (FBs) beziehungsweise zum Aufruf der von Siemens gelieferten Standardfunktionen (SFCs) und Standardfunktionsbausteinen (SFBs). Die Operation CALL ruft die FC und SFC oder den FB und SFB auf, die oder den Sie als Operanden eingeben, unabhängig vom VKE oder einer anderen Bedingung. Wenn Sie einen FB oder SFB mit CALL aufrufen, müssen Sie ihn mit einem Instanz-Datenbaustein versehen. Nach der Bearbeitung des aufgerufenen Bausteins wird das Programm des aufrufenden Bausteins weiterbearbeitet. Die Kennung des Codebausteins kann absolut oder symbolisch angegeben werden. Register-Inhalte werden nach einem SFB-/SFC-Aufruf wieder restauriert.

Beispiel: CALL FB1, DB1 bzw. CALL FILLVAT1, RECIPE1

Codebaustein	Bausteinart	Syntax für den Aufruf (Absolute Adresse)
FC	Funktion	CALL FCn
SFC	Systemfunktion	CALL SFCn
FB	Funktionsbaustein	CALL FBn1,DBn2
SFB	Systemfunktionsbaustein	CALL SFBn1,DBn2

Hinweis

Wenn Sie mit dem AWL-Editor arbeiten, müssen sich die Angaben (n, n1 bzw. n2) aus der Tabelle oben auf bereits vorhandene gültige Bausteine beziehen. Die symbolischen Namen müssen Sie ebenfalls vorher definieren.

Übertragen von Parametern (arbeiten Sie hierzu im inkrementellen Bearbeitungsmodus)

Der aufrufende Baustein kann mit dem aufgerufenen Baustein über die Variablenliste Parameter austauschen. Die Variablenliste wird in Ihrem AWL-Programm automatisch ergänzt, wenn Sie eine gültige Anweisung CALL eingeben.

Wenn Sie einen FB bzw. einen SFB oder eine FC bzw. eine SFC aufrufen und die Variablendeklarationstabelle des aufgerufenen Bausteins über Deklarationen vom Typ IN, OUT und IN_OUT verfügt, werden diese Variablen im Programm des aufrufenden Bausteins als Liste der Formalparameter ergänzt.

Beim Aufruf der FCs und SFCs müssen Sie den Formalparametern Aktualparameter des aufrufenden Codebausteins zuordnen.

Beim Aufruf der FBs und SFBs müssen Sie nur die Aktualparameter angeben, die sich gegenüber dem letzten Aufruf ändern sollen, da die Aktualparameter nach der Bearbeitung des FB im Instanz-DB gespeichert sind. Ist der Aktualparameter ein DB, muß immer die vollständige, absolute Adresse angegeben werden, z.B. DB1, DBW2.

Die Parameter IN können als Konstanten oder als absolute bzw. symbolische Adressen angegeben werden. Die Parameter OUT und IN_OUT müssen als absolute bzw. symbolische Adressen angegeben werden. Achten Sie darauf, daß alle Adressen und Konstanten mit den Datentypen, die übertragen werden, kompatibel sind.

Die Operation CALL speichert die Rücksprungadresse (Selektor und relative Adresse), die Selektoren der beiden geöffneten Datenbausteine und das MA-Bit im B-Stack. Zusätzlich deaktiviert die Operation die MCR-Abhängigkeit und erstellt den Lokaldatenbereich des Bausteins, der aufgerufen werden soll.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel 1: Zuordnung von Parametern zu dem Aufruf der Funktion FC6

```
CALL    FC6
        Formalparameter      Aktualparameter
NO OF TOOL      := MW100
TIME OUT        := MW110
FOUND           := A 0.1
ERROR          := A 100.0
```

Beispiel 2: Aufruf einer SFC ohne Parameter

```
CALL      SFC43           //Rufe SFC43 auf, um die Zeitüberwachung neu
                        zu starten (ohne Parameter).
```

Beispiel 3: Aufruf des FB99 mit Instanz-Datenbaustein DB1

```
CALL      FB99,DB16
Formalparameter      Aktualparameter
MAX_RPM              := #RPM1_MAX
MIN_RPM              := #RPM1
MAX_POWER            := #POWER1
MAX_TEMP             := #TEMP1
```

Beispiel 4: Aufruf des FB99 mit Instanz-Datenbaustein DB2

```
CALL      FB99,DB2
Formalparameter      Aktualparameter
MAX_RPM              := #RPM2_MAX
MIN_RPM              := #RPM2
MAX_POWER            := #POWER2
MAX_TEMP             := #TEMP2
```

Hinweis

Jeder Aufruf eines FBs oder eines SFBs muß über einen Instanz-Datenbaustein verfügen. In dem obigen Beispiel müssen die Bausteine DB1 und DB2 vor dem Aufruf vorhanden sein.

10.6 FB aufrufen

Format

CALL FB n1, DB n1

Beschreibung

Die Operation dient zum Aufruf von selbsterstellten Funktionsbausteinen (FBs). Die Operation CALL ruft den FB auf, den Sie als Operanden eingeben haben, unabhängig vom VKE oder einer anderen Bedingung. Wenn Sie einen FB mit CALL aufrufen, müssen Sie ihn mit einem Instanz-Datenbaustein versehen. Nach der Bearbeitung des aufgerufenen Bausteins wird das Programm des aufrufenden Bausteins weiterbearbeitet. Die Kennung des Codebausteins kann absolut oder symbolisch angegeben werden.

Übertragen von Parametern (arbeiten Sie hierzu im inkrementellen Bearbeitungsmodus)

Der aufrufende Baustein kann mit dem aufgerufenen Baustein über die Variablenliste Parameter austauschen. Die Variablenliste wird in Ihrem AWL-Programm automatisch ergänzt, wenn Sie eine gültige Anweisung CALL eingeben.

Wenn Sie einen FB aufrufen und die Variablendeklarationstabelle des aufgerufenen Bausteins über Deklarationen vom Typ IN, OUT und IN_OUT verfügt, werden diese Variablen im Programm des aufrufenden Bausteins als Liste der Formalparameter ergänzt.

Beim Aufruf der FBs müssen Sie nur die Aktualparameter angeben, die sich gegenüber dem letzten Aufruf ändern sollen, da die Aktualparameter nach der Bearbeitung des FB im Instanz-DB gespeichert sind. Ist der Aktualparameter ein DB, muß immer die vollständige, absolute Adresse angegeben werden, z.B. DB1, DBW2.

Die Parameter IN können als Konstanten oder als absolute bzw. symbolische Adressen angegeben werden. Die Parameter OUT und IN_OUT müssen als absolute bzw. symbolische Adressen angegeben werden. Achten Sie darauf, daß alle Adressen und Konstanten mit den Datentypen, die übertragen werden, kompatibel sind.

Die Operation CALL speichert die Rücksprungadresse (Selektor und relative Adresse), die Selektoren der beiden geöffneten Datenbausteine und das MA-Bit im B-Stack. Zusätzlich deaktiviert die Operation die MCR-Abhängigkeit und erstellt den Lokaldatenbereich des Bausteins, der aufgerufen werden soll.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel 1: Aufruf des FB99 mit Instanz-Datenbaustein DB1

```

CALL    FB99, DB1
        Formalparameter      Aktualparameter
MAX_RPM      := #RPM1_MAX
MIN_RPM      := #RPM1
MAX_POWER    := #POWER1
MAX_TEMP     := #TEMP1
    
```

Beispiel 2: Aufruf des FB99 mit Instanz-Datenbaustein DB2

```

CALL    FB99, DB2
        Formalparameter      Aktualparameter
MAX_RPM      := #RPM2_MAX
MIN_RPM      := #RPM2
MAX_POWER    := #POWER2
MAX_TEMP     := #TEMP2
    
```

Hinweis

Jeder Aufruf eines FBs muß über einen Instanz-Datenbaustein verfügen. In dem obigen Beispiel müssen die Bausteine DB1 und DB2 vor dem Aufruf vorhanden sein.

10.7 FC aufrufen

Format

CALL FC n

Hinweis

Wenn Sie mit dem AWL-Editor arbeiten, muß sich die Angabe (n) auf bereits vorhandene gültige Bausteine beziehen. Die symbolischen Namen müssen Sie ebenfalls vorher definieren.

Beschreibung

Die Operation dient zum Aufruf von Funktionen (FCs). Die Operation CALL ruft die FC auf, die Sie als Operanden eingeben, unabhängig vom VKE oder einer anderen Bedingung. Nach der Bearbeitung des aufgerufenen Bausteins wird das Programm des aufrufenden Bausteins weiterbearbeitet. Die Kennung des Codebausteins kann absolut oder symbolisch angegeben werden.

Übertragen von Parametern (arbeiten Sie hierzu im inkrementellen Bearbeitungsmodus)

Der aufrufende Baustein kann mit dem aufgerufenen Baustein über die Variablenliste Parameter austauschen. Die Variablenliste wird in Ihrem AWL-Programm automatisch ergänzt, wenn Sie eine gültige Anweisung CALL eingeben.

Wenn Sie eine FC aufrufen und die Variablendeklarationstabelle des aufgerufenen Bausteins über Deklarationen vom Typ IN, OUT und IN_OUT verfügt, werden diese Variablen im Programm des aufrufenden Bausteins als Liste der Formalparameter ergänzt.

Beim Aufruf der FCs müssen Sie den Formalparametern Aktualparameter des aufrufenden Codebausteins zuordnen.

Die Parameter IN können als Konstanten oder als absolute bzw. symbolische Adressen angegeben werden. Die Parameter OUT und IN_OUT müssen als absolute bzw. symbolische Adressen angegeben werden. Achten Sie darauf, daß alle Adressen und Konstanten mit den Datentypen, die übertragen werden, kompatibel sind.

Die Operation CALL speichert die Rücksprungadresse (Selektor und relative Adresse), die Selektoren der beiden geöffneten Datenbausteine und das MA-Bit im B-Stack. Zusätzlich deaktiviert die Operation die MCR-Abhängigkeit und erstellt den Lokaldatenbereich des Bausteins, der aufgerufen werden soll.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel: Zuordnung von Parametern zu dem Aufruf der Funktion FC6

```

CALL    FC6
        Formalparameter      Aktualparameter
NO OF TOOL      := MW100
TIME OUT        := MW110
FOUND           := A 0.1
ERROR          := A 100.0
    
```

10.8 SFB aufrufen

Format

CALL SFB n1, DB n2

Beschreibung

Die Operation dient zum Aufruf der von Siemens gelieferten Standardfunktionsbausteine (SFBs). Die Operation CALL ruft die SFB auf, die Sie als Operanden eingeben, unabhängig vom VKE oder einer anderen Bedingung. Wenn Sie einen SFB mit CALL aufrufen, müssen Sie ihn mit einem Instanz-Datenbaustein versehen. Nach der Bearbeitung des aufgerufenen Bausteins wird das Programm des aufrufenden Bausteins weiterbearbeitet. Die Kennung des Codebausteins kann absolut oder symbolisch angegeben werden.

Übertragen von Parametern (arbeiten Sie hierzu im inkrementellen Bearbeitungsmodus)

Der aufrufende Baustein kann mit dem aufgerufenen Baustein über die Variablenliste Parameter austauschen. Die Variablenliste wird in Ihrem AWL-Programm automatisch ergänzt, wenn Sie eine gültige Anweisung CALL eingeben.

Wenn Sie einen SFB aufrufen und die Variablendeklarationstabelle des aufgerufenen Bausteins über Deklarationen vom Typ IN, OUT und IN_OUT verfügt, werden diese Variablen im Programm des aufrufenden Bausteins als Liste der Formalparameter ergänzt.

Beim Aufruf der SFBs müssen Sie nur die Aktualparameter angeben, die sich gegenüber dem letzten Aufruf ändern sollen, da die Aktualparameter nach der Bearbeitung des SFB im Instanz-DB gespeichert sind. Ist der Aktualparameter ein DB, muß immer die vollständige, absolute Adresse angegeben werden, z.B. DB1, DBW2.

Die Parameter IN können als Konstanten oder als absolute bzw. symbolische Adressen angegeben werden. Die Parameter OUT und IN_OUT müssen als absolute bzw. symbolische Adressen angegeben werden. Achten Sie darauf, daß alle Adressen und Konstanten mit den Datentypen, die übertragen werden, kompatibel sind.

Die Operation CALL speichert die Rücksprungadresse (Selektor und relative Adresse), die Selektoren der beiden geöffneten Datenbausteine und das MA-Bit im B-Stack. Zusätzlich deaktiviert die Operation die MCR-Abhängigkeit und erstellt den Lokaldatenbereich des Bausteins, der aufgerufen werden soll.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel

CALL	SFB4 ,DB4	
	Formalparameter	Aktualparameter
	IN:	E0.1
	PT:	T#20s
	Q:	M0.0
	ET:	MW10

Hinweis

Jeder Aufruf eines SFBs muß über einen Instanz-Datenbaustein verfügen. In dem obigen Beispiel müssen die Bausteine SFB4 und DB4 vor dem Aufruf vorhanden sein.

10.9 SFC aufrufen

Format

CALL SFC n

Hinweis

Wenn Sie mit dem AWL-Editor im inkrementellen Bearbeitungsmodus arbeiten, muß sich die Angabe (n) auf bereits vorhandene gültige Bausteine beziehen. Die symbolischen Namen müssen Sie ebenfalls vorher definieren.

Beschreibung

Die Operation dient zum Aufruf von Siemens gelieferten Standardfunktionen (SFCs). Die Operation CALL ruft die SFC auf, die Sie als Operanden eingeben, unabhängig vom VKE oder einer anderen Bedingung. Nach der Bearbeitung des aufgerufenen Bausteins wird das Programm des aufrufenden Bausteins weiterbearbeitet. Die Kennung des Codebausteins kann absolut oder symbolisch angegeben werden.

Übertragen von Parametern (arbeiten Sie hierzu im inkrementellen Bearbeitungsmodus)

Der aufrufende Baustein kann mit dem aufgerufenen Baustein über die Variablenliste Parameter austauschen. Die Variablenliste wird in Ihrem AWL-Programm automatisch ergänzt, wenn Sie eine gültige Anweisung CALL eingeben.

Wenn Sie eine SFC aufrufen und die Variablendeklarationstabelle des aufgerufenen Bausteins über Deklarationen vom Typ IN, OUT und IN_OUT verfügt, werden diese Variablen im Programm des aufrufenden Bausteins als Liste der Formalparameter ergänzt.

Beim Aufruf der SFCs müssen Sie den Formalparametern Aktualparameter des aufrufenden Codebausteins zuordnen.

Die Parameter IN können als Konstanten oder als absolute bzw. symbolische Adressen angegeben werden. Die Parameter OUT und IN_OUT müssen als absolute bzw. symbolische Adressen angegeben werden. Achten Sie darauf, daß alle Adressen und Konstanten mit den Datentypen, die übertragen werden, kompatibel sind.

Die Operation CALL speichert die Rücksprungadresse (Selektor und relative Adresse), die Selektoren der beiden geöffneten Datenbausteine und das MA-Bit im B-Stack. Zusätzlich deaktiviert die Operation die MCR-Abhängigkeit und erstellt den Lokaldatenbereich des Bausteins, der aufgerufen werden soll.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel: Aufruf einer SFC ohne Parameter

AWL	Erläuterung
CALL SFC43	//Rufe SFC43 auf, um die Zeitüberwachung neu zu starten (ohne Parameter).

10.10 Multiinstanz aufrufen

Format

CALL # Variablenname

Beschreibung

Eine Multiinstanz entsteht durch die Deklaration einer statischen Variable vom Datentyp eines Funktionsbausteins. Nur bereits deklarierte Multiinstanzen werden im Programmelementekatalog aufgeführt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	X	X	X

10.11 Baustein aus einer Bibliothek aufrufen

Die im SIMATIC Manager bekannten Bibliotheken werden Ihnen im Programmelemente-Katalog zur Auswahl angeboten.

Aus diesen Bibliotheken können Sie Bausteine auswählen,

- die im Betriebssystem Ihrer CPU integriert sind (Bibliothek "Standard Library"),
- die Sie selbst in Bibliotheken abgelegt haben, weil Sie sie mehrfach verwenden wollen.

10.12 CC Bedingter Bausteinanruf

Format

CC <Kennung des Codebausteins>

Beschreibung

CC <Kennung des Codebausteins> (bedingter Bausteinanruf) ruft bei VKE = 1 einen Codebaustein vom Typ FC oder FB ohne Parameter auf. Die Operation CC gleicht der Operation CALL, mit dem Unterschied, daß keine Parameter übergeben werden können. Die Operation speichert die Rücksprungadresse (Selektor und relative Adresse), die Selektoren der beiden aktuellen Datenbausteine sowie das MA-Bit im B-Stack, deaktiviert die MCR-Abhängigkeit, erstellt den Lokaldatenbereich des Bausteins, der aufgerufen werden soll, und beginnt, den aufgerufenen Code auszuführen. Die Kennung des Codebausteins kann absolut oder symbolisch angegeben werden.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	1	0

Beispiel

AWL	Erläuterung
U E 2.0	//Frage den Signalzustand am Eingang E 2.0 ab.
CC FC6	//Rufe die Funktion FC6 auf, wenn E 2.0 = 1 ist.
U M3.0	//Wird nach Rückkehr von der aufgerufenen Funktion ausgeführt (bei E 2.0 = 1) bzw. unmittelbar nach der Anweisung U E 2.0, wenn E 2.0 = 0 ist.

Hinweis

Wird mit der Operation **CALL** ein Funktionsbaustein (FB) oder ein Systemfunktionsbaustein (SFB) aufgerufen, muß ein Instanz-Datenbaustein (DB-Nr.) in der Anweisung angegeben werden. Bei einem Aufruf mit der Operation **CC** können Sie in dem Operanden der Anweisung keinen Datenbaustein zuordnen.

Je nach dem Netzwerk, mit dem Sie arbeiten, erzeugt "KOP/AWL: Bausteine programmieren" bei der Übersetzung der Programmiersprache Kontaktplan in die Programmiersprache Anweisungsliste teilweise die Operation **UC** und teilweise die Operation **CC**. Verwenden Sie im allgemeinen die Operation **CALL**, damit in den von Ihnen erstellten Programmen keine Fehler auftreten.

10.13 UC Unbedingter Bausteinanruf

Format

UC <Kennung des Codebausteins>

Beschreibung

UC <Kennung des Codebausteins> (unbedingter Bausteinanruf) ruft einen Codebaustein vom Typ FC, FB, SFC oder SFB auf. Die Operation UC gleicht der Operation CALL, mit dem Unterschied, daß keine Parameter übergeben werden können. Die Operation speichert die Rücksprungadresse (Selektor und relative Adresse), die Selektoren der beiden aktuellen Datenbausteine sowie das MA-Bit im B-Stack, deaktiviert die MCR-Abhängigkeit, erstellt den Lokaldatenbereich des Bausteins, der aufgerufen werden soll, und beginnt, den aufgerufenen Code auszuführen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	0	0	1	-	0

Beispiel 1

AWL	Erläuterung
UC FC6	//Rufe die Funktion FC6 auf (ohne Parameter).

Beispiel 2

AWL	Erläuterung
UC SFC43	//Rufe die Systemfunktion SFC43 auf (ohne Parameter).

Hinweis

Wird mit der Operation **CALL** ein FB oder ein SFB aufgerufen, muß ein Instanz-Datenbaustein (DB-Nr.) in der Anweisung angegeben werden. Bei einem Aufruf mit der Operation **UC** können Sie in dem Operanden der Anweisung keinen Datenbaustein zuordnen.

Je nach dem Netzwerk, mit dem Sie arbeiten, erzeugt "KOP/AWL: Bausteine programmieren" bei der Übersetzung der Programmiersprache Kontaktplan in die Programmiersprache Anweisungsliste teilweise die Operation UC und teilweise die Operation **CC**. Verwenden Sie im allgemeinen die Operation **CALL**, damit in den von Ihnen erstellten Programmen keine Fehler auftreten.

10.14 Das Master Control Relay

Wichtige Hinweise zur Benutzung der MCR-Funktionalität



Warnung

Um das Risiko möglicher Personen- oder Sachschäden auszuschließen, verwenden Sie das MCR niemals als Ersatz für ein festverdrahtetes, mechanisches Master Control Relay, das als NOT-AUS-Einrichtung dient.

Definition des Master Control Relay (MCR)

Das Master Control Relay wird in Relais-Kontaktplänen für das Aktivieren und Deaktivieren des Signalfusses verwendet. Operationen, die von den folgenden Bitverknüpfungs- und Transferoperationen ausgelöst werden, sind vom MCR abhängig:

- = <Bit>
- S <Bit>
- R <Bit>
- T <Byte>, T <Wort>, T <Doppelwort>

Die Operation T, die mit Byte, Wort oder Doppelwort verwendet wird, schreibt eine "0" in den Speicher, wenn das MCR "0" ist. Die Operationen S und R ändern den bereits bestehenden Wert nicht. Die Operation = schreibt eine "0" in das adressierte Bit.

Reaktionen der Operationen auf den Signalzustand des MCR

Signalzustand des MCR	= <Bit>	S <Bit>, R <Bit>	T <Byte>, T <Wort>, T <Doppelwort>
0 ("AUS")	Schreibt "0". (Imitiert ein Relais, das bei Spannungsausfall in seinen Ruhezustand geht.)	Schreibt nicht. (Imitiert ein Relais, das bei Spannungsausfall in seinem aktuellen Zustand bleibt.)	Schreibt "0". (Imitiert eine Komponente, die bei Spannungsausfall den Wert "0" ausgibt.)
1 ("EIN")	Normale Bearbeitung	Normale Bearbeitung	Normale Bearbeitung

MCR(- Beginn MCR-Bereich,)MCR - Ende MCR-Bereich

Das MCR wird von einem Stack gesteuert, der ein Bit breit und acht Bits tief ist. Das MCR ist solange eingeschaltet, wie alle acht Einträge gleich "1" sind. Die Operation MCR(kopiert das VKE-Bit in den MCR-Stack. Die Operation)MCR löscht den letzten Eintrag aus dem Stack und setzt die freigewordene Stelle auf "1". Die Operationen MCR(und)MCR müssen immer paarweise verwendet werden. Folgen mehr als acht MCR(-Operationen aufeinander oder wird bei leerem Stack versucht, eine Operation)MCR auszuführen, tritt die Fehlermeldung MCRF auf.

MCRA - Aktiviere MCR-Bereich, MCRD - Deaktiviere MCR-Bereich

Die Operationen MCRA und MCRD müssen immer paarweise verwendet werden. Anweisungen, die zwischen MCRA und MCRD programmiert sind, sind vom Status des MCR-Bits abhängig. Die Anweisungen, die sich außerhalb einer MCRA-MCRD-Folge befinden, sind nicht vom Status des MCR-Bits abhängig.

Bei Aufruf von FCs und FBs müssen Sie die MCR-Abhängigkeit in diesen Bausteinen programmieren. Verwenden Sie hierzu die Operation MCRA im aufgerufenen Baustein.

10.15 Wichtige Hinweise zur Benutzung der MCR-Funktionalität



Vorsicht bei Bausteinen, in denen mit MCRA das Master Control Relay aktiviert wurde:

- Wenn das MCR abgeschaltet ist, wird in Programmabschnitten zwischen **MCR(** und **)MCR** durch alle Zuweisungen (T, =) der Wert 0 geschrieben!
 - Das MCR ist genau dann abgeschaltet, wenn vor einem **MCR(** -Befehl das VKE = 0 war.
-



Gefahr: STOP der AS oder undefiniertes Laufzeitverhalten !

Der Compiler greift für Adreßberechnungen auch schreibend auf Lokaldaten hinter den in VAR_TEMP definierten temporären Variablen zu. Daher setzen folgende Befehlssequenzen die AS in STOP oder führen zu undefiniertem Laufzeitverhalten:

Formalparameterzugriffe

- Zugriffe auf Komponenten komplexer FC-Parameter vom Typ STRUCT, UDT, ARRAY, STRING
- Zugriffe auf Komponenten komplexer FB-Parameter vom Typ STRUCT, UDT, ARRAY, STRING aus dem Bereich IN_OUT in einem multiinstanzfähigen Baustein (Bausteinversion 2).
- Zugriffe auf Parameter eines multiinstanzfähigen FB (Bausteinversion 2), wenn ihre Adresse größer als 8180.0 ist.
- Zugriff im multiinstanzfähigen FB (Bausteinversion 2) auf einen Parameter vom Typ BLOCK_DB schlägt den DB 0 auf. Nachfolgende Datenzugriffe bringen die CPU in STOP. Bei TIMER, COUNTER, BLOCK_FC, BLOCK_FB wird auch immer T 0, Z 0, FC 0 bzw. FB 0 verwendet.

Parameterübergabe

- Calls, bei denen Parameter übergeben werden.

KOP/FUP

- T-Abzweige und Konnektoren in KOP oder FUP starten mit VKE = 0.

Abhilfe

Lösen Sie die genannten Befehle aus der MCR-Abhängigkeit:

1. Deaktivieren Sie das Master Control Relay mit dem MCRD-Befehl vor der betreffenden Anweisung bzw. vor dem betreffenden Netzwerk.
 2. Aktivieren Sie das Master Control Relay mit dem MCRA-Befehl nach der betreffenden Anweisung bzw. nach dem betreffenden Netzwerk.
-

10.16 MCR(Sichere VKE im MCR-Stack, Beginn MCR-Bereich

Wichtige Hinweise zur Benutzung der MCR-Funktionalität

Format

MCR(

Beschreibung

MCR((Öffne einen MCR-Bereich) speichert das VKE im MCR-Stack und öffnet einen MCR-Bereich. MCR-Bereich: Anweisungen zwischen der Operation **MCR(** und der dazugehörigen Operation **)MCR**. Die Operationen **MCR(** und **)MCR** müssen immer paarweise verwendet werden.

Wenn das VKE = 1 ist, dann ist das MCR "eingeschaltet". Die MCR-abhängigen Anweisungen innerhalb dieses MCR-Bereichs werden normal ausgeführt.

Wenn das VKE = 0 ist, dann ist das MCR "ausgeschaltet".

Die MCR-abhängigen Anweisungen innerhalb dieses MCR-Bereichs werden entsprechend der folgenden Tabelle ausgeführt.

Reaktionen der Operationen auf den Signalzustand des MCR

Signalzustand des MCR	= <Bit>	S <Bit>, R <Bit>	T <Byte>, T <Wort>, T <Doppelwort>
0 ("AUS")	Schreibt "0". (Imitiert ein Relais, das bei Spannungsausfall in seinen Ruhezustand geht.)	Schreibt nicht. (Imitiert ein Relais, das bei Spannungsausfall in seinem aktuellen Zustand bleibt.)	Schreibt "0". (Imitiert eine Komponente, die bei Spannungsausfall den Wert "0" ausgibt.)
1 ("EIN")	Normale Bearbeitung	Normale Bearbeitung	Normale Bearbeitung

Die Operationen **MCR(** und **)MCR** können geschachtelt werden. Die maximale Schachteltiefe liegt bei acht Operationen. Der Stack kann also maximal acht Einträge enthalten. Wird die Operation **MCR(** bei vollem Stack ausgeführt, ruft dies einen MCR-Stackfehler (MCRF) hervor.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

Beispiel

AWL	Erläuterung
MCRA	//Aktiviere MCR-Bereich.
U E 1.0	
MCR(//Speichere das VKE im MCR-Stack, öffne einen MCR-Bereich. Das MCR ist "EIN", wenn das VKE = 1 (E 1.0 = 1) ist. Das MCR ist "AUS", wenn das VKE = 0 (E 1.0 = 0) ist.
U E 4.0	
= A 8.0	//Wenn das MCR = "AUS" ist, wird A 8.0 auf "0" gesetzt, ohne E 4.0 zu berücksichtigen.
L MW20	
T AW10	//Wenn das MCR = "AUS" ist, wird der Wert "0" in AW10 transferiert.
)MCR	//Beende den MCR-Bereich.
MCRD	//Deaktiviere MCR-Bereich.
U E 1.1	
= A 8.1	//Diese Anweisungen liegen außerhalb des MCR-Bereichs und sind nicht vom MCR-Bit abhängig.

10.17)MCR Beende MCR-Bereich

Wichtige Hinweise zur Benutzung der MCR-Funktionalität

Format

)MCR

Beschreibung

)MCR (Beende einen MCR-Bereich) löscht einen Eintrag aus dem MCR-Stack und beendet einen MCR-Bereich. Der letzte Eintrag des MCR-Stacks wird frei und auf "1" gesetzt. Die Operationen MCR(und)MCR müssen immer paarweise verwendet werden. Wird die Operation)MCR bei leerem Stack ausgeführt, ruft dies einen MCR-Stackfehler (MCRF) hervor.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	1	-	0

Beispiel

AWL	Erläuterung
MCRA	Aktiviere den MCR-Bereich.
U E 1.0	
MCR(//Speichere das VKE im MCR-Stack, öffne einen MCR-Bereich. Das MCR ist "EIN", wenn das VKE = 1 (E 1.0 = 1) ist. Das MCR ist "AUS", wenn das VKE = 0 (E 1.0 = 0) ist.
U E 4.0	
= A 8.0	//Wenn das MCR = "AUS" ist, wird A 8.0 auf "0" gesetzt, ohne E 4.0 zu berücksichtigen.
L MW20	
T AW10	//Wenn das MCR = "AUS" ist, wird "0" in AW10 transferiert.
)MCR	//Beende den MCR-Bereich.
MCRD	//Deaktiviere den MCR-Bereich.
U E 1.1	
= A 8.1	//Diese Anweisungen liegen außerhalb des MCR-Bereichs und sind nicht vom MCR-Bit abhängig.

10.18 MCRA Aktiviere MCR-Bereich

Wichtige Hinweise zur Benutzung der MCR-Funktionalität

Format

MCRA

Beschreibung

MCRA (Aktivierung des Master Control Relay) schaltet die MCR-Abhängigkeit für die Anweisungen ein, die dieser Operation folgen. Die Operationen MCRA und MCRD (Deaktivierung des Master Control Relay) müssen immer paarweise verwendet werden. Die Anweisungen, die zwischen MCRA und MCRD programmiert sind, sind vom Signalzustand des MCR-Bits abhängig.

Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
MCRA	//Aktiviere den MCR-Bereich.
U E 1.0	
MCR(//Speichere das VKE im MCR-Stack, öffne einen MCR-Bereich. Das MCR ist "EIN", wenn das VKE = 1 (E 1.0 = 1) ist. Das MCR ist "AUS", wenn das VKE = 0 (E 1.0 = 0) ist.
U E 4.0	
= A 8.0	//Wenn das MCR = "AUS" ist, wird A 8.0 auf "0" gesetzt, ohne E 4.0 zu berücksichtigen.
L MW20	
T AW10	//Wenn das MCR = "AUS" ist, wird "0" nach AW10 transferiert.
)MCR	//Beende den MCR-Bereich.
MCRD	//Deaktiviere den MCR-Bereich.
U E 1.1	
= A 8.1	//Diese Anweisungen liegen außerhalb des MCR-Bereichs und sind nicht vom MCR-Bit abhängig.

10.19 MCRD Deaktiviere MCR-Bereich

Wichtige Hinweise zur Benutzung der MCR-Funktionalität

Format

MCRD

Beschreibung

MCRD (Deaktivierung des Master Control Relay) schaltet die MCR-Abhängigkeit für die Anweisungen aus, die dieser Operation folgen. Die Operationen MCRD und MCRA (Aktivierung des Master Control Relay) müssen immer paarweise verwendet werden. Die Anweisungen, die zwischen MCRA und MCRD programmiert sind, sind vom Signalzustand des MCR-Bits abhängig.

Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
MCRA	//Aktiviere den MCR-Bereich.
U E 1.0	
MCR(//Speichere das VKE im MCR-Stack, öffne einen MCR-Bereich. Das MCR ist "EIN", wenn das VKE = 1 (E 1.0 = 1) ist. Das MCR ist "AUS", wenn das VKE = 0 (E 1.0 = 0) ist.
U E 4.0	
= A 8.0	//Wenn das MCR = "AUS" ist, wird A 8.0 auf "0" gesetzt, ohne E 4.0 zu berücksichtigen.
L MW20	
T AW10	//Wenn das MCR = "AUS" ist, wird "0" nach AW10 transferiert.
)MCR	//Beende den MCR-Bereich.
MCRD	//Deaktiviere den MCR-Bereich.
U E 1.1	
= A 8.1	//Diese Anweisungen liegen außerhalb des MCR-Bereichs und sind nicht vom MCR-Bit abhängig.

11 Schieben/Rotieren

11.1 Schiebeoperationen

11.1.1 Schiebeoperationen Übersicht

Beschreibung

Mit den Schiebeoperationen können Sie den Inhalt des niederwertigen Worts von AKKU 1 oder den Inhalt des gesamten Akkumulators bitweise nach links oder rechts schieben (siehe auch CPU-Register). Ein Schieben um n Bits nach links multipliziert den Akkumulatorinhalt mit 2 hoch n ; ein Schieben um n Bits nach rechts dividiert den Akkumulatorinhalt durch 2 hoch n . Wenn Sie also beispielsweise das binäre Äquivalent des Dezimalwerts 3 um 3 Bits nach links schieben, so ergibt sich das binäre Äquivalent des Dezimalwerts 24. Schieben Sie das binäre Äquivalent des Dezimalwerts 16 um 2 Bits nach rechts, so ergibt sich das binäre Äquivalent des Dezimalwerts 4.

Die Zahl, die auf eine Schiebeoperation oder einen Wert im niederwertigen Byte des niederwertigen Worts von AKKU 2 folgt, gibt an, um wie viele Bits geschoben werden soll. Die Stellen, die durch die Schiebeoperation frei werden, werden entweder mit Nullen oder mit dem Signalzustand des Vorzeichenbits aufgefüllt ("0" steht für positiv, "1" steht für negativ). Das zuletzt geschobene Bit wird in das Bit A1 des Statusworts geladen. Die Bits A0 und OV werden auf "0" zurückgesetzt. Mit den Sprungoperationen können Sie das Bit A1 im Statuswort auswerten.

Die Schiebeoperationen sind absolut, d.h. ihre Ausführung hängt nicht von bestimmten Bedingungen ab. Sie beeinflussen das VKE nicht.

Folgende Schiebeoperationen stehen Ihnen zur Verfügung:

- SSI Schiebe Vorzeichen rechts Ganzzahl (16 Bit)
- SSD Schiebe Vorzeichen rechts Ganzzahl (32 Bit)
- SLW Schiebe links Wort (16 Bit)
- SRW Schiebe rechts Wort (16 Bit)
- SLD Schiebe links Doppelwort (32 Bit)
- SRD Schiebe rechts Doppelwort (32 Bit)

11.1.2 SSI Schiebe Vorzeichen rechts Ganzzahl (16 Bit)

Formate

SSI

SSI <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die geschoben werden soll; Bereich von 0 bis 15

Beschreibung

SSI (Schiebe Ganzzahl vorzeichenrichtig nach rechts) schiebt nur den Inhalt von AKKU1-L bitweise nach rechts. In die Bitstellen, die durch das Schieben frei werden, wird der Signalzustand des Vorzeichenbits (Bit 15) geschrieben. Das zuletzt geschobene Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die geschoben werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

SSI <Anzahl>: Die Schiebezahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte von 0 bis 15. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer als Null ist. Ist <Anzahl> gleich "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

SSI: Die Schiebezahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte von 0 bis 255. Eine Schiebezahl > 16 ruft immer das gleiche Ergebnis hervor: (AKKU 1 = 16#0000, A1 = 0 bzw. AKKU 1 = 16#FFFF, A1 = 1. Ist die Schiebezahl > 0, werden die Statusbits A0 und OV auf "0" zurückgesetzt. Ist die Schiebezahl "0", wird die Schiebeoperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
Bit	31 16	15 0
vor Ausführung von SSI 6	0101	1111	0110	0100	1001	1101	0011	1011
nach Ausführung von SSI 6	0101	1111	0110	0100	1111	1110	0111	0100

Beispiel 1

AWL	Erläuterung	
L	MW4	//Lade den Wert in AKKU 1.
SSI	6	//Schiebe die Bits in AKKU1 vorzeichenrichtig um 6 Stellen nach rechts.
T	MW8	//Transferiere das Ergebnis nach MW8.

Beispiel 2

AWL	Erläuterung	
L	+3	//Lade den Wert +3 in AKKU 1.
L	MW20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MW20 in AKKU 1.
SSI		//Die Schiebezahl ist der Wert von AKKU2-L-L. => Schiebe die Bits in AKKU1-L vorzeichenrichtig um 3 Stellen nach rechts, setze die freien Stellen auf den Signalzustand des Vorzeichenbits.
SPP	NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt geschobene Bit (A1) = 1 ist.

11.1.3 SSD Schiebe Vorzeichen rechts Ganzzahl (32 Bit)

Formate

SSD

SSD <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die geschoben werden soll; Bereich von 0 bis 32

Beschreibung

SSD (Schiebe Ganzzahl, 32 Bit, vorzeichenrichtig nach rechts) schiebt den gesamten Inhalt von AKKU 1 bitweise nach rechts. In die Bitstellen, die durch das Schieben frei werden, wird der Signalzustand des Vorzeichenbits geschrieben. Das zuletzt geschobene Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die geschoben werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

SSD <Anzahl>: Die Schiebezahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte von 0 bis 32. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer als Null ist. Ist <Anzahl> gleich "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

SSD: Die Schiebezahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte von 0 bis 255. Eine Schiebezahl > 32 ruft immer das gleiche Ergebnis hervor: AKKU 1 = 32#00000000, A1 = 0 bzw. AKKU 1 = 32#FFFFFFFF, A1 = 1. Ist die Schiebezahl > 0, werden die Statusbits A0 und OV auf "0" zurückgesetzt. Ist die Schiebezahl "0", wird die Schiebeoperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
	31 16	15 0
vor Ausführung von SSD 7	1000	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von SSD 7	1111	1111	0001	1110	1100	1000	1011	1010

Beispiel 1

AWL	Erläuterung	
L	MD4	//Lade den Wert in AKKU 1.
SSD	7	//Verschiebe die Bits in AKKU1 vorzeichenrichtig um 7 Stellen nach rechts.
T	MD8	//Transferiere das Ergebnis nach MD8.

Beispiel 2

AWL	Erläuterung	
L	3	//Lade den Wert +3 in AKKU 1.
L	MD20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD20 in AKKU 1.
SSD		//Die Schiebezahl ist der Wert von AKKU2-L-L. => Schiebe die Bits in AKKU 1 vorzeichenrichtig um 3 Stellen nach rechts, setze die freien Stellen auf den Signalzustand des Vorzeichenbits.
SPP	NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt geschobene Bit (A1) = 1 ist.

11.1.4 SLW Schiebe links Wort (16 Bit)

Formate

SLW

SLW <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die geschoben werden soll; Bereich von 0 bis 32

Beschreibung

SLW (Schiebe links Wort) schiebt nur den Inhalt von AKKU1-L bitweise nach links. In die Bitstellen, die durch das Schieben frei werden, werden Nullen geschrieben. Das zuletzt geschobene Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die geschoben werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

SLW <Anzahl>: Die Schiebezahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte von 0 bis 15. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer als Null ist. Ist <Anzahl> gleich "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

SLW: Die Schiebezahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte von 0 bis 255. Eine Schiebezahl > 16 ruft immer das gleiche Ergebnis hervor: AKKU1-L = 0, A1 = 0, A0 = 0, OV = 0. Gilt $0 < \text{Schiebezahl} \leq 16$, werden die Statusbits A0 und OV auf "0" zurückgesetzt. Ist die Schiebezahl "0", wird die Schiebeoperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
	31 16	15 0
vor Ausführung von SLW 5	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von SLW 5	0101	1111	0110	0100	1010	0111	0110	0000

Beispiel 1

AWL	Erläuterung	
L	MW4	//Lade den Wert nach AKKU 1.
SLW	5	//Schiebe die Bits in AKKU 1 um 5 Stellen nach links.
T	MW8	//Transferiere das Ergebnis nach MW8.

Beispiel 2

AWL	Erläuterung	
L	3	//Lade den Wert +3 in AKKU 1.
L	MW20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MW20 in AKKU 1.
SLW		//Die Schiebezahl ist der Wert von AKKU2-L-L. => Schiebe die Bits in AKKU1-L um 3 Stellen nach links.
SPP	NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt geschobene Bit (A1) = 1 ist.

11.1.5 SRW Schiebe rechts Wort (16 Bit)

Formate

SRW

SRW <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die geschoben werden soll; Bereich von 0 bis 15

Beschreibung

SRW (Schiebe rechts Wort) schiebt nur den Inhalt von AKKU1-L bitweise nach rechts. In die Bitstellen, die durch das Schieben frei werden, werden Nullen geschrieben. Das zuletzt geschobene Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die geschoben werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

SRW <Anzahl>: Die Schiebezahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte von 0 bis 15. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer als Null ist. Ist <Anzahl> gleich "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

SRW: Die Schiebezahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte von 0 bis 255. Eine Schiebezahl > 16 ruft immer das gleiche Ergebnis hervor: AKKU1-L = 0, A1 = 0, A0 = 0, OV = 0. Gilt $0 < \text{Schiebezahl} \leq 16$, werden die Statusbits A0 und OV auf "0" zurückgesetzt. Ist die Schiebezahl "0", wird die Schiebeoperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
	31 16	15 0
vor Ausführung von SRW 6	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von SRW 6	0101	1111	0110	0100	0000	0001	0111	0100

Beispiel 1

AWL	Erläuterung	
L	MW4	//Lade den Wert in AKKU 1.
SRW	6	//Schiebe die Bits in AKKU 1 um 6 Stellen nach rechts.
T	MW8	//Transferiere das Ergebnis in MW8.

Beispiel 2

AWL	Erläuterung	
L	3	//Lade den Wert +3 in AKKU 1.
L	MW20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MW20 in AKKU 1.
SRW		//Die Schiebezahl ist der Wert von AKKU2-L-L. => Schiebe die Bits in AKKU1-L um 3 Stellen nach rechts.
SPP	NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt geschobene Bit (A1) = 1 ist.

11.1.6 SLD Schiebe links Doppelwort (32 Bit)

Formate

SLD

SLD <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die geschoben werden soll; Bereich von 0 bis 32

Beschreibung

SLD (Schiebe links Doppelwort) schiebt den gesamten Inhalt von AKKU 1 bitweise nach links. In die Bitstellen, die durch das Schieben frei werden, werden Nullen geschrieben. Das zuletzt geschobene Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die geschoben werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

SLD <Anzahl>: Die Schiebezahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte von 0 bis 32. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer als Null ist. Ist <Anzahl> gleich "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

SLD: Die Schiebezahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte von 0 bis 255. Eine Schiebezahl > 32 ruft immer das gleiche Ergebnis hervor: AKKU 1 = 0, A1 = 0, A0 = 0, OV = 0. Gilt $0 < \text{Schiebezahl} \leq 32$, werden die Statusbits A0 und OV auf "0" zurückgesetzt. Ist die Schiebezahl "0", wird die Schiebeoperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
	31 16	15 0
vor Ausführung von SLD 5	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von SLD 5	1110	1100	1000	1011	1010	0111	0110	0000

Beispiel 1

AWL	Erläuterung	
L MD4	//Lade den Wert in AKKU 1.	
SLD 5	//Verschiebe die Bits in AKKU 1 um 5 Stellen nach links.	
T MD8	//Transferiere das Ergebnis nach MD8.	

Beispiel 2

AWL	Erläuterung	
L 3	//Lade den Wert +3 in AKKU 1.	
L MD20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD20 in AKKU 1.	
SLD	//Die Schiebezahl ist der Wert von AKKU2-L-L. => Schiebe die Bits in AKKU 1 um 3 Stellen nach links.	
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt geschobene Bit (A1) = 1 ist.	

11.1.7 SRD Schiebe rechts Doppelwort (32 Bit)

Formate

SRD

SRD <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die geschoben werden soll; Bereich von 0 bis 32

Beschreibung

SRD (Schiebe rechts Doppelwort) schiebt den gesamten Inhalt von AKKU 1 bitweise nach rechts. In die Bitstellen, die durch das Schieben frei werden, werden Nullen geschrieben. Das zuletzt geschobene Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die geschoben werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

SRD <Anzahl>: Die Schiebezahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte von 0 bis 32. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer als Null ist. Ist <Anzahl> gleich "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

SRD: Die Schiebezahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte von 0 bis 255. Eine Schiebezahl > 32 ruft immer das gleiche Ergebnis hervor: AKKU 1 = 0, A1 = 0, A0 = 0, OV = 0. Gilt $0 < \text{Schiebezahl} \leq 32$, werden die Statusbits A0 und OV auf "0" zurückgesetzt. Ist die Schiebezahl "0", wird die Schiebeoperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
	31 16	15 0
vor Ausführung von SRD 7	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von SRD 7	0000	0000	1011	1110	1100	1000	1011	1010

Beispiel 1

AWL	Erläuterung	
L MD4	//Lade den Wert in AKKU 1.	
SRD 7	//Verschiebe die Bits in AKKU 1 um 7 Stellen nach rechts.	
T MD8	//Transferiere das Ergebnis in MD8.	

Beispiel 2

AWL	Erläuterung	
L 3	//Lade den Wert +3 in AKKU 1.	
L MD20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD20 in AKKU 1.	
SRD	//Die Schiebezahl ist der Wert von AKKU2-L-L. => Schiebe die Bits in AKKU 1 um 3 Stellen nach rechts.	
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt geschobene Bit (A1) = 1 ist.	

11.2 Rotieroperationen

11.2.1 Rotieroperationen Übersicht

Beschreibung

Mit den Rotieroperationen können Sie den gesamten Inhalt von AKKU 1 bitweise nach rechts oder links rotieren. Die frei gewordenen Stellen werden mit den Signalzuständen der Bits aufgefüllt, die aus dem Akkumulator geschoben werden.

Die Zahl, die auf eine Rotieroperation folgt oder ein Wert im niederwertigen Byte des niederwertigen Worts von AKKU 2 gibt an, um wie viele Bits rotiert werden soll.

Je nach Operation wird die Rotation über das Bit A1 ausgeführt. Das Bit A0 im Statuswort wird auf "0" zurückgesetzt.

Folgende Rotieroperationen stehen Ihnen zur Verfügung:

- RLD Rotiere links Doppelwort (32 Bit)
- RRD Rotiere rechts Doppelwort (32 Bit)
- RLDA Rotiere Akku 1 links über A1-Anzeige (32 Bit)
- RRDA Rotiere Akku 1 rechts über A1-Anzeige (32 Bit)

11.2.2 RLD Rotiere links Doppelwort (32 Bit)

Formate

RLD

RLD <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die rotiert werden soll; Bereich von 0 bis 32

Beschreibung

RLD (Rotiere links Doppelwort) rotiert den gesamten Inhalt von AKKU 1 bitweise nach links. In die Bitstellen, die durch das Rotieren frei werden, werden die Signalzustände der Bits geschrieben, die aus dem AKKU 1 geschoben werden. Das zuletzt rotierte Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die rotiert werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

RLD <Anzahl>: Die Rotierzahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte zwischen 0 und 32. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer Null ist. Ist <Anzahl> gleich "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

RLD: Die Rotierzahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte zwischen 0 und 255. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn der Inhalt von AKKU2-L-L größer Null ist. Ist die Rotierzahl "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
Bit	31 16	15 0
vor Ausführung von RLD 4	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von RLD 4	1111	0110	0100	0101	1101	0011	1011	0101

Beispiel 1

AWL	Erläuterung	
L	MD2	//Lade den Wert in AKKU 1.
RLD	4	//Rotiere die Bits in AKKU 1 um 4 Stellen nach links.
T	MD8	//Transferiere das Ergebnis nach MD8.

Beispiel 2

AWL	Erläuterung	
L	3	//Lade den Wert +3 in AKKU 1.
L	MD20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD20 in AKKU 1.
RLD		//Die Rotierzahl ist der Wert von AKKU2-L-L. => Rotiere die Bits in AKKU 1 um 3 Stellen nach links.
SPP	NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt rotierte Bit (A1) = 1 ist.

11.2.3 RRD Rotiere rechts Doppelwort (32 Bit)

Formate

RRD

RRD <Anzahl>

Operand	Datentyp	Beschreibung
<Anzahl>	Ganzzahl, vorzeichenlos	Anzahl der Bitstellen, um die rotiert werden soll; Bereich von 0 bis 32

Beschreibung

RRD (Rotiere rechts Doppelwort) rotiert den gesamten Inhalt von AKKU 1 bitweise nach rechts. In die Bitstellen, die durch das Rotieren frei werden, werden die Signalzustände der Bits geschrieben, die aus dem AKKU 1 rotiert werden. Das zuletzt rotierte Bit wird in das Statusbit A1 geladen. Die Anzahl an Bitstellen, um die rotiert werden soll, wird entweder von dem Operanden <Anzahl> oder von einem Wert in AKKU2-L-L angegeben.

RRD <Anzahl>: Die Rotierzahl wird von dem Operanden <Anzahl> angegeben. Zulässig sind Werte von 0 bis 32. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn <Anzahl> größer als Null ist. Ist <Anzahl> gleich Null, wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

RRD: Die Rotierzahl wird von dem Wert in AKKU2-L-L angegeben. Zulässig sind Werte von 0 bis 255. Die Statusbits A0 und OV werden auf "0" zurückgesetzt, wenn der Inhalt von AKKU2-L-L größer als Null ist. Ist die Rotierzahl "0", wird die Rotieroperation wie eine Operation **NOP** bearbeitet.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	x	x	-	-	-	-	-

Beispiele

Inhalt	AKKU1-H				AKKU1-L			
	31 16	15 0
vor Ausführung von RRD 4	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von RRD 4	1011	0101	1111	0110	0100	0101	1101	0011

Beispiel 1

AWL	Erläuterung	
L	MD2	//Lade den Wert in AKKU 1.
RRD	4	//Rotiere die Bits in AKKU 1 um 4 Stellen nach rechts.
T	MD8	//Transferiere das Ergebnis nach MD8.

Beispiel 2

AWL	Erläuterung	
L	+3	//Lade den Wert +3 in AKKU 1.
L	MD20	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Wert von MD20 in AKKU 1.
RRD		//Die Rotierzahl ist der Wert von AKKU2-L-L. => Rotiere die Bits in AKKU 1 um 3 Stellen nach rechts.
SPP	NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt rotierte Bit (A1) = 1 ist.

11.2.4 RLDA Rotiere Akku 1 links über A1-Anzeige (32 Bit)

Format

RLDA

Beschreibung

RLDA (Rotiere links Doppelwort über A1) rotiert den Inhalt von AKKU 1 um eine Bitposition nach links herum durch das Anzeigebit A1. Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Inhalt	A1	AKKU1-H				AKKU1-L			
Bit		31 16	15 0
vor Ausführung von RLDA	X	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von RLDA	0	1011	1110	1100	1000	1011	1010	0111	011 X
	(X = 0 oder 1, alter Signalzustand von A1)								

AWL	Erläuterung
L MD2	//Lade den Inhalt von MD2 in AKKU 1.
RLDA	//Rotiere die Bits in AKKU 1 um eine Stelle nach links über A1.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt rotierte Bit (A1) = 1 ist.

11.2.5 RRDA Rotiere Akku 1 rechts über A1-Anzeige (32 Bit)

Format

RRDA

Beschreibung

RRDA (Rotiere rechts Doppelwort über A1) rotiert den Inhalt von AKKU1 um eine Bitposition nach rechts herum durch das Anzeigebit A1. Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Inhalt	A1	AKKU1-H				AKKU1-L			
Bit		31 16	15 0
vor Ausführung von RRDA	X	0101	1111	0110	0100	0101	1101	0011	1011
nach Ausführung von RRDA	1	X 010	1111	1011	0010	0010	1110	1001	1101
		(X = 0 oder 1, alter Signalzustand von A1)							

AWL	Erläuterung
L MD2	//Lade den Inhalt von MD2 in AKKU 1.
RRDA	//Rotiere die Bits in AKKU 1 um eine Stelle nach rechts über A1.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das zuletzt rotierte Bit (A1) = 1 ist.

12 Zeiten

12.1 Zeitoperationen Übersicht

Beschreibung

Unter "Speicherbereiche und Komponenten einer Zeit" finden Sie Informationen zum Einstellen und zur Auswahl der richtigen Zeit.

Folgende Zeitoperationen stehen Ihnen zur Verfügung:

- FR Freigabe Timer
- L Lade aktuellen Zeitwert als Ganzzahl in AKKU 1
- LC Lade aktuellen Zeitwert als BCD in AKKU 1
- R Rücksetze Timer
- SI Zeit als Impuls
- SV Zeit als verlängerter Impuls
- SE Zeit als Einschaltverzögerung
- SS Zeit als speichernde Einschaltverzögerung
- SA Zeit als Ausschaltverzögerung

12.2 Speicherbereiche und Komponenten einer Zeit

Speicherbereich

Zeiten haben einen eigenen reservierten Speicherbereich in Ihrer CPU. Dieser Speicherbereich reserviert ein 16-Bit-Wort für jeden Zeitoperanden. Das Programmieren mit FUP unterstützt 256 Zeiten. Wie viele Zeitworte in Ihrer CPU zur Verfügung stehen, entnehmen Sie bitte deren technischen Daten.

Folgende Funktionen greifen auf den Speicherbereich der Zeiten zu:

- Zeitoperationen
- Aktualisieren der Timerwörter über Zeitimpulsgeber. Diese Funktion Ihrer CPU im RUN-Zustand vermindert einen bestimmten Wert um jeweils eine Einheit in einem Intervall, das von der Zeitbasis festgelegt wurde, bis der Zeitwert gleich "0" ist.

Zeitwert

Die Bits 0 bis 9 des Timerworts enthalten den Zeitwert binär-codiert. Der Zeitwert gibt eine Anzahl von Einheiten an. Das Aktualisieren der Zeit vermindert den Zeitwert um jeweils eine Einheit in einem Intervall, der von der Zeitbasis festgelegt wurde. Der Zeitwert wird solange vermindert, bis er gleich "0" ist.

Mit der folgenden Syntax können Sie einen vordefinierten Zeitwert laden:

- L w#16#txyz
 - t = Zeitbasis (d.h. Zeitintervall oder Auflösung)
 - xyz = Zeitwert im BCD-Format
- L S5T#aH_bM_cS_dMS
 - H (Stunden), M (Minuten), S (Sekunden), MS (Millisekunden); a, b, c, d werden vom Anwender definiert.
 - Die Zeitbasis wird automatisch gewählt und der Wert zur nächstniederen Zahl mit dieser Zeitbasis gerundet

Sie können einen Zeitwert von max. 9 990 Sekunden bzw. 2H_46M_30S eingeben.

Zeitbasis

Die Bits 12 und 13 des Timerworts enthalten die Zeitbasis binär-codiert. Die Zeitbasis definiert das Intervall, in dem der Zeitwert um eine Einheit vermindert wird. Die kleinste Zeitbasis beträgt 10 ms, die größte 10 s.

Zeitbasis	Binär-code für Zeitbasis
10 ms	00
100 ms	01
1 s	10
10 s	11

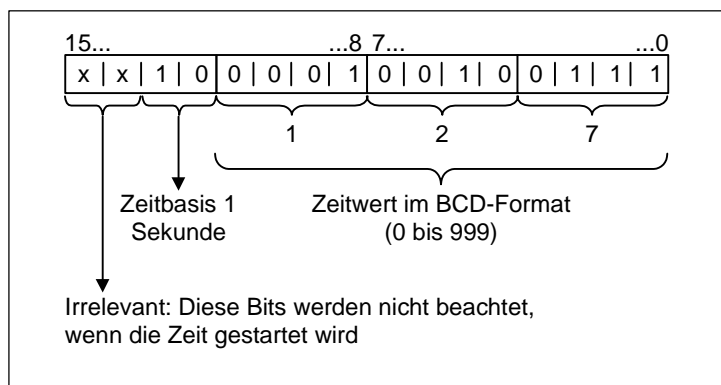
Die Werte dürfen 2H_46M_30S nicht überschreiten. Werte, die für einen Bereich oder für eine Auflösung zu groß sind, werden gerundet. Das allgemeine Format für den Datentyp S5TIME hat folgende Grenzwerte:

Auflösung	Bereich
0,01 Sekunde	10MS bis 9S_990MS
0,1 Sekunde	100MS bis 1M_39S_900MS
1 Sekunde	1S bis 16M_39S
10 Sekunden	10S bis 2H_46M_30S

Bit-Konfiguration in AKKU 1

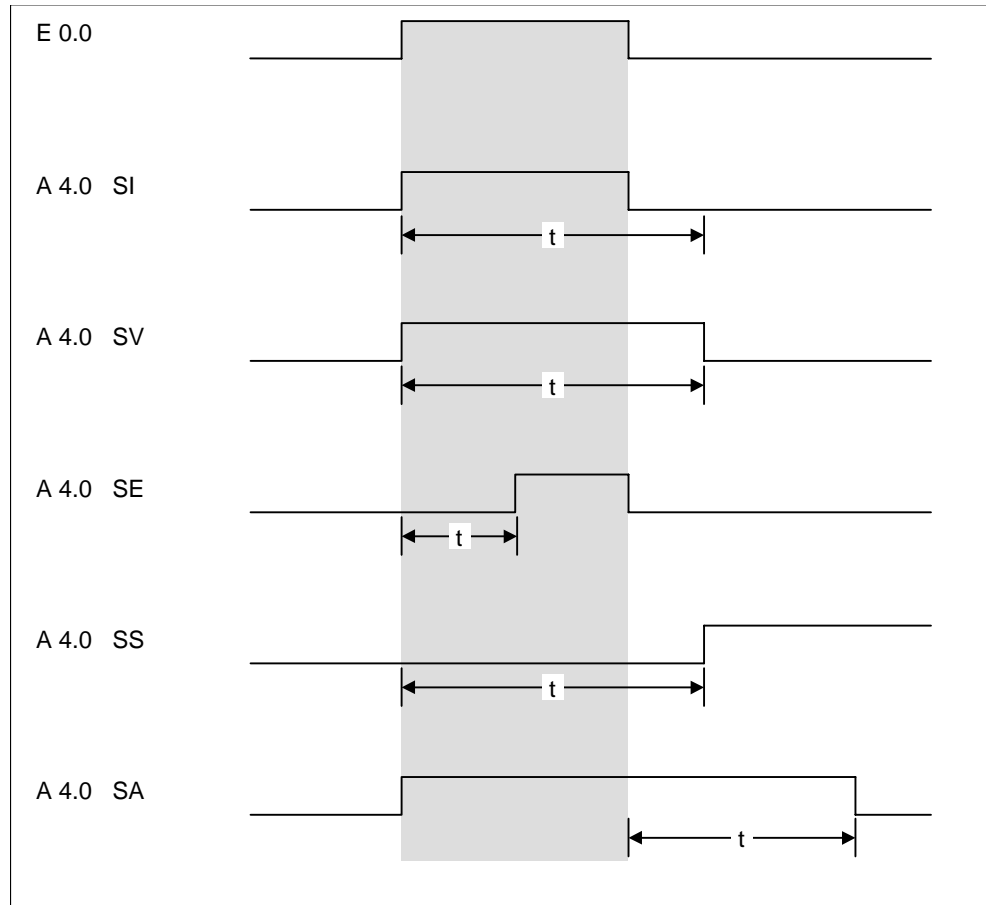
Wird eine Zeit gestartet, so wird der Inhalt des AKKU 1 als Zeitwert verwendet. Die Bits 0 bis 11 des AKKU1-L enthalten den Zeitwert im binär-codierten Dezimalformat (BCD-Format: jede Gruppe von vier Bits enthält den Binär-code für einen Dezimalwert). Die Bits 12 und 13 enthalten die Zeitbasis im Binär-code.

Folgendes Bild zeigt den Inhalt des AKKU1-L, nachdem Sie den Zeitwert 127 mit der Zeitbasis 1 Sekunde geladen haben:



Auswahl der richtigen Zeit

Die Übersicht über die 5 verschiedenen Zeiten soll Ihnen helfen, die für Ihre Zwecke adäquate Zeit auszuwählen.



Zeiten	Erklärung
SI Zeit als Impuls	Die maximale Zeit, in der das Ausgangssignal auf "1" bleibt, ist gleich dem programmierten Zeitwert t. Das Ausgangssignal bleibt für eine kürzere Zeit auf "1", wenn das Eingangssignal auf "0" wechselt.
SV Zeit als verlängerter Impuls	Das Ausgangssignal bleibt für die programmierte Zeit auf "1", unabhängig davon, wie lange das Eingangssignal auf "1" bleibt.
SE Zeit als Einschaltverzögerung	Das Ausgangssignal ist nur "1", wenn die programmierte Zeit abgelaufen ist und das Eingangssignal noch immer "1" beträgt.
SS Zeit als speichernde Einschaltverzögerung	Das Ausgangssignal wechselt nur von "0" auf "1", wenn die programmierte Zeit abgelaufen ist, unabhängig davon, wie lange das Eingangssignal auf "1" bleibt.
SA Zeit als Ausschaltverzögerung	Das Ausgangssignal ist "1", wenn das Eingangssignal "1" ist oder die Zeit läuft. Die Zeit wird gestartet wenn das Eingangssignal von "1" auf "0" wechselt.

12.3 FR Freigabe Timer

Format

FR <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

FR <Zeit> löscht den Flankenmerker, der für das Starten der adressierten Zeit verwendet wird, wenn das VKE von "0" auf "1" wechselt. Ein Wechsel des VKE-Bits von "0" auf "1" vor einer Operation Freigabe Timer (FR) gibt eine Zeit frei.

Die Operation Freigabe Timer ist für das normale Starten einer Zeit nicht erforderlich. Eine Freigabe wird nur dazu verwendet, eine laufende Zeit neu zu starten. Dies ist nur möglich, wenn die Startoperation weiterhin mit dem VKE = 1 bearbeitet wird.

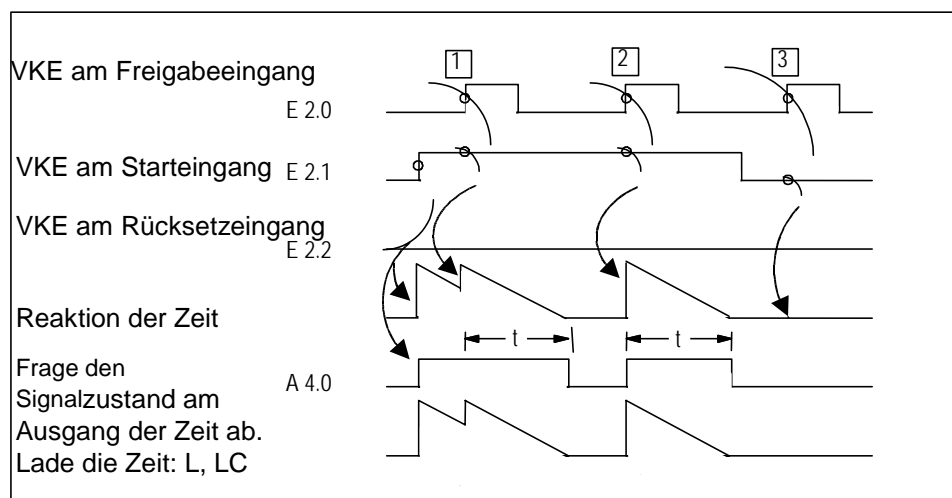
Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.0	
FR T1	//Gib die Zeit T1 frei.
U E 2.1	
L S5T#10s	//Richte eine Voreinstellung von 10 Sekunden in AKKU 1 ein.
SI T1	//Starte die Zeit T1 als Impuls.
U E 2.2	
R T1	//Setze Zeit T1 zurück.
U T1	//Frage den Signalzustand der Zeit T1 ab.
= A 4.0	
L T1	//Lade den aktuellen Zeitwert der Zeit T1 als Binärzahl.
T MW10	



t = programmierte Zeitdauer

- (1) Wechselt das VKE am Freigabeeingang von "0" auf "1"; während die Zeit läuft, wird die Zeit neu gestartet. Die programmierte Zeit ist die aktuelle Zeit für den Wiederanlauf. Wechselt das VKE am Freigabeeingang von "1" auf "0", so hat dies keinen Einfluß.
- (2) Wechselt das VKE am Freigabeeingang von "0" auf "1", und die Zeit läuft nicht, während am Starteingang ein VKE von "1" anliegt, wird die Zeit mit dem programmierten Zeitwert als Impuls gestartet.
- (3) Wechselt das VKE am Freigabeeingang von "0" auf "1"; während am Starteingang ein VKE von "0" anliegt, so hat dies keinen Einfluß auf die Zeit.

12.4 L Lade aktuellen Zeitwert als Ganzzahl in AKKU 1

Format

L <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

L <Zeit> lädt den aktuellen Zeitwert aus dem adressierten Timerwort ohne Zeitbasis als binäre Ganzzahl in den AKKU1-L, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 geladen wurde.

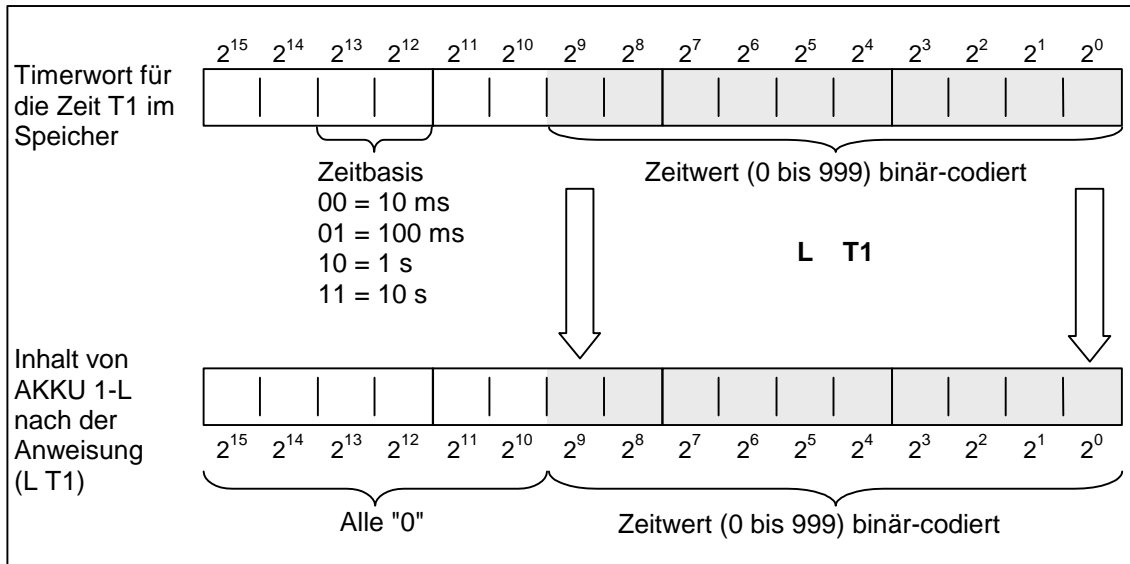
Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L T1	//Lade AKKU1-L mit dem aktuellen Zeitwert der Zeit T1 im Binärcode.



Hinweis

L<Zeit> lädt nur den Binärcode des aktuellen Zeitwerts in AKKU1-L, nicht die Zeitbasis. Der Zeitwert, der geladen wird, ist der Ausgangswert der Zeit minus der Zeit, die seit dem Start der Zeitfunktion abgelaufen ist.

12.5 LC Lade aktuellen Zeitwert als BCD in AKKU 1

Format

LC <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

LC <Zeit> lädt den aktuellen Zeitwert und die Zeitbasis aus dem adressierten Timerwort als binär-codierte Dezimalzahl (BCD) in AKKU 1, nachdem zuvor der Inhalt von AKKU 1 in AKKU 2 geladen wurde.

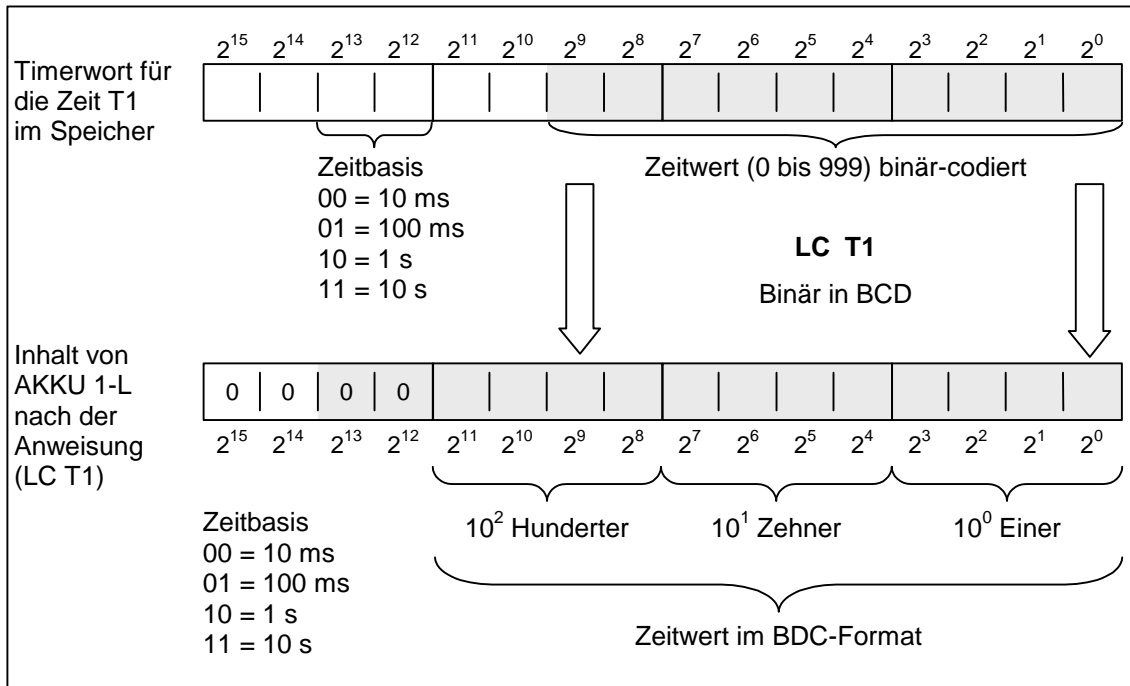
Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
LC T1	//Lade AKKU1-L mit der Zeitbasis und dem aktuellen Zeitwert der Zeit T1 im BCD-Format in AKKU1-L.



12.6 R Rücksetze Timer

Format

R <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

R <Zeit> beendet die aktuelle Zeitfunktion und löscht den Zeitwert und die Zeitbasis des adressierten Timerworts, wenn das VKE von "0" nach "1" wechselt.

Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.1	
R T1	//Frage den Signalzustand am Eingang E 2.1 ab. Wenn das VKE von "0" nach "1" wechselt ist, dann setze die Zeit T1 zurück.

12.7 SI Zeit als Impuls

Format

SI <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

SI <Zeit> startet die adressierte Zeit, wenn das VKE von "0" auf "1" wechselt. Die programmierte Zeitdauer läuft ab, solange das VKE = 1 ist. Wechselt das VKE auf "0", bevor die Zeit abgelaufen ist, wird die Zeit angehalten. Für diese Operation (Starten der Zeit) müssen der Zeitwert und die Zeitbasis im BCD-Format in AKKU1-L gespeichert sein.

Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

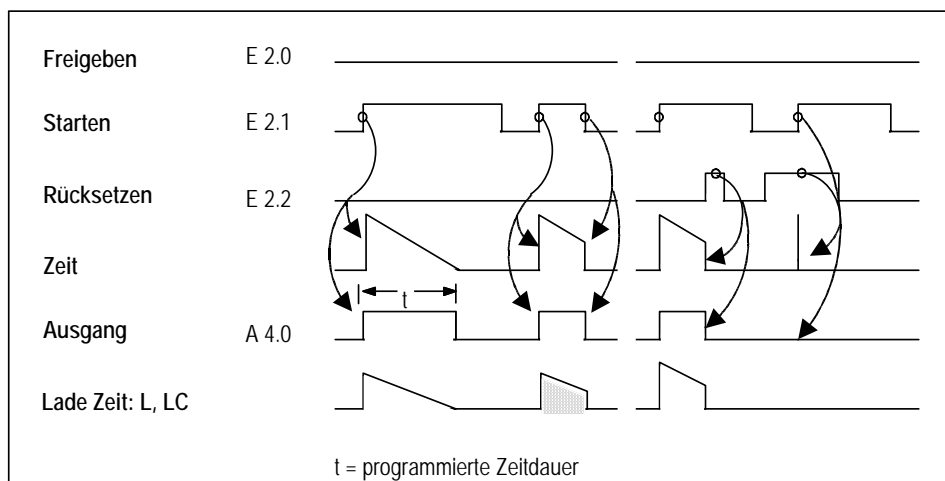
Siehe auch Speicherbereiche und Komponenten einer Zeit.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.0	
FR T1	//Gib die Zeit T1 frei.
U E 2.1	
L S5T#10s	//Richte eine Voreinstellung von 10 Sekunden in AKKU 1 ein.
SI T1	//Starte die Zeit T1 als Impuls.
U E 2.2	
R T1	//Setze die Zeit T1 zurück.
U T1	//Frage den Signalzustand der Zeit T1 ab.
= A 4.0	
L T1	//Lade den aktuellen Zeitwert der Zeit T1 als Binärzahl.
T MW10	
LC T1	//Lade den aktuellen Zeitwert der Zeit T1 im BCD-Format.
T MW12	



12.8 SV Zeit als verlängerter Impuls

Format

SV <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

SV <Zeit> startet die adressierte Zeit, wenn das VKE von "0" auf "1" wechselt. Die programmierte Zeitdauer läuft ab, auch wenn das VKE inzwischen auf "0" wechselt. Wechselt das VKE von "0" auf "1", bevor die programmierte Zeit abgelaufen ist, wird die programmierte Zeitdauer neu gestartet. Für diesen Befehl zum Starten der Zeit müssen der Zeitwert und die Zeitbasis im BCD-Format in AKKU1-L gespeichert sein.

Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

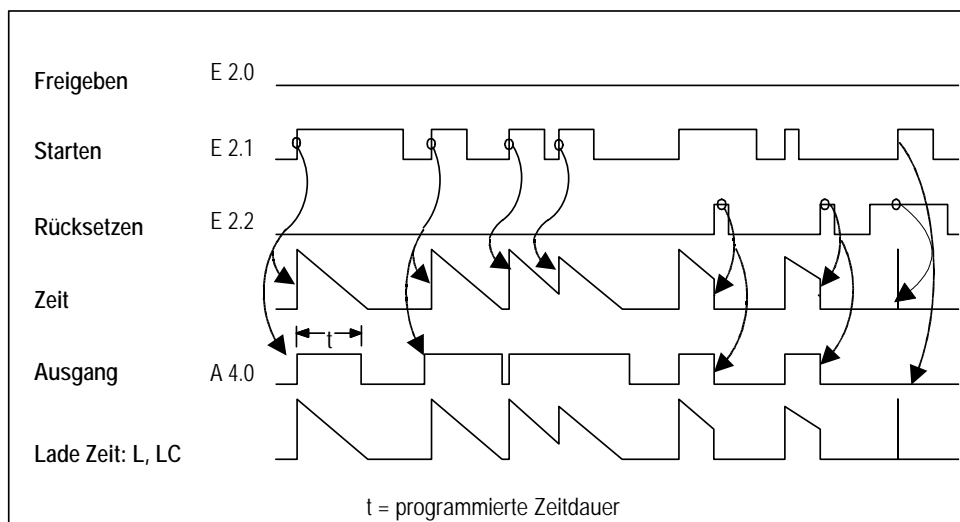
Siehe auch Speicherbereiche und Komponenten einer Zeit.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.0	
FR T1	//Gib die Zeit T1 frei.
U E 2.1	
L S5T#10s	//Richte eine Voreinstellung von 10 Sekunden in AKKU 1 ein.
SV T1	//Starte die Zeit T1 als verlängerten Impuls.
U E 2.2	
R T1	//Setze die Zeit T1 zurück.
U T1	//Frage den Signalzustand der Zeit T1 ab.
= A 4.0	
L T1	//Lade den aktuellen Zeitwert der Zeit T1 als Binärzahl.
T MW10	
LC T1	//Lade den aktuellen Zeitwert der Zeit T1 im BCD-Format.
T MW12	



12.9 SE Zeit als Einschaltverzögerung

Format

SE <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

SE <Zeit> startet die adressierte Zeit, wenn das VKE von "0" auf "1" wechselt. Die programmierte Zeitdauer läuft ab, solange das VKE = 1 ist. Wechselt das VKE auf "0", bevor die programmierte Zeitdauer abgelaufen ist, wird die Zeit angehalten. Für diese Operation (Starten der Zeit) müssen der Zeitwert und die Zeitbasis im BCD-Format in AKKU1-L gespeichert sein.

Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

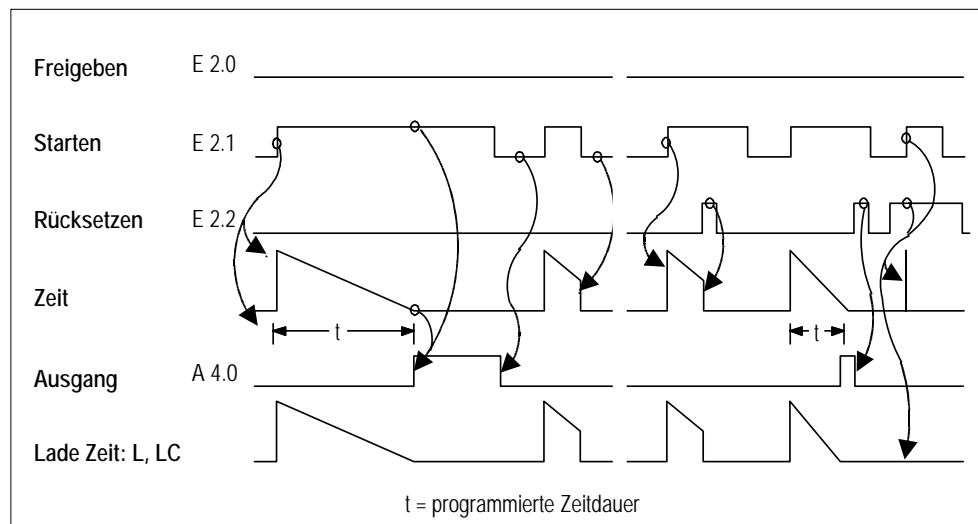
Siehe auch Speicherbereiche und Komponenten einer Zeit.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.0	
FR T1	//Gib die Zeit T1 frei.
U E 2.1	
L S5T#10s	//Richte eine Voreinstellung von 10 Sekunden in AKKU 1 ein.
SE T1	//Starte die Zeit T1 als Einschaltverzögerung.
U E 2.2	
R T1	//Setze die Zeit T1 zurück.
U T1	//Frage den Signalzustand der Zeit T1 ab.
= A 4.0	
L T1	//Lade den aktuellen Zeitwert der Zeit T1 als Binärzahl.
T MW10	
LC T1	//Lade den aktuellen Zeitwert der Zeit T1 im BCD-Format.
T MW12	



12.10 SS Zeit als speichernde Einschaltverzögerung

Format

SS <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

SS <Zeit> (Starte Zeit als speichernde Einschaltverzögerung) startet die adressierte Zeit, wenn das VKE von "0" auf "1" wechselt. Die programmierte Zeitdauer läuft ab, auch wenn das VKE inzwischen auf "0" wechselt. Wechselt das VKE von "0" auf "1", bevor die programmierte Zeit abgelaufen ist, wird die programmierte Zeitdauer erneut gestartet. Für diese Operation (Starten der Zeit) müssen der Zeitwert und die Zeitbasis im BCD-Format in AKKU1-L gespeichert sein.

Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

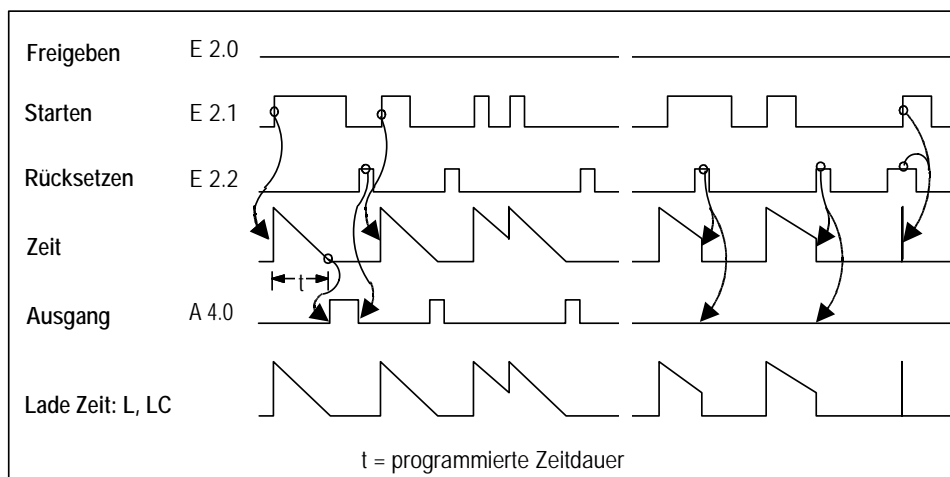
Siehe auch Speicherbereiche und Komponenten einer Zeit.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.0	
FR T1	//Gib die Zeit T1 frei.
U E 2.1	
L S5T#10s	//Richte eine Voreinstellung von 10 Sekunden in AKKU 1 ein.
SS T1	//Starte die Zeit T1 als speichernde Einschaltverzögerung.
U E 2.2	
R T1	//Setze die Zeit T1 zurück.
U T1	//Frage den Signalzustand der Zeit T1 ab.
= A 4.0	
L T1	//Lade den aktuellen Zeitwert der Zeit T1 als Binärzahl.
T MW10	
LC T1	//Lade den aktuellen Zeitwert der Zeit T1 im BCD-Format.
T MW12	



12.11 SA Zeit als Ausschaltverzögerung

Format

SA <Zeit>

Operand	Datentyp	Speicherbereich	Beschreibung
<Zeit>	TIMER	T	Nummer der Zeit; Bereich hängt von der CPU ab

Beschreibung

SA <Zeit> startet die adressierte Zeit, wenn das VKE von "1" auf "0" wechselt. Die programmierte Zeit läuft ab, solange das VKE = 0 ist. Wechselt das VKE auf "1", bevor die programmierte Zeitdauer abgelaufen ist, wird die Zeit angehalten. Für diesen Befehl zum Starten der Zeit müssen der Zeitwert und die Zeitbasis im BCD-Format in AKKU1-L gespeichert sein.

Auch wenn die Zeit nicht gestartet wird (VKE=0), muß im AKKU1-L eine Zahl im BCD-Format gespeichert sein.

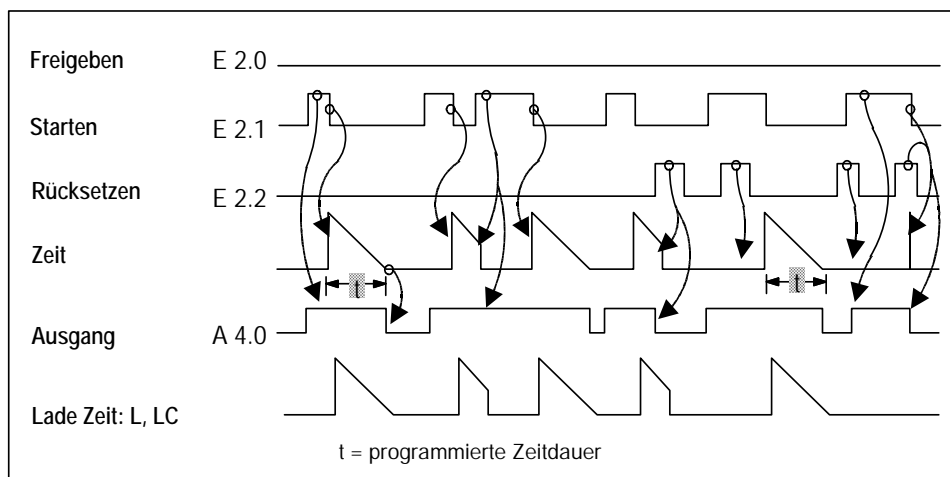
Siehe auch Speicherbereiche und Komponenten einer Zeit.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	0	-	-	0

Beispiel

AWL	Erläuterung
U E 2.0	
FR T1	//Gib die Zeit T1 frei.
U E 2.1	
L S5T#10s	//Richte eine Voreinstellung von 10 Sekunden in AKKU 1 ein.
SA T1	//Starte die Zeit T1 als Ausschaltverzögerung.
U E 2.2	
R T1	//Setze die Zeit T1 zurück.
U T1	//Frage den Signalzustand der Zeit T1 ab.
= A 4.0	
L T1	//Lade den aktuellen Zeitwert der Zeit T1 als Binärzahl.
T MW10	
LC T1	//Lade den aktuellen Zeitwert der Zeit T1 im BCD-Format.
T MW12	



13 Wortverknüpfung

13.1 Wortverknüpfungsoperationen Übersicht

Beschreibung

Durch Wortverknüpfungsoperationen werden Paare von Wörtern (16 Bits) oder Doppelwörtern (32 Bits) entsprechend der Booleschen Logik bitweise miteinander verknüpft. Jedes der beiden Wörter oder Doppelwörter muss sich in einem der beiden Akkumulatoren befinden.

Bei der Verknüpfung von Wörtern wird der Inhalt des niederwertigen Worts von AKKU 2 mit dem Inhalt des niederwertigen Worts von AKKU 1 verknüpft. Das Verknüpfungsergebnis wird im niederwertigen Wort von AKKU 1 gespeichert, wobei der alte Inhalt überschrieben wird.

Bei der Verknüpfung von Doppelwörtern wird der Inhalt von AKKU 2 mit dem Inhalt von AKKU 1 verknüpft. Das Verknüpfungsergebnis wird in AKKU 1 gespeichert, wobei der alte Inhalt überschrieben wird.

Folgende Operationen stehen Ihnen für Wortverknüpfungen zur Verfügung:

- UW UND-Wort (16 Bit)
- OW ODER-Wort (16 Bit)
- XOW EXKLUSIV-ODER-Wort (16 Bit)
- UD UND-Doppelwort (32 Bit)
- OD ODER-Doppelwort (32 Bit)
- XOD EXKLUSIV-ODER-Doppelwort (32 Bit)

13.2 UW UND-Wort (16 Bit)

Format

UW

UW <Konstante>

Operand	Datentyp	Beschreibung
<Konstante>	WORD, Konstante (16 Bit)	Bitmuster, das mit AKKU1-L durch UND verknüpft werden soll.

Beschreibung

UW (UND-Wort) verknüpft den Inhalt von AKKU1-L mit AKKU2-L bzw. einer Konstanten (16 Bit) bitweise gemäß der booleschen UND-Verknüpfung. Nur wenn die entsprechenden Bits von beiden Wörtern, die verknüpft werden sollen, "1" sind, ist das Bit im Ergebniswort "1". Das Ergebnis wird in AKKU1-L gespeichert. AKKU1-H und AKKU 2 (und, bei CPUs mit vier Akkus, AKKU 3 und AKKU 4) werden nicht verändert. Als Ergebnis der Operation wird das Statusbit A1 gesetzt (A1 = 1, wenn das Ergebnis ungleich Null ist). Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

UW: Verknüpft AKKU1-L mit AKKU2-L.

UW <Konstante>: Verknüpft AKKU1-L mit einer Konstanten (16 Bit).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Bit	15 0
AKKU 1 vor Ausführung von UW	0101	1001	0011	1011
AKKU2-L oder Konstante (16 Bit)	1111	0110	1011	0101
Ergebnis (AKKU 1) nach Ausführung von UW	0101	0000	0011	0001

Beispiel 1

AWL	Erläuterung
L EW20	//Lade den Inhalt von EW20 in AKKU1-L.
L EW22	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Inhalt von EW22 in AKKU 1-L.
UW	//Verknüpfe die Bits von AKKU1-L mit den Bits von AKKU2-L durch UND, speichere das Ergebnis in AKKU1-L.
T MW 8	//Transferiere das Ergebnis nach MW8.

Beispiel 2

AWL	Erläuterung
L EW20	//Lade den Inhalt von EW20 in AKKU1-L.
UW W#16#0FFF	//Verknüpfe die Bits von AKKU1-L mit dem Bitmuster der Konstanten (16 Bit) (0000_1111_1111_1111) durch UND, speichere das Ergebnis in AKKU1-L.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das Ergebnis ungleich Null ist (A1 = 1).

13.3 OW ODER-Wort (16 Bit)

Format

OW

OW <Konstante>

Operand	Datentyp	Beschreibung
<Konstante>	WORD, Konstante (16 Bit)	Bitmuster, das mit AKKU1-L durch ODER verknüpft werden soll.

Beschreibung

OW (ODER-Wort) verknüpft den Inhalt von AKKU1-L mit AKKU2-L bzw. einer Konstanten (16 Bit) bitweise gemäß der booleschen ODER-Verknüpfung. Wenn mindestens eines der entsprechenden Bits der beiden Wörter, die verknüpft werden sollen, "1" ist, ist das Bit im Ergebniswort "1". Das Ergebnis wird in AKKU1-L gespeichert. AKKU1-H und AKKU 2 (und, bei CPUs mit vier Akkus, AKKU 3 und AKKU 4) werden nicht verändert. Als Ergebnis der Operation wird das Statusbit A1 gesetzt (A1 = 1, wenn das Ergebnis ungleich Null ist). Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

OW: Verknüpft AKKU1-L mit AKKU2-L.

OW <Konstante>: Verknüpft AKKU1-L mit einer Konstanten (16 Bit).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Bit	15 0
AKKU 1 vor Ausführung von OW	0101	0101	0011	1011
AKKU2-L oder Konstante (16 Bit)	1111	0110	1011	0101
Ergebnis (AKKU 1) nach Ausführung von OW	1111	0111	1011	1111

Beispiel 1

AWL	Erläuterung
L EW20	//Lade den Inhalt von EW20 in AKKU1-L.
L EW22	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Inhalt von EW22 in AKKU 1-L.
OW	//Verknüpfe die Bits von AKKU1-L mit den Bits von AKKU2-L durch ODER, speichere das Ergebnis in AKKU1-L.
T MW8	//Transferiere das Ergebnis nach MW 8.

Beispiel 2

AWL	Erläuterung
L EW20	//Lade den Inhalt von EW20 in AKKU1-L.
OW W#16#0FFF	//Verknüpfe die Bits von AKKU1-L mit dem Bitmuster der Konstanten (16 Bit) (0000_1111_1111_1111) durch ODER, speichere das Ergebnis in AKKU1-L.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das Ergebnis ungleich Null ist (A1 = 1).

13.4 XOW EXKLUSIV-ODER-Wort (16 Bit)

Format

XOW

XOW <Konstante>

Operand	Datentyp	Beschreibung
<Konstante>	WORD, Konstante (16 Bit)	Bitmuster, das mit AKKU1-L durch EXKLUSIV ODER verknüpft werden soll.

Beschreibung

XOW (EXKLUSIV-ODER-Wort) verknüpft den Inhalt von AKKU1-L mit AKKU2-L bzw. einer Konstanten (16 Bit) bitweise gemäß der booleschen EXKLUSIV-ODER-Verknüpfung. Wenn nur eines der entsprechenden Bits der beiden Wörter, die verknüpft werden sollen, "1" ist, ist das Bit im Ergebniswort "1". Das Ergebnis wird in AKKU1-L gespeichert. AKKU1-H und AKKU 2 (und, bei CPUs mit vier Akkus, AKKU 3 und AKKU 4) werden nicht verändert. Als Ergebnis der Operation wird das Statusbit A1 gesetzt (A1 = 1, wenn das Ergebnis ungleich Null ist). Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

Sie können die Exklusiv-ODER-Funktion auch mehrfach nacheinander anwenden. Dann ist das gemeinsame Verknüpfungsergebnis "1", wenn eine ungerade Anzahl der abgefragten Operanden das Abfrageergebnis "1" liefert.

XOW: Verknüpft AKKU1-L mit AKKU2-L.

XOW <Konstante>: Verknüpft AKKU1-L mit einer Konstanten (16 Bit).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Bit	15 0
AKKU 1 vor Ausführung von XOW	0101	0101	0011	1011
AKKU2-L oder Konstante (16 Bit)	1111	0110	1011	0101
Ergebnis (AKKU 1) nach Ausführung von XOW	1010	0011	1000	1110

Beispiel 1

AWL	Erläuterung
L EW20	//Lade den Inhalt von EW20 in AKKU1-L.
L EW22	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Inhalt von ED24 in AKKU 1-L.
XOW	//Verknüpfe die Bits von AKKU1-L mit den Bits von AKKU2-L durch EXKLUSIV ODER, speichere das Ergebnis in AKKU1-L.
T MW8	//Transferiere das Ergebnis nach MW8.

Beispiel 2

AWL	Erläuterung
L EW20	//Lade den Inhalt von EW20 in AKKU1-L.
XOW 16#0FFF	//Verknüpfe die Bits von AKKU1-L mit dem Bitmuster der Konstanten (16 Bit) (0000_1111_1111_1111) durch EXKLUSIV ODER, speichere das Ergebnis in AKKU1-L.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das Ergebnis ungleich Null ist (A1 = 1).

13.5 UD UND-Doppelwort (32 Bit)

Format

UD

UD <Konstante>

Operand	Datentyp	Beschreibung
<Konstante>	DWORD, Konstante (32 Bit)	Bitmuster, das mit AKKU1 durch UND verknüpft werden soll.

Beschreibung

UD (UND-Doppelwort) verknüpft den Inhalt von AKKU 1 mit AKKU 2 bzw. einer Konstanten (32 Bit) bitweise gemäß der booleschen UND-Verknüpfung. Nur wenn die entsprechenden Bits von beiden Doppelwörtern, die verknüpft werden sollen, "1" sind, ist das Bit im Ergebnisdoppelwort "1". Das Ergebnis wird in AKKU 1 gespeichert. AKKU 2 (und, bei CPUs mit vier Akkus, AKKU 3 und AKKU 4) wird nicht verändert. Als Ergebnis der Operation wird das Statusbit A1 gesetzt (A1 = 1, wenn das Ergebnis ungleich Null ist). Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

UD: Verknüpft AKKU 1 mit AKKU 2.

UD <Konstante>: Verknüpft AKKU 1 mit einer Konstanten (32 Bit).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Bit	31 0
AKKU 1 vor Ausführung von UD	0101	0000	1111	1100	1000	1001	0011	1011
AKKU 2 oder Konstante (32 Bit):	1111	0011	1000	0101	0111	0110	1011	0101
Ergebnis (AKKU 1) nach Ausführung von UD	0101	0000	1000	0100	0000	0000	0011	0001

Beispiel 1

AWL	Erläuterung
L ED20	//Lade den Inhalt von ED20 in AKKU 1.
L ED24	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Inhalt von ED24 in AKKU 1.
UD	//Verknüpfe die Bits von AKKU 1 mit den Bits von AKKU 2 durch UND, speichere das Ergebnis in AKKU 1.
T MD8	//Transferiere das Ergebnis nach MD8.

Beispiel 2

AWL	Erläuterung
L ED 20	//Lade den Inhalt von ED20 in AKKU 1.
UD DW#16#0FFF_EF2 1	//Verknüpfe die Bits von AKKU 1 mit dem Bitmuster der Konstanten (32 Bit) (0000_1111_1111_1111_1110_1111_0010_0001) durch UND, speichere das Ergebnis in AKKU 1.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das Ergebnis ungleich Null ist (A1 = 1).

13.6 OD ODER-Doppelwort (32 Bit)

Format

OD

OD <Konstante>

Operand	Datentyp	Beschreibung
<Konstante>	DWORD, Konstante (32 Bit)	Bitmuster, das mit AKKU1 durch ODER verknüpft werden soll.

Beschreibung

OD (ODER-Doppelwort) verknüpft den Inhalt von AKKU 1 mit AKKU 2 bzw. einer Konstanten (32 Bit) bitweise gemäß der booleschen ODER-Verknüpfung. Wenn mindestens eines der entsprechenden Bits der beiden Doppelwörter, die verknüpft werden sollen, "1" ist, dann ist das Bit im Ergebnisdoppelwort "1". Das Ergebnis wird in AKKU 1 gespeichert. AKKU 2 (bei CPUs mit vier Akkus auch AKKU 3 und AKKU 4) wird nicht verändert. Als Ergebnis der Operation wird das Statusbit A1 gesetzt (A1 = 1, wenn das Ergebnis ungleich Null ist). Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

OD: Verknüpft AKKU 1 mit AKKU 2.

OD <Konstante>: Verknüpft AKKU 1 mit einer Konstanten (32 Bit).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Bit	31 0
AKKU 1 vor Ausführung von OD	0101	0000	1111	1100	1000	0101	0011	1011
AKKU 2 oder Konstante (16 Bit):	1111	0011	1000	0101	0111	0110	1011	0101
Ergebnis (AKKU 1) nach Ausführung von OD	1111	0011	1111	1101	1111	0111	1011	1111

Beispiel 1

AWL	Erläuterung
L ED20	//Lade den Inhalt von ED20 in AKKU 1.
L ED24	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Inhalt von ED24 in AKKU 1.
OD	//Verknüpfe die Bits von AKKU 1 mit den Bits von AKKU 2 durch ODER, speichere das Ergebnis in AKKU 1.
T MD8	//Transferiere das Ergebnis nach MD8.

Beispiel 2

AWL	Erläuterung
L ED20	//Lade den Inhalt von ED20 in AKKU 1.
OD DW#16#0FFF_EF21	//Verknüpfe die Bits von AKKU 1 mit dem Bitmuster der Konstanten (32 Bit) (0000_1111_1111_1111_1110_1111_0010_0001) durch ODER, speichere das Ergebnis in AKKU 1.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das Ergebnis ungleich Null ist (A1 = 1).

13.7 XOD EXKLUSIV-ODER-Doppelwort (32 Bit)

Format

XOD

XOD <Konstante>

Operand	Datentyp	Beschreibung
<Konstante>	DWORD, Konstante (32 Bit)	Bitmuster, das mit AKKU1 durch EXKLUSIV ODER verknüpft werden soll.

Beschreibung

XOD (EXKLUSIV-ODER-Doppelwort) verknüpft den Inhalt von AKKU 1 mit AKKU 2 bzw. einer Konstanten (32 Bit) bitweise gemäß der booleschen EXKLUSIV-ODER-Verknüpfung. Wenn nur eines der entsprechenden Bits der beiden Doppelwörter, die verknüpft werden sollen, "1" ist, ist das Bit im Ergebnisdoppelwort "1". Das Ergebnis wird in AKKU 1 gespeichert. AKKU 2 (und, bei CPUs mit vier Akkus, AKKU 3 und AKKU 4) wird nicht verändert. Als Ergebnis der Operation wird das Statusbit A1 gesetzt (A1 = 1, wenn das Ergebnis ungleich Null ist). Die Statusbits A0 und OV werden auf "0" zurückgesetzt.

Sie können die Exklusiv-ODER-Funktion auch mehrfach nacheinander anwenden. Dann ist das gemeinsame Verknüpfungsergebnis "1", wenn eine ungerade Anzahl der abgefragten Operanden das Abfrageergebnis "1" liefert.

XOD: Verknüpft AKKU 1 mit AKKU 2.

XOD <Konstante>: Verknüpft AKKU 1 mit einer Konstanten (32 Bit).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	x	0	0	-	-	-	-	-

Beispiele

Bit	31 0
AKKU 1 vor Ausführung von XOD	0101	0000	1111	1100	1000	0101	0011	1011
AKKU 2 oder Konstante (32 Bit):	1111	0011	1000	0101	0111	0110	1011	0101
Ergebnis (AKKU 1) nach Ausführung von XOD	1010	0011	0111	1001	1111	0011	1000	1110

Beispiel 1

AWL	Erläuterung
L ED20	//Lade den Inhalt von ED20 in AKKU 1.
L ED24	//Lade den Inhalt von AKKU 1 in AKKU 2. Lade den Inhalt von ED24 in AKKU 1.
XOD	//Verknüpfe die Bits von AKKU 1 mit den Bits von AKKU 2 durch EXKLUSIV ODER, speichere das Ergebnis in AKKU 1.
T MD8	//Transferiere das Ergebnis nach MD8.

Beispiel 2

AWL	Erläuterung
L ED20	//Lade den Inhalt von ED20 in AKKU 1.
XOD DW#16#0FFF_EF21	//Verknüpfe die Bits von AKKU 1 mit dem Bitmuster der Konstanten (32 Bit) (0000_1111_1111_1111_1111_1110_0010_0001) durch EXKLUSIV ODER, speichere das Ergebnis in AKKU 1.
SPP NEXT	//Springe zur Sprungmarke NEXT, wenn das Ergebnis ungleich Null ist (A1 = 1).

14 Akkumulator-Operationen

14.1 Akkumulatoroperationen Übersicht

Beschreibung

Folgende Operationen stehen Ihnen zur Verfügung, um den Inhalt von einem oder mehreren Akkumulatoren bzw. Adreßregistern zu bearbeiten:

- TAK Tausche AKKU 1 mit AKKU 2
- PUSH CPU mit zwei Akkus
- PUSH CPU mit vier Akkus
- POP CPU mit zwei Akkus
- POP CPU mit vier Akkus

- ENT Enter AKKU-Stack
- LEAVE Leave AKKU-Stack
- INC Inkrementiere AKKU 1-L-L
- DEC Dekrementiere AKKU 1-L-L

- +AR1 Addiere AKKU 1 zum Adreßregister 1
- +AR2 Addiere AKKU 1 zum Adreßregister 2

- BLD Bildbefehl (Nulloperation)
- NOP 0 Nulloperation
- NOP 1 Nulloperation

14.2 TAK Tausche AKKU 1 mit AKKU 2

Format

TAK

Beschreibung

TAK (Tausche AKKU 1 mit AKKU 2) tauscht den Inhalt von AKKU 1 mit dem Inhalt von AKKU 2. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen. Die Inhalte von AKKU 3 und AKKU 4 bleiben unverändert (bei CPUs mit vier Akkus).

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel: Subtrahiere den kleineren Wert vom größeren Wert

AWL	Erläuterung	
L MW10		//Lade den Inhalt von MW10 in AKKU1-L.
L MW12		//Lade den Inhalt von AKKU1-L in AKKU2-L. Lade den Inhalt von MW12 //in AKKU1-L.
>I		//Prüfe, ob AKKU2-L (MW10) größer ist als AKKU1-L (MW12).
SPB NEXT		//Springe zu Sprungmarke NEXT, wenn AKKU 2 (MW10) größer ist als //AKKU 1 (MW12).
TAK		//Tausche die Inhalte von AKKU 1 und AKKU 2.
NEXT: -I		//Subtrahiere den Inhalt von AKKU1-L vom Inhalt von AKKU2-L.
T MW14		//Transferiere das Ergebnis (= größerer Wert minus kleinerer Wert) //nach MW14.

Inhalt	AKKU 1	AKKU 2
vor der Ausführung von TAK	<MW12>	<MW10>
nach der Ausführung von TAK	<MW10>	<MW12>

14.3 PUSH CPU mit zwei Akkus

Format

PUSH

Beschreibung

PUSH (AKKU 1 in AKKU 2) kopiert den gesamten Inhalt von AKKU 1 in AKKU 2. AKKU 1 wird nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MW10	//Lade den Inhalt von MW10 in AKKU 1.
PUSH	//Kopiere den gesamten Inhalt von AKKU 1 in AKKU 2.

Inhalt	AKKU 1	AKKU 2
vor der Ausführung von PUSH	<MW10>	<X>
nach der Ausführung von PUSH	<MW10>	<MW10>

14.4 PUSH CPU mit vier Akkus

Format

PUSH

Beschreibung

PUSH (CPU mit vier Akkus) kopiert den Inhalt von AKKU 3 in AKKU 4, den Inhalt von AKKU 2 in AKKU 3, und den Inhalt von AKKU 1 in AKKU 2. AKKU 1 wird nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MW10	//Lade den Inhalt von MW10 in AKKU 1.
PUSH	//Kopiere den gesamten Inhalt von AKKU 1 in AKKU 2, den Inhalt von AKKU 2 in AKKU 3, und den Inhalt von AKKU 3 in AKKU 4.

Inhalt	AKKU 1	AKKU 2	AKKU 3	AKKU 4
vor der Ausführung von PUSH	Wert A	Wert B	Wert C	Wert D
nach der Ausführung von PUSH	Wert A	Wert A	Wert B	Wert C

14.5 POP CPU mit zwei Akkus

Format

POP

Beschreibung

POP (CPU mit zwei Akkus) kopiert den gesamten Inhalt von AKKU 2 in AKKU 1. AKKU 2 wird nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
T MD10	//Transferiere den Inhalt von AKKU 1 (= Wert A) nach MD10.
POP	//Kopiere den gesamten Inhalt von AKKU 2 in AKKU 1.
T MD14	//Transferiere den Inhalt von von AKKU 1 (= Wert B) nach MD14.

Inhalt	AKKU 1	AKKU 2
vor der Ausführung von POP	Wert A	Wert B
nach der Ausführung von POP	Wert B	Wert B

14.6 POP CPU mit vier Akkus

Format

POP

Beschreibung

POP (CPU mit vier Akkus) kopiert den Inhalt von AKKU 2 in AKKU 1, den Inhalt von AKKU 3 in AKKU 2, und den Inhalt von AKKU 4 in AKKU 3. AKKU 4 wird nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
T MD10	//Transferiere den Inhalt von AKKU 1 (= Wert A) nach MD10.
POP	//Kopiere den Inhalt von AKKU 2 in AKKU 1, den Inhalt von AKKU 3 in AKKU 2 und den Inhalt von AKKU 4 in AKKU 3.
T MD14	//Transferiere den Inhalt von von AKKU 1 (= Wert B) nach MD14.

Inhalt	AKKU 1	AKKU 2	AKKU 3	AKKU 4
vor der Ausführung von POP	Wert A	Wert B	Wert C	Wert D
nach der Ausführung von POP	Wert B	Wert C	Wert D	Wert D

14.7 ENT Enter AKKU-Stack

Format

ENT

Beschreibung

ENT (Enter AKKU-Stack) kopiert den Inhalt von AKKU 3 in AKKU 4 und den Inhalt von AKKU 2 in AKKU 3. Wenn Sie die Operation **ENT** direkt vor einer Ladeoperation programmieren, können Sie damit ein Zwischenergebnis in AKKU 3 retten.

Beispiel

AWL	Erläuterung
L DBD0	//Lade den Wert aus Datendoppelwort DBD0 in AKKU 1. (Dieser Wert muss Gleitpunktformat haben.)
L DBD4	//Kopiere den Wert aus AKKU 1 in AKKU 2. Lade den Wert aus Datendoppelwort DBD4 in AKKU 1. (Dieser Wert muss Gleitpunktformat haben.)
+R	//Addiere die Inhalte von AKKU 1 und AKKU 2 als Gleitpunktzahlen (32 Bit, IEEE-FP) und speichere das Ergebnis in AKKU 1.
L DBD8	//Kopiere den Wert aus AKKU 1 in AKKU 2. Lade den Wert aus Datendoppelwort DBD8 in AKKU 1.
ENT	//Kopiere den Inhalt von AKKU 3 in AKKU 4. Kopiere den Inhalt von AKKU 2 (Zwischenergebnis) in AKKU 3.
L DBD12	//Lade den Wert aus Datendoppelwort DBD12 in AKKU 1.
-R	//Subtrahiere den Inhalt von AKKU 1 vom Inhalt von AKKU 2 und speichere das Ergebnis in AKKU 1. Kopiere den Inhalt von AKKU 3 in AKKU 2 und den Inhalt von AKKU 4 in AKKU 3.
/R	//Dividiere den Inhalt von AKKU 2 (DBD0 + DBD4) durch den Inhalt von AKKU 1 (DBD8 - DBD12) und speichere das Ergebnis in AKKU 1.
T DBD16	//Transferiere das Ergebnis (AKKU 1) in das Datendoppelwort DBD16

14.8 LEAVE Leave AKKU-Stack

Format

LEAVE

Beschreibung

LEAVE (Leave AKKU-Stack) kopiert den Inhalt von AKKU 3 in AKKU 2 und den Inhalt von AKKU 4 in AKKU 3. Wenn Sie die Operation **LEAVE** direkt vor einer Schiebe- oder Rotieroperation programmieren, die Akkumulatoren verknüpft, dann funktioniert die Operation **LEAVE** wie eine arithmetische Operation. Die Inhalte von AKKU 1 und AKKU 4 bleiben unverändert.

14.9 INC Inkrementiere AKKU 1-L-L

Format

INC <Ganzzahl, 8 Bit>

Operand	Datentyp	Beschreibung
<Ganzzahl, 8 Bit>	Konstante (Ganzzahl, 8 Bit)	Konstante, die zu AKKU1-L-L addiert wird; Bereich von 0 bis 255

Beschreibung

INC <Ganzzahl, 8 Bit> (Inkrementiere AKKU1-L-L) addiert die Ganzzahl (8 Bit) zum Inhalt von AKKU1-L-L und speichert das Ergebnis in AKKU1-L-L. AKKU1-L-H, AKKU1-H und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Hinweis

Diese Operation eignet sich nicht für arithmetische Operationen (16 oder 32 Bit), da vom niederwertigen Byte des niederwertigen Worts von AKKU 1 nichts in das höherwertige Byte des niederwertigen Worts von AKKU 1 übertragen wird. Verwenden Sie für arithmetische Operationen die Operation +I bzw +D.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MB22	//Lade den Wert von MB22.
INC 1	//Inkrementiere AKKU 1 (MB 22) um 1, speichere das Ergebnis in AKKU1-L-L.
T MB22	//Transferiere den Inhalt von AKKU1-L-L (Ergebnis) zurück nach MB22.

14.10 DEC Dekrementiere AKKU 1-L-L

Format

DEC <Ganzzahl, 8 Bit>

Operand	Datentyp	Beschreibung
<Ganzzahl, 8 Bit>	Konstante (Ganzzahl, 8 Bit)	Konstante, die von AKKU1-L-L subtrahiert wird; Bereich von 0 bis 255

Beschreibung

DEC <Ganzzahl, 8 Bit> (Dekrementiere AKKU1-L-L) subtrahiert die Ganzzahl (8 Bit) vom Inhalt von AKKU1-L-L und speichert das Ergebnis in AKKU1-L-L. AKKU1-L-H, AKKU1-H und AKKU 2 werden nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

Hinweis

Diese Operation eignet sich nicht für arithmetische Operationen (16 oder 32 Bit), da vom niederwertigen Byte des niederwertigen Worts von AKKU 1 nichts in das höherwertige Byte des niederwertigen Worts von AKKU 1 übertragen wird. Verwenden Sie für arithmetische Operationen die Operation +I bzw +D.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel

AWL	Erläuterung
L MB250	//Lade den Wert von MB250.
DEC 1	//Dekrementiere AKKU1-L-L um 1, speichere das Ergebnis in AKKU1-L-L.
T MB250	//Transferiere den Inhalt von AKKU1-L- L (Ergebnis) zurück nach MB250.

14.11 +AR1 Addiere AKKU 1 zum Adreßregister 1

Formate

+AR1
+AR1 <P#Byte.Bit>

Operand	Datentyp	Beschreibung
<P#Byte.Bit>	Pointerkonstante	Adresse, die zu AR1 addiert wird.

Beschreibung

+AR1 (Addiere zu AR1) addiert einen Versatz, der entweder in der Anweisung oder in AKKU1-L angegeben wird, zum Inhalt von AR1. Die Ganzzahl (16 Bit) wird zunächst vorzeichenrichtig auf 24 Bit erweitert und danach zu den niederwertigsten 24 Bit von AR1 (Teil der relativen Adresse in AR1) addiert. Der Teil der Bereichskennung in AR1 (Bits 24, 25 und 26) wird nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

+AR1: Die Ganzzahl (16 Bit), die zum Inhalt von AR1 addiert werden soll, wird durch den Wert in AKKU1-L angegeben. Zulässig sind Werte von -32768 bis +32767.

+AR1 <P#Byte.Bit>: Der Versatz, der addiert werden soll, wird durch den Operanden <P#Byte.Bit> angegeben.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel 1

AWL	Erläuterung
L +300	//Lade den Wert in AKKU1-L.
+AR1	//Addiere AKKU1-L (Ganzzahl, 16 Bit) zu AR 1.

Beispiel 2

AWL	Erläuterung
+AR1 P#300.0	//Addiere den Versatz 300.0 zu AR 1.

14.12 +AR2 Addiere AKKU 1 zum Adreßregister 2

Formate

+AR2
+AR2 <P#Byte.Bit>

Operand	Datentyp	Beschreibung
<P#Byte.Bit>	Pointerkonstante	Adresse, die zu AR2 addiert wird.

Beschreibung

+AR2 (Addiere zu AR2) addiert einen Versatz, der entweder in der Anweisung oder in AKKU1-L angegeben wird, zum Inhalt von AR2. Die Ganzzahl (16 Bit) wird zunächst vorzeichenrichtig auf 24 Bit erweitert und danach zu den niederwertigsten 24 Bit von AR 2 (Teil der relativen Adresse in AR2) addiert. Der Teil der Bereichskennung in AR2 (Bits 24, 25 und 26) wird nicht verändert. Die Operation wird ausgeführt, ohne die Statusbits zu berücksichtigen oder zu beeinflussen.

+AR2: Die Ganzzahl (16 Bit), die zum Inhalt von AR2 addiert werden soll, wird durch den Wert in AKKU1-L angegeben. Zulässig sind Werte von -32768 bis +32767.

+AR2 <P#Byte.Bit>: Der Versatz, der addiert werden soll, wird durch den Operanden <P#Byte.Bit> angegeben.

Hinweis

Das Adreßregister AR2 wird bei der Bearbeitung von Multiinstanzen verwendet. Wenn Sie den Befehl **+AR2** programmieren, so sollten Sie zuvor den Inhalt des AR2 "retten" und später wieder zurückladen.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

Beispiel 1

AWL	Erläuterung	
L	+300	//Lade den Wert in AKKU1-L.
+AR2		//Addiere AKKU1-L (Ganzzahl, 16-Bit) zu AR 2.

Beispiel 2

AWL	Erläuterung	
+AR2	P#300.0	//Addiere den Versatz 300.0 zu AR 2.

14.13 BLD Bildbefehl (Nulloperation)

Format

BLD <Zahl>

Operand	Beschreibung
<Zahl>	Kennummer der Operation BLD; Bereich von 0 bis 255

Beschreibung

BLD <Zahl> (Bildbefehl; Nulloperation) führt keine Funktion aus und beeinflusst die Statusbits nicht. Die Operation dient dem Programmiergerät (PG) zum grafischen Bildaufbau. Sie wird automatisch erzeugt, wenn ein KOP- oder FUP-Programm in AWL angezeigt wird. Der Operand <Zahl> ist die Kennummer der Operation **BLD** und wird vom Programmiergerät erzeugt.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

14.14 NOP 0 Nulloperation

Format

NOP 0

Beschreibung

NOP 0 (Operation NOP mit dem Operand "0") führt keine Funktion aus und beeinflusst die Statusbits nicht. Der Operationscode enthält ein Bitmuster mit 16 Nullen. Die Operation ist nur für das Programmiergerät (PG) wichtig, wenn ein Programm angezeigt wird.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

14.15 NOP 1 Nulloperation

Format

NOP 1

Beschreibung

NOP 1 (Operation NOP mit dem Operand "1") führt keine Funktion aus und beeinflusst die Statusbits nicht. Der Operationscode enthält ein Bitmuster mit 16 Einsen. Die Operation ist nur für das Programmiergerät (PG) wichtig, wenn ein Programm angezeigt wird.

Statuswort

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
schreibt:	-	-	-	-	-	-	-	-	-

A AWL-Operationen Übersicht

A.1 AWL-Operationen sortiert nach deutscher Mnemonik (SIMATIC)

Deutsche Mnemonic	Englische Mnemonik	Operation/ Funktion	Beschreibung
=	=	Bitverknüpfung	Zuweisung
))	Bitverknüpfung	Verzweigung schließen
*D	*D	Festpunkt-Funktion	Multipliziere AKKU 1 und 2 als Ganzzahl (32 Bit)
*I	*I	Festpunkt-Funktion	Multipliziere AKKU 1 und 2 als Ganzzahl (16 Bit)
*R	*R	Gleitpunkt-Funktion	Multipliziere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
/D	/D	Festpunkt-Funktion	Dividiere AKKU 2 durch 1 als Ganzzahl (32 Bit)
/I	/I	Festpunkt-Funktion	Dividiere AKKU 2 durch 1 als Ganzzahl (16 Bit)
/R	/R	Gleitpunkt-Funktion	Dividiere AKKU 2 durch 1 als Gleitpunktzahl (32 Bit)
? D	? D	Vergleicher	Vergleiche Ganzzahlen (32 Bit) ==, <>, >, <, >=, <=
? I	? I	Vergleicher	Vergleiche Ganzzahlen (16 Bit) ==, <>, >, <, >=, <=
? R	? R	Vergleicher	Vergleiche Gleitpunktzahlen (32 Bit) ==, <>, >, <, >=, <=
+	+	Festpunkt-Funktion	Addiere Ganzzahlkonstante (16, 32 Bit)
+AR1	+AR1	Akkumulator	Addiere AKKU 1 zum Adreßregister 1
+AR2	+AR2	Akkumulator	Addiere AKKU 1 zum Adreßregister 2
+D	+D	Festpunkt-Funktion	Addiere AKKU 1 und 2 als Ganzzahl (32 Bit)
+I	+I	Festpunkt-Funktion	Addiere AKKU 1 und 2 als Ganzzahl (16 Bit)
+R	+R	Gleitpunkt-Funktion	Addiere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
-D	-D	Festpunkt-Funktion	Subtrahiere AKKU 1 von 2 als Ganzzahl (32 Bit)
-I	-I	Festpunkt-Funktion	Subtrahiere AKKU 1 von 2 als Ganzzahl (16 Bit)
-R	-R	Gleitpunkt-Funktion	Subtrahiere AKKU 1 von 2 als Gleitpunktzahl (32 Bit)
ABS	ABS	Gleitpunkt-Funktion	Absolutwert einer Gleitpunktzahl (32 Bit, IEEE-FP)
ACOS	ACOS	Gleitpunkt-Funktion	Bilden des Arcuscossinus einer Gleitpunktzahl (32 Bit)
ASIN	ASIN	Gleitpunkt-Funktion	Bilden des Arcussinus einer Gleitpunktzahl (32 Bit)
ATAN	ATAN	Gleitpunkt-Funktion	Bilden des Arcustangens einer Gleitpunktzahl (32 Bit)
AUF	OPN	Datenbaustein	Aufschlage Datenbaustein
BE	BE	Programmsteuerung	Bausteinende
BEA	BEU	Programmsteuerung	Bausteinende absolut
BEB	BEC	Programmsteuerung	Bausteinende bedingt
BLD	BLD	Akkumulator	Bildbefehl (Nulloperation)
BTD	BTD	Umwandler	BCD wandeln in Ganzzahl (32 Bit)

Deutsche Mnemonic	Englische Mnemonic	Operation/ Funktion	Beschreibung
BTI	BTI	Umwandler	BCD wandeln in Ganzzahl (16 Bit)
CALL	CALL	Programmsteuerung	Baustein aus einer Bibliothek aufrufen
CALL	CALL	Programmsteuerung	Bausteinaufruf
CALL	CALL	Programmsteuerung	Multiinstanz aufrufen
CC	CC	Programmsteuerung	Bedingter Bausteinaufruf
CLR	CLR	Bitverknüpfung	Rücksetze VKE (=0)
COS	COS	Gleitpunkt-Funktion	Bilden des Cosinus eines Winkels als Gleitpunktzahlen (32 Bit)
DEC	DEC	Akkumulator	Dekrementiere AKKU 1
DTB	DTB	Umwandler	Ganzzahl (32 Bit) in BCD wandeln
DTR	DTR	Umwandler	Ganzzahl (32 Bit) wandeln in Gleitpunktzahl (32 Bit, IEEE-FP)
ENT	ENT	Akkumulator	Enter AKKU-Stack
EXP	EXP	Gleitpunkt-Funktion	Bilden des Exponentialwerts einer Gleitpunktzahl (32 Bit)
FN	FN	Bitverknüpfung	Flanke Negativ
FP	FP	Bitverknüpfung	Flanke Positiv
FR	FR	Zeiten	Freigabe Timer
FR	FR	Zähler	Freigabe Zähler (Frei, FR Z 0 zu Z 255)
INC	INC	Akkumulator	Inkrementiere AKKU 1
INVD	INVD	Umwandler	1-Komplement Ganzzahl (32 Bit)
INVI	INVI	Umwandler	1-Komplement Ganzzahl (16 Bit)
ITB	ITB	Umwandler	Ganzzahl (16 Bit) wandeln in BCD
ITD	ITD	Umwandler	Ganzzahl (16 Bit) wandeln in Ganzzahl (32 Bit)
L DBLG	L DBLG	Datenbaustein	L DBLG Lade Länge Global-DB in AKKU 1
L DBNO	L DBNO	Datenbaustein	L DBNO Lade Nummer Global-DB in AKKU 1
L DILG	L DILG	Datenbaustein	L DILG Lade Länge Instanz-DB in AKKU 1
L DINO	L DINO	Datenbaustein	L DINO Lade Nummer Instanz-DB in AKKU 1
L	L	Laden/Transferieren	Lade
L	L	Zähler	Lade aktuellen Zählerwert als Ganzzahl in AKKU 1 (der aktuelle Zählerwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: L Z 15)
L	L	Zeiten	Lade aktuellen Zeitwert als Ganzzahl in AKKU 1 (der aktuelle Zeitwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: L T 32)
L STW	L STW	Laden/Transferieren	Lade Statuswort in AKKU 1
LAR1	LAR1	Laden/Transferieren	Lade Adreßregister 1 mit Inhalt von Adresregister 2
LAR1	LAR1	Laden/Transferieren	Lade Adreßregister 1 mit Inhalt von AKKU 1
LAR1	LAR1	Laden/Transferieren	Lade Adreßregister 1 mit Pointer (32 Bit-Format)
LAR2	LAR2	Laden/Transferieren	Lade Adreßregister 2 mit Ganzzahl (32 Bit)
LAR2	LAR2	Laden/Transferieren	Lade Adreßregister 2 mit Inhalt von AKKU 1
LC	LC	Zähler	Lade aktuellen Zählerwert als BCD in AKKU 1 (der aktuelle Zählerwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: LC Z 15)

Deutsche Mnemonic	Englische Mnemonic	Operation/ Funktion	Beschreibung
LC	LC	Zeiten	Lade aktuellen Zeitwert als BCD in AKKU 1 (der aktuelle Zeitwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: LC T 32)
LEAVE	LEAVE	Akkumulator	Leave AKKU-Stack
LN	LN	Gleitpunkt-Funktion	Bilden des natürlichen Logarithmus einer Gleitpunktzahl (32 Bit)
LOOP	LOOP	Sprünge	Programmschleife
MCR(MCR(Programmsteuerung	Sichere VKE im MCR-Stack, Beginn MCR-Bereich
)MCR)MCR	Programmsteuerung	Beende MCR-Bereich
MCRA	MCRA	Programmsteuerung	Aktiviere MCR-Bereich
MCRD	MCRD	Programmsteuerung	Deaktiviere MCR-Bereich
MOD	MOD	Festpunkt-Funktion	Divisionsrest Ganzzahl (32 Bit)
NEGD	NEGD	Umwandler	2-Komplement Ganzzahl (32 Bit)
NEGI	NEGI	Umwandler	2-Komplement Ganzzahl (16 Bit)
NEGR	NEGR	Umwandler	Negiere Gleitpunktzahl (32 Bit, IEEE-FP)
NOP 0	NOP 0	Akkumulator	Nulloperation 0
NOP 1	NOP 1	Akkumulator	Nulloperation 1
NOT	NOT	Bitverknüpfung	Negiere VKE
O	O	Bitverknüpfung	ODER
O	O	Bitverknüpfung	Und vor Oder
O(O(Bitverknüpfung	Oder mit Verzweigung
OD	OD	Wortverknüpfung	ODER-Doppelwort (32 Bit)
ON	ON	Bitverknüpfung	Oder Nicht
ON(ON(Bitverknüpfung	Oder Nicht mit Verzweigung
OW	OW	Wortverknüpfung	ODER-Wort (16 Bit)
POP	POP	Akkumulator	POP CPU mit vier Akkus
POP	POP	Akkumulator	POP CPU mit zwei Akkus
PUSH	PUSH	Akkumulator	PUSH CPU mit vier Akkus
PUSH	PUSH	Akkumulator	PUSH CPU mit zwei Akkus
R	R	Bitverknüpfung	Rücksetze
R	R	Zeiten	Rücksetze Timer (der aktuelle Timer kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: R T 32)
R	R	Zähler	Rücksetze Zähler (der aktuelle Zähler kann eine Zahl von 0 bis 255 sein, zum Beispiel: R Z 15)
RLD	RLD	Schieben/Rotieren	Rotiere links Doppelwort (32 Bit)
RLDA	RLDA	Schieben/Rotieren	Rotiere Akku 1 links über A1-Anzeige (32 Bit)
RND	RND	Umwandler	Runden einer Gleitpunktzahl zur Ganzzahl
RND-	RND-	Umwandler	Runden einer Gleitpunktzahl zur nächstniederen Ganzzahl
RND+	RND+	Umwandler	Runden einer Gleitpunktzahl zur nächsthöheren Ganzzahl
RRD	RRD	Schieben/Rotieren	Rotiere rechts Doppelwort (32 Bit)
RRDA	RRDA	Schieben/Rotieren	Rotiere Akku 1 links über A1-Anzeige (32 Bit)

Deutsche Mnemonic	Englische Mnemonik	Operation/ Funktion	Beschreibung
S	S	Bitverknüpfung	Setze
S	S	Zähler	Setze Zählerstartwert (der aktuelle Zähler kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: S Z 15)
SA	SF	Zeiten	Zeit als Ausschaltverzögerung
SAVE	SAVE	Bitverknüpfung	Sichere VKE im BIE-Bit
SE	SD	Zeiten	Zeit als Einschaltverzögerung
SET	SET	Bitverknüpfung	Setze
SI	SP	Zeiten	Zeit als Impuls
SIN	SIN	Gleitpunkt-Funktion	Bilden des Sinus eines Winkels als Gleitpunktzahlen (32 Bit)
SLD	SLD	Schieben/Rotieren	Schiebe links Doppelwort (32 Bit)
SLW	SLW	Schieben/Rotieren	Schiebe links Wort (16 Bit)
SPA	JU	Sprünge	Springe absolut
SPB	JC	Sprünge	Springe, wenn VKE = 1
SPBB	JCB	Sprünge	Springe, wenn VKE = 1 und rette VKE ins BIE
SPBI	JB I	Sprünge	Springe, wenn BIE = 1
SPBIN	JNBI	Sprünge	Springe, wenn BIE = 0
SPBN	JCN	Sprünge	Springe, wenn VKE = 0
SPBNB	JNB	Sprünge	Springe, wenn VKE = 0 und rette VKE ins BIE
SPL	JL	Sprünge	Sprungleiste
SPM	JM	Sprünge	Springe, wenn Ergebnis < 0
SPMZ	JMZ	Sprünge	Springe, wenn Ergebnis <= 0
SPN	JN	Sprünge	Springe, wenn Ergebnis <> 0
SPO	JO	Sprünge	Springe, wenn OV = 1
SPP	JP	Sprünge	Springe, wenn Ergebnis > 0
SPPZ	JPZ	Sprünge	Springe, wenn Ergebnis >= 0
SPS	JOS	Sprünge	Springe, wenn OS = 1
SPU	JUO	Sprünge	Springe, wenn Ergebnis ungültig
SPZ	JZ	Sprünge	Springe, wenn Ergebnis = 0
SQR	SQR	Gleitpunkt-Funktion	Bilden des Quadrats einer Gleitpunktzahl (32 Bit)
SQRT	SQRT	Gleitpunkt-Funktion	Bilden der Quadratwurzel einer Gleitpunktzahl (32 Bit)
SRD	SRD	Schieben/Rotieren	Schiebe rechts Doppelwort (32 Bit)
SRW	SRW	Schieben/Rotieren	Schiebe rechts Wort (16 Bit)
SS	SS	Zeiten	Zeit als speichernde Einschaltverzögerung
SSD	SSD	Schieben/Rotieren	Schiebe Vorzeichen rechts Ganzzahl (32 Bit)
SSI	SSI	Schieben/Rotieren	Schiebe Vorzeichen rechts Ganzzahl (16 Bit)
SV	SE	Zeiten	Zeit als verlängerter Impuls
T	T	Laden/Transferieren	Transferiere
T STW	T STW	Laden/Transferieren	T STW Transferiere AKKU 1 in Statuswort
TAD	CAD	Umwandler	Tausche Reihenfolge der Bytes im AKKU 1 (32 Bit)
TAK	TAK	Akkumulator	Tausche AKKU 1 mit AKKU 2
TAN	TAN	Gleitpunkt-Funktion	Bilden des Tangens eines Winkels als Gleitpunktzahlen (32 Bit)

Deutsche Mnemonic	Englische Mnemonik	Operation/ Funktion	Beschreibung
TAR	CAR	Laden/Transferieren	Tausche Adreßregister 1 mit 2
TAR1	TAR1	Laden/Transferieren	Transferiere Adreßregister 1 in Adreßregister 2
TAR1	TAR1	Laden/Transferieren	Transferiere Adreßregister 1 in AKKU 1
TAR1	TAR1	Laden/Transferieren	Transferiere Adreßregister 1 nach Zieladresse (32-Bit-Pointer)
TAR2	TAR2	Laden/Transferieren	Transferiere Adreßregister 2 in AKKU 1
TAR2	TAR2	Laden/Transferieren	Transferiere Adreßregister 2 nach Zieladresse (32-Bit-Pointer)
TAW	CAW	Umwandler	Tausche Reihenfolge der Bytes im AKKU 1-L (16 Bit)
TDB	CDB	Datenbaustein	Tausche Global-DB und Instanz-DB
TRUNC	TRUNC	Umwandler	Runden einer Gleitpunktzahl durch Abschneiden
U	A	Bitverknüpfung	Und
U(A(Bitverknüpfung	Und mit Verzweigung
UC	UC	Programmsteuerung	Unbedingter Bausteinaufruf
UD	AD	Wortverknüpfung	UND-Doppelwort (32 Bit)
UN	AN	Bitverknüpfung	Und Nicht
UN(AN(Bitverknüpfung	Und Nicht mit Verzweigung
UW	AW	Wortverknüpfung	UND-Wort (16 Bit)
X	X	Bitverknüpfung	Exklusiv Oder
X(X(Bitverknüpfung	Exklusiv Oder mit Verzweigung
XN	XN	Bitverknüpfung	Exklusiv Oder Nicht
XN(XN(Bitverknüpfung	Exklusiv Oder Nicht mit Verzweigung
XOD	XOD	Wortverknüpfung	EXKLUSIV-ODER-Doppelwort (32 Bit)
XOW	XOW	Wortverknüpfung	EXKLUSIV-ODER-Wort (16 Bit)
ZR	CD	Zähler	Zählen rückwärts
ZV	CU	Zähler	Zählen vorwärts

A.2 AWL-Operationen sortiert nach englischer Mnemonik (International)

Englische Mnemonik	Deutsche-Mnemonik	Operation/ Funktion	Beschreibung
=	=	Bitverknüpfung	Zuweisung
))	Bitverknüpfung	Verzweigung schließen
*D	*D	Festpunkt-Funktion	Multipliziere AKKU 1 und 2 als Ganzzahl (32 Bit)
*I	*I	Festpunkt-Funktion	Multipliziere AKKU 1 und 2 als Ganzzahl (16 Bit)
*R	*R	Gleitpunkt-Funktion	Multipliziere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
/D	/D	Festpunkt-Funktion	Dividiere AKKU 2 durch 1 als Ganzzahl (32 Bit)
/I	/I	Festpunkt-Funktion	Dividiere AKKU 2 durch 1 als Ganzzahl (16 Bit)
/R	/R	Gleitpunkt-Funktion	Dividiere AKKU 2 durch 1 als Gleitpunktzahl (32 Bit)
? D	? D	Vergleicher	Vergleiche Ganzzahlen (32 Bit) ==, <>, >, <, >=, <=
? I	? I	Vergleicher	Vergleiche Ganzzahlen (16 Bit) ==, <>, >, <, >=, <=
? R	? R	Vergleicher	Vergleiche Gleitpunktzahlen (32 Bit) ==, <>, >, <, >=, <=
+	+	Festpunkt-Funktion	Addiere Ganzzahlkonstante (16, 32 Bit)
+AR1	+AR1	Akkumulator	Addiere AKKU 1 zum Adreßregister 1
+AR2	+AR2	Akkumulator	Addiere AKKU 1 zum Adreßregister 2
+D	+D	Festpunkt-Funktion	Addiere AKKU 1 und 2 als Ganzzahl (32 Bit)
+I	+I	Festpunkt-Funktion	Addiere AKKU 1 und 2 als Ganzzahl (16 Bit)
+R	+R	Gleitpunkt-Funktion	Addiere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)
A	U	Bitverknüpfung	Und
A(U(Bitverknüpfung	Und mit Verzweigung
ABS	ABS	Gleitpunkt-Funktion	Absolutwert einer Gleitpunktzahl (32 Bit, IEEE-FP)
ACOS	ACOS	Gleitpunkt-Funktion	Bilden des Arcuscossinus einer Gleitpunktzahl (32 Bit)
AD	UD	Wortverknüpfung	UND-Doppelwort (32 Bit)
AN	UN	Bitverknüpfung	Und Nicht
AN(UN(Bitverknüpfung	Und Nicht mit Verzweigung
ASIN	ASIN	Gleitpunkt-Funktion	Bilden des Arcussinus einer Gleitpunktzahl (32 Bit)
ATAN	ATAN	Gleitpunkt-Funktion	Bilden des Arcustangens einer Gleitpunktzahl (32 Bit)
AW	UW	Wortverknüpfung	UND-Wort (16 Bit)
BE	BE	Programmsteuerung	Bausteinende
BEC	BEB	Programmsteuerung	Bausteinende bedingt
BEU	BEA	Programmsteuerung	Bausteinende absolut
BLD	BLD	Akkumulator	Bildbefehl (Nulloperation)
BTD	BTD	Umwandler	BCD wandeln in Ganzzahl (32 Bit)
BTI	BTI	Umwandler	BCD wandeln in Ganzzahl (16 Bit)
CAD	TAD	Umwandler	Tausche Reihenfolge der Bytes im AKKU 1 (32 Bit)
CALL	CALL	Programmsteuerung	Baustein aus einer Bibliothek aufrufen
CALL	CALL	Programmsteuerung	Bausteinaufruf
CALL	CALL	Programmsteuerung	Multiinstanz aufrufen
CAR	TAR	Laden/Transferieren	Tausche Adreßregister 1 mit 2
CAW	TAW	Umwandler	Tausche Reihenfolge der Bytes im AKKU 1-L (16 Bit)

Englische Mnemonik	Deutsche-Mnemonik	Operation/ Funktion	Beschreibung
CC	CC	Programmsteuerung	Bedingter Bausteinaufruf
CD	ZR	Zähler	Zählen rückwärts
CDB	TDB	Datenbaustein	Tausche Global-DB und Instanz-DB
CLR	CLR	Bitverknüpfung	Rücksetze VKE (=0)
COS	COS	Gleitpunkt-Funktion	Bilden des Cosinus eines Winkels als Gleitpunktzahlen (32 Bit)
CU	ZV	Zähler	Zählen vorwärts
-D	-D	Festpunkt-Funktion	Subtrahiere AKKU 1 von 2 als Ganzzahl (32 Bit)
DEC	DEC	Akkumulator	Dekrementiere AKKU 1
DTB	DTB	Umwandler	Ganzzahl (32 Bit) in BCD wandeln
DTR	DTR	Umwandler	Ganzzahl (32 Bit) wandeln in Gleitpunktzahl (32 Bit, IEEE-FP)
ENT	ENT	Akkumulator	Enter AKKU-Stack
EXP	EXP	Gleitpunkt-Funktion	Bilden des Exponentialwerts einer Gleitpunktzahl (32 Bit)
FN	FN	Bitverknüpfung	Flanke Negativ
FP	FP	Bitverknüpfung	Flanke Positiv
FR	FR	Zeiten	Freigabe Timer
FR	FR	Zähler	Freigabe Zähler (Frei, FR Z 0 zu Z 255)
-I	-I	Festpunkt-Funktion	Subtrahiere AKKU 1 von 2 als Ganzzahl (16 Bit)
INC	INC	Akkumulator	Inkrementiere AKKU 1
INVD	INVD	Umwandler	1-Komplement Ganzzahl (32 Bit)
INVI	INVI	Umwandler	1-Komplement Ganzzahl (16 Bit)
ITB	ITB	Umwandler	Ganzzahl (16 Bit) wandeln in BCD
ITD	ITD	Umwandler	Ganzzahl (16 Bit) wandeln in Ganzzahl (32 Bit)
JBI	SPBI	Sprünge	Springe, wenn BIE = 1
JC	SPB	Sprünge	Springe, wenn VKE = 1
JCB	SPBB	Sprünge	Springe, wenn VKE = 1 und rette VKE ins BIE
JCN	SPBN	Sprünge	Springe, wenn VKE = 0
JL	SPL	Sprünge	Sprungleiste
JM	SPM	Sprünge	Springe, wenn Ergebnis < 0
JMZ	SPMZ	Sprünge	Springe, wenn Ergebnis <= 0
JN	SPN	Sprünge	Springe, wenn Ergebnis <> 0
JNB	SPBNB	Sprünge	Springe, wenn VKE = 0 und rette VKE ins BIE
JNBI	SPBIN	Sprünge	Springe, wenn BIE = 0
JO	SPO	Sprünge	Springe, wenn OV = 1
JOS	SPS	Sprünge	Springe, wenn OS = 1
JP	SPP	Sprünge	Springe, wenn Ergebnis > 0
JPZ	SPPZ	Sprünge	Springe, wenn Ergebnis >= 0
JU	SPA	Sprünge	Springe absolut
JUO	SPU	Sprünge	Springe, wenn Ergebnis ungültig
JZ	SPZ	Sprünge	Springe, wenn Ergebnis = 0
L DBLG	L DBLG	Datenbaustein	Lade Länge Global-DB in AKKU 1

Englische Mnemonik	Deutsche-Mnemonik	Operation/ Funktion	Beschreibung
L DBNO	L DBNO	Datenbaustein	Lade Nummer Global-DB in AKKU 1
L DILG	L DILG	Datenbaustein	Lade Länge Instanz-DB in AKKU 1
L DINO	L DINO	Datenbaustein	Lade Nummer Instanz-DB in AKKU 1
L	L	Laden/Transferieren	Lade
L	L	Zähler	Lade aktuellen Zählerwert als Ganzzahl in AKKU 1 (der aktuelle Zählerwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: L Z 15)
L	L	Zeiten	Lade aktuellen Zeitwert als Ganzzahl in AKKU 1 (der aktuelle Zeitwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: L T 32)
L STW	L STW	Laden/Transferieren	Lade Statuswort in AKKU 1
LAR1	LAR1	Laden/Transferieren	Lade Adreßregister 1 mit Inhalt von Adressregister 2
LAR1	LAR1	Laden/Transferieren	Lade Adreßregister 1 mit Inhalt von AKKU 1
LAR1	LAR1	Laden/Transferieren	Lade Adreßregister 1 mit Pointer (32 Bit-Format)
LAR2	LAR2	Laden/Transferieren	Lade Adreßregister 2 mit Ganzzahl (32 Bit)
LAR2	LAR2	Laden/Transferieren	Lade Adreßregister 2 mit Inhalt von AKKU 1
LC	LC	Zähler	Lade aktuellen Zählerwert als BCD in AKKU 1 (der aktuelle Zählerwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: LC Z 15)
LC	LC	Zeiten	Lade aktuellen Zeitwert als BCD in AKKU 1 (der aktuelle Zeitwert kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: LC T 32)
LEAVE	LEAVE	Akkumulator	Leave AKKU-Stack
LN	LN	Gleitpunkt-Funktion	Bilden des natürlichen Logarithmus einer Gleitpunktzahl (32 Bit)
LOOP	LOOP	Sprünge	Programmschleife
MCR(MCR(Programmsteuerung	Sichere VKE im MCR-Stack, Beginn MCR-Bereich
)MCR)MCR	Programmsteuerung	Beende MCR-Bereich
MCRA	MCRA	Programmsteuerung	Aktiviere MCR-Bereich
MCRD	MCRD	Programmsteuerung	Deaktiviere MCR-Bereich
MOD	MOD	Festpunkt-Funktion	Divisionsrest Ganzzahl (32 Bit)
NEGD	NEGD	Umwandler	2-Komplement Ganzzahl (32 Bit)
NEGI	NEGI	Umwandler	2-Komplement Ganzzahl (16 Bit)
NEGR	NEGR	Umwandler	Negiere Gleitpunktzahl (32 Bit, IEEE-FP)
NOP 0	NOP 0	Akkumulator	Nulloperation 0
NOP 1	NOP 1	Akkumulator	Nulloperation 1
NOT	NOT	Bitverknüpfung	Negiere VKE
O	O	Bitverknüpfung	ODER
O	O	Bitverknüpfung	Und vor Oder
O(O(Bitverknüpfung	Oder mit Verzweigung
OD	OD	Wortverknüpfung	ODER-Doppelwort (32 Bit)
ON	ON	Bitverknüpfung	Oder Nicht

Englische Mnemonik	Deutsche-Mnemonik	Operation/ Funktion	Beschreibung
ON(ON(Bitverknüpfung	Oder Nicht mit Verzweigung
OPN	AUF	Datenbaustein	Aufschlage Datenbaustein
OW	OW	Wortverknüpfung	ODER-Wort (16 Bit)
POP	POP	Akkumulator	POP CPU mit vier Akkus
POP	POP	Akkumulator	POP CPU mit zwei Akkus
PUSH	PUSH	Akkumulator	PUSH CPU mit vier Akkus
PUSH	PUSH	Akkumulator	PUSH CPU mit zwei Akkus
R	R	Bitverknüpfung	Rücksetze
R	R	Zeiten	Rücksetze Timer (der aktuelle Timer kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: R T 32)
R	R	Zähler	Rücksetze Zähler (der aktuelle Zähler kann eine Zahl von 0 bis 255 sein, zum Beispiel: R Z 15)
-R	-R	Gleitpunkt-Funktion	Subtrahiere AKKU 1 von 2 als Gleitpunktzahl (32 Bit)
RLD	RLD	Schieben/Rotieren	Rotiere links Doppelwort (32 Bit)
RLDA	RLDA	Schieben/Rotieren	Rotiere Akku 1 links über A1-Anzeige (32 Bit)
RND	RND	Umwandler	Runden einer Gleitpunktzahl zur Ganzzahl
RND-	RND-	Umwandler	Runden einer Gleitpunktzahl zur nächstniederen Ganzzahl
RND+	RND+	Umwandler	Runden einer Gleitpunktzahl zur nächsthöheren Ganzzahl
RRD	RRD	Schieben/Rotieren	Rotiere rechts Doppelwort (32 Bit)
RRDA	RRDA	Schieben/Rotieren	Rotiere Akku 1 links über A1-Anzeige (32 Bit)
S	S	Bitverknüpfung	Setze
S	S	Zähler	Setze Zählerstartwert (der aktuelle Zähler kann eine Zahl im Bereich von 0 bis 255 sein, zum Beispiel: S Z 15)
SAVE	SAVE	Bitverknüpfung	Sichere VKE im BIE-Bit
SD	SE	Zeiten	Zeit als Einschaltverzögerung
SE	SV	Zeiten	Zeit als verlängerter Impuls
SET	SET	Bitverknüpfung	Setze
SF	SA	Zeiten	Zeit als Ausschaltverzögerung
SIN	SIN	Gleitpunkt-Funktion	Bilden des Sinus eines Winkels als Gleitpunktzahlen (32 Bit)
SLD	SLD	Schieben/Rotieren	Schiebe links Doppelwort (32 Bit)
SLW	SLW	Schieben/Rotieren	Schiebe links Wort (16 Bit)
SP	SI	Zeiten	Zeit als Impuls
SQR	SQR	Gleitpunkt-Funktion	Bilden des Quadrats einer Gleitpunktzahl (32 Bit)
SQRT	SQRT	Gleitpunkt-Funktion	Bilden der Quadratwurzel einer Gleitpunktzahl (32 Bit)
SRD	SRD	Schieben/Rotieren	Schiebe rechts Doppelwort (32 Bit)
SRW	SRW	Schieben/Rotieren	Schiebe rechts Wort (16 Bit)
SS	SS	Zeiten	Zeit als speichernde Einschaltverzögerung
SSD	SSD	Schieben/Rotieren	Schiebe Vorzeichen rechts Ganzzahl (32 Bit)
SSI	SSI	Schieben/Rotieren	Schiebe Vorzeichen rechts Ganzzahl (16 Bit)
T	T	Laden/Transferieren	T STW Transferiere AKKU 1 in Statuswort

Englische Mnemonik	Deutsche-Mnemonik	Operation/ Funktion	Beschreibung
T	T	Laden/Transferieren	Transferiere
T STW	T STW	Transferiere AKKU 1 in Statuswort	T STW Transferiere AKKU 1 in Statuswort
TAK	TAK	Akkumulator	Tausche AKKU 1 mit AKKU 2
TAN	TAN	Gleitpunkt-Funktion	Bilden des Tangens eines Winkels als Gleitpunktzahlen (32 Bit)
TAR1	TAR1	Laden/Transferieren	Transferiere Adreßregister 1 in Adreßregister 2
TAR1	TAR1	Laden/Transferieren	Transferiere Adreßregister 1 in AKKU 1
TAR1	TAR1	Laden/Transferieren	Transferiere Adreßregister 1 nach Zieladresse (32-Bit-Pointer)
TAR2	TAR2	Laden/Transferieren	Transferiere Adreßregister 2 in AKKU 1
TAR2	TAR2	Laden/Transferieren	Transferiere Adreßregister 2 nach Zieladresse (32-Bit-Pointer)
TRUNC	TRUNC	Umwandler	Runden einer Gleitpunktzahl durch Abschneiden
UC	UC	Programmsteuerung	Unbedingter Bausteinaufruf
X	X	Bitverknüpfung	Exklusiv Oder
X(X(Bitverknüpfung	Exklusiv Oder mit Verzweigung
XN	XN	Bitverknüpfung	Exklusiv Oder Nicht
XN(XN(Bitverknüpfung	Exklusiv Oder Nicht mit Verzweigung
XOD	XOD	Wortverknüpfung	EXKLUSIV-ODER-Doppelwort (32 Bit)
XOW	XOW	Wortverknüpfung	EXKLUSIV-ODER-Wort (16 Bit)

B Programmierbeispiele

B.1 Programmierbeispiele Übersicht

Praktische Anwendungen

Jede AWL-Operation löst eine bestimmte Funktion aus. Durch Kombination der Operationen in einem Programm können Sie eine breite Palette von Automatisierungsaufgaben ausführen. Hier einige Beispiele für praktische Anwendungen:

- Steuern eines Förderbandes durch Bitverknüpfungsoperationen
- Erfassen der Richtung eines Förderbandes durch Bitverknüpfungsoperationen
- Generieren eines Taktimpulses durch Zeitoperationen
- Überwachen des Lagerbereichs durch Zähl- und Vergleichsoperationen
- Berechnungen mit arithmetischen Operationen für Ganzzahlen
- Einstellen der Zeitdauer für das Beheizen eines Ofens

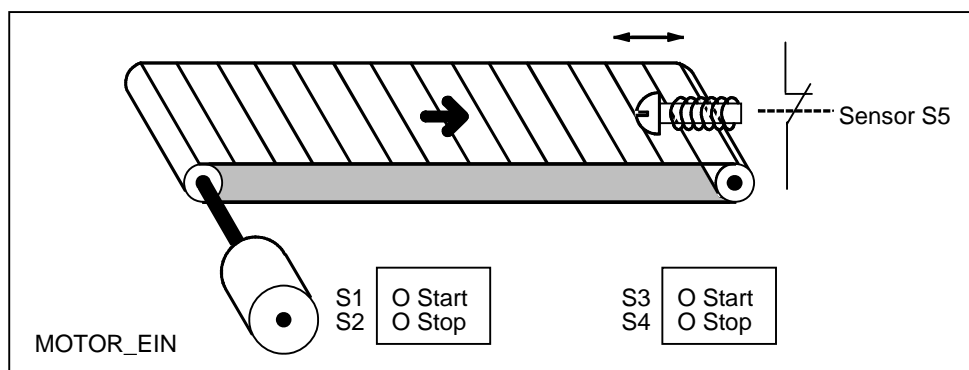
Verwendete Operationen

Mnemonic	Operation	Beschreibung
UW	Wortverknüpfung	16 Bit UND verknüpfen
OW	Wortverknüpfung	16 Bit ODER verknüpfen
ZV, ZR	Zähler	Vorwärtszählen, Rückwärtszählen
S, R	Bitverknüpfung	Ausgang setzen, Ausgang rücksetzen
NOT	Bitverknüpfung	Negiere VKE
FP	Bitverknüpfung	Flanke 0 → 1 abfragen
+I	Festpunkt-Funktion	Ganze Zahlen addieren (16 Bit)
/I	Festpunkt-Funktion	Ganze Zahlen dividieren (16 Bit)
*I	Festpunkt-Funktion	Ganze Zahlen multiplizieren (16 Bit)
>=I, <=I	Vergleicher	Ganze Zahlen vergleichen (16 Bit)
U, UN	Bitverknüpfung	UND, UND NICHT
O, ON	Bitverknüpfung	ODER, ODER NICHT
=	Bitverknüpfung	Zuweisung
INC	Akkumulator	Inkrementiere AKKU 1
BE, BEB	Programmsteuerung	Baustein, Bausteinende bedingt
L, T	Laden/Transferieren	Laden, Transferieren
SV	Zeiten	Zeit als verlängerten Impuls starten

B.2 Bitverknüpfungsoperationen Beispiel

Beispiel 1: Steuern eines Förderbandes

Das folgende Bild zeigt ein Förderband, das elektrisch in Gang gesetzt werden kann. Am Anfang des Bandes befinden sich zwei Druckschalter, S1 für START und S2 für STOP. Am Ende des Bandes befinden sich ebenfalls zwei Druckschalter, S3 für START und S4 für STOP. Das Band kann von beiden Enden aus gestartet oder gestoppt werden. Außerdem stoppt der Sensor S5 das Band, wenn ein Gegenstand auf dem Band dessen Ende erreicht.



Absolute und symbolische Programmierung

Sie können ein Programm zum Steuern des Förderbandes schreiben, indem Sie die verschiedenen Komponenten des Förderbandsystems mit Hilfe von **absoluten Adressen** oder **Symbolen** darstellen.

Die von Ihnen gewählten Symbole setzen Sie in der Symboltabelle mit den absoluten Adressen in Beziehung (siehe Online-Hilfe zu STEP 7).

Systemkomponente	Absolute Adresse	Symbol	Symboltabelle
Startschalter	E 1.1	S1	E 1.1 S1
Stoppschalter	E 1.2	S2	E 1.2 S2
Startschalter	E 1.3	S3	E 1.3 S3
Stoppschalter	E 1.4	S4	E 1.4 S4
Sensor	E 1.5	S5	E 1.5 S5
Motor	A 4.0	MOTOR_EIN	A 4.0 MOTOR_EIN

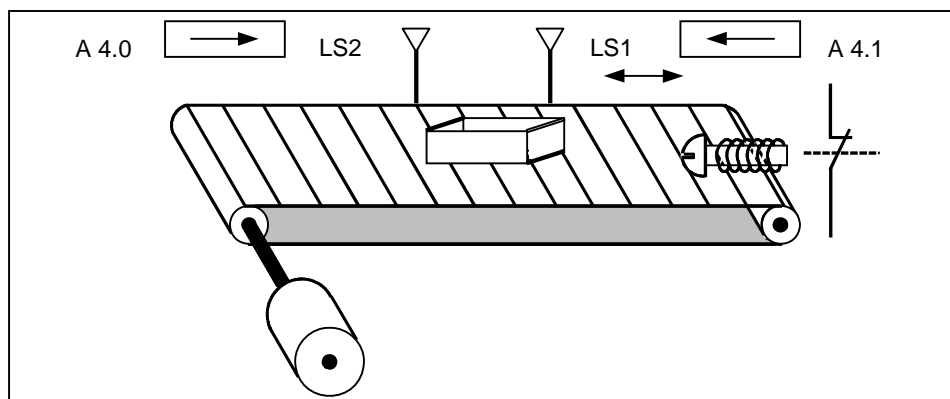
Absolute Programmierung	Symbolische Programmierung
O E 1.1	O S1
O E 1.3	O S3
S A 4.0	S MOTOR_EIN
O E 1.2	O S2
O E 1.4	O S4
ON E 1.5	ON S5
R A 4.0	R MOTOR_EIN

Anweisungsliste zum Steuern des Förderbandes

AWL	Erläuterung
O E 1.1	//Durch Drücken einer der beiden Startschalter wird der Motor eingeschaltet.
O E 1.3	
S A 4.0	
O E 1.2	//Durch Drücken einer der beiden Stoppschalter oder Öffnen eines Öffners am Ende des Bandes wird der Motor ausgeschaltet.
O E 1.4	
ON E 1.5	
R A 4.0	

Beispiel 2: Erfassen der Richtung eines Förderbandes

Das folgende Bild zeigt ein Förderband, das mit zwei Lichtschranken (LS1, LS2) ausgestattet ist. Die Lichtschranken sollen feststellen, in welche Richtung sich ein Paket auf dem Band bewegt.



Absolute und symbolische Programmierung

Sie können ein Programm schreiben, das die Richtungsanzeige für das Förderbandsystem aktiviert, indem Sie die verschiedenen Komponenten des Fördersystems mit Hilfe von **absoluten Adressen** oder **Symbolen** darstellen.

Die von Ihnen gewählten Symbole setzen Sie in der Symboltabelle mit den absoluten Adressen in Beziehung (siehe Online-Hilfe zu STEP 7).

Systemkomponente	Absolute Adresse	Symbol	Symboltabelle
Lichtschanke 1	E 0.0	LS1	E 0.0 LS1
Lichtschanke 2	E 0.1	LS2	E 0.1 LS2
Anzeige für Bewegung nach rechts	A 4.0	RECHTS	A 4.0 RECHTS
Anzeige für Bewegung nach links	A 4.1	LINKS	A 4.1 LINKS
Taktmerker 1	M 0.0	TM1	M 0.0 TM1
Taktmerker 2	M 0.1	TM2	M 0.1 TM2

Absolute Programmierung		Symbolische Programmierung	
U	E 0.0	U	LS1
FP	M 0.0	FP	TM1
UN	E 0.1	UN	LS2
S	A 4.1	S	LINKS
U	E 0.1	U	LS2
FP	M 0.1	FP	TM2
UN	E 0.0	UN	LS1
S	A 4.0	S	RECHTS
UN	E 0.0	UN	LS1
UN	E 0.1	UN	LS2
R	A 4.0	R	RECHTS
R	A 4.1	R	LINKS

Anweisungsliste zur Richtungserfassung eines Förderbandes

AWL		Erläuterung
U	E 0.0	//Wenn an E 0.0 ein Wechsel des Signalzustands von "0" auf "1" auftritt (positive Flanke) und gleichzeitig der Signalzustand an E 0.1 "0" ist, dann bewegt sich das Paket auf dem Band nach links
EP	M 0.0	
UN	E 0.1	
S	A 4.1	
U	E 0.1	//Wenn an E 0.1 ein Wechsel des Signalzustands von "0" auf "1" auftritt (positive Flanke) und gleichzeitig der Signalzustand an E 0.0 "0" ist, dann bewegt sich das Paket auf dem Band nach rechts. Ist eine der Lichtschranken unterbrochen, dann befindet sich ein Paket zwischen den Schranken.
FP	M 0.1	
UN	E 0.0	
S	A 4.0	
UN	E 0.0	//Sind die Lichtschranken nicht unterbrochen, dann befindet sich kein Paket zwischen den Schranken. Die Richtungsanzeiger sind ausgeschaltet.
UN	E 0.1	
R	A 4.0	
R	A 4.1	

B.3 Zeitoperationen Beispiel

Taktgeber

Zur Erzeugung eines sich periodisch wiederholenden Signals können Sie einen Taktgeber oder ein Blinkrelais verwenden. Taktgeber finden sich häufig in Meldesystemen, die das Blinken von Anzeigeleuchten steuern.

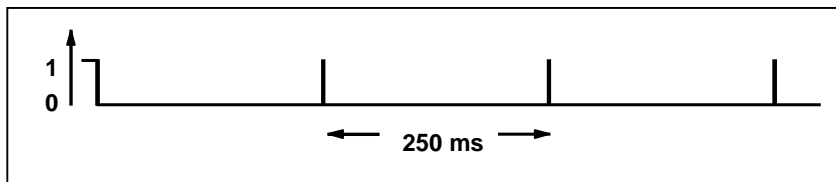
Wenn Sie S7-300 einsetzen, können Sie eine Taktgeberfunktion implementieren, indem Sie die zeitgesteuerte Verarbeitung in speziellen Organisationsbausteinen verwenden.

Anweisungsliste zum Generieren eines Taktes (Tastverhältnis 1:1)

AWL		Erläuterung
U	T1	//Wenn Zeit T1 abgelaufen ist
L	S5T#250ms	//dann lade den Zeitwert 250 ms in T1
SV	T1	//und starte T1 als verlängerten Impuls.
NOT		//Negiere das Verknüpfungsergebnis (kehre es um).
BEB		//Beende den aktuellen Baustein, wenn die Zeit läuft.
L	MB100	//Wenn die Zeit abgelaufen ist, dann lade den Inhalt von Merkerbyte //MB100
INC	1	//inkrementiere den Inhalt um "1"
T	MB100	//und transferiere das Ergebnis ins Merkerbyte MB100.

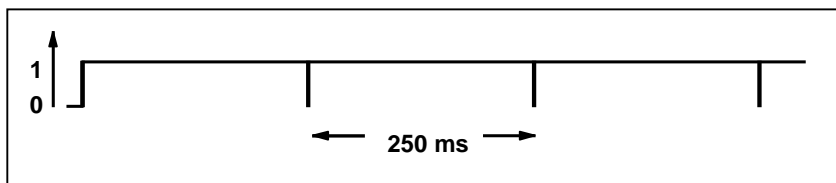
Signalabfrage

Eine Signalabfrage der Zeit T1 liefert für die Anweisung **UN T1** folgendes Verknüpfungsergebnis:



Sobald die Zeit abgelaufen ist, wird die Zeit erneut gestartet. Daher liefert die Signalabfrage, die von der Anweisung **UN T1** ausgeführt wird, nur kurz den Signalzustand "1".

Negiertes VKE-Bit der Zeit T1:



Alle 250 ms beträgt das VKE-Bit "0". Die Operation **BEB** beendet die Bearbeitung des Bausteins dann jedoch nicht. Stattdessen wird der Inhalt des Merkerbytes MB100 um "1" inkrementiert.

Der Inhalt des Merkerbytes MB100 verändert sich alle 250 ms wie folgt:

0 -> 1 -> 2 -> 3 -> ... -> 254 -> 255 -> 0 -> 1 ...

Erzielen einer bestimmten Frequenz

Mit den Bits von Merkerbyte MB100 können Sie folgende Frequenzen erzielen:

MB100	Frequenz in Hertz	Dauer
M 100.0	2.0	0.5 s (250 ms ein / 250 ms aus)
M 100.1	1.0	1 s (0.5 s ein / 0.5 s aus)
M 100.2	0.5	2 s (1 s ein / 1 s aus)
M 100.3	0.25	4 s (2 s ein / 2 s aus)
M 100.4	0.125	8 s (4 s ein / 4 s aus)
M 100.5	0.0625	16 s (8 s ein / 8 s aus)
M 100.6	0.03125	32 s (16 s ein / 16 s aus)
M 100.7	0.015625	64 s (32 s ein / 32 s aus)

Anweisungsliste

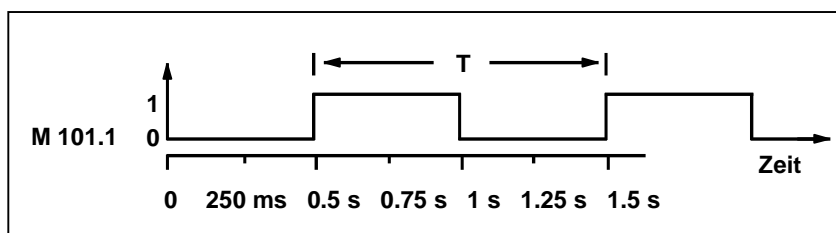
AWL	Erläuterung
U M10.0	//M10.0 ist "1", wenn ein Fehler auftritt. Tritt ein Fehler auf,
	//dann blinkt die Fehlerlampe mit einer Frequenz von 1 Hz auf.
U M100.1	
= A 4.0	

Signalzustände der Bits von Merkerbyte MB 100

Zyklus	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Zeitwert in ms
0	0	0	0	0	0	0	0	0	250
1	0	0	0	0	0	0	0	1	250
2	0	0	0	0	0	0	1	0	250
3	0	0	0	0	0	0	1	1	250
4	0	0	0	0	0	1	0	0	250
5	0	0	0	0	0	1	0	1	250
6	0	0	0	0	0	1	1	0	250
7	0	0	0	0	0	1	1	1	250
8	0	0	0	0	1	0	0	0	250
9	0	0	0	0	1	0	0	1	250
10	0	0	0	0	1	0	1	0	250
11	0	0	0	0	1	0	1	1	250
12	0	0	0	0	1	1	0	0	250

Signalzustand des Merkerbits M 101.1

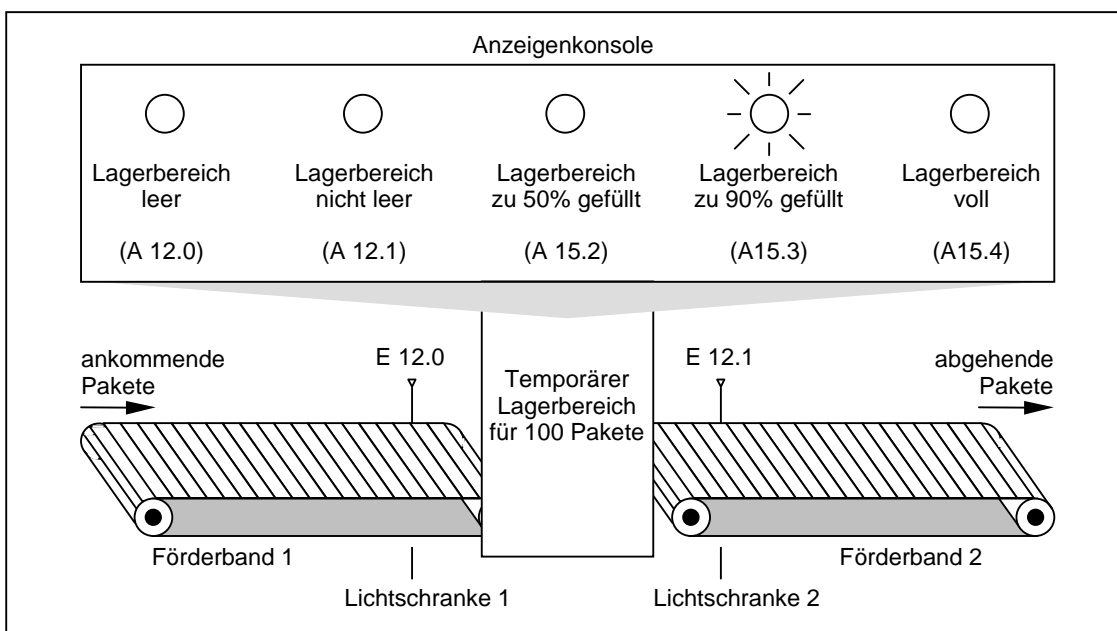
Frequenz = $1/T = 1/1 \text{ s} = 1 \text{ Hz}$



B.4 Zähl- und Vergleichsoperationen Beispiel

Lagerbereich mit Zähler und Vergleich

Das folgende Bild zeigt ein System mit zwei Förderbändern und einem temporären Lagerbereich dazwischen. Förderband 1 transportiert die Pakete zum Lagerbereich. Eine Lichtschranke am Ende des Förderbandes 1 neben dem Lagerbereich ermittelt, wie viele Pakete in den Lagerbereich transportiert werden. Förderband 2 transportiert Pakete von diesem temporären Lagerbereich zu einer Laderampe, wo sie zur Auslieferung beim Kunden auf LKW verladen werden. Eine Lichtschranke am Ende des Förderbandes 2 neben dem Lagerbereich ermittelt, wie viele Pakete aus dem Lagerbereich heraus zur Laderampe transportiert werden. Fünf Anzeigeleuchten zeigen an, wie weit der temporäre Lagerbereich gefüllt ist.



Anweisungsliste, die die Anzeigeleuchten aktiviert

AWL	Erläuterung
U	E 0.0 //Jeder durch die Lichtschranke 1 generierte Impuls
ZV	Z1 //erhöht den Zählwert des Zählers Z1 um "1", wodurch die Zahl der Pakete //gezählt wird, die in den Lagerbereich transportiert werden.
U	E 0.1 //Jeder durch die Lichtschranke 2 generierte Impuls
ZR	Z1 //vermindert den Zählwert des Zählers Z1 um "1", wodurch die Zahl der //Pakete gezählt wird, die den Lagerbereich verlassen.
UN	Z1 //Wenn der Zählwert "0" beträgt,
=	A 4.0 //schaltet sich die Anzeigeleuchte für die Meldung "Lagerbereich leer" //ein.
U	Z1 //Beträgt der Zählwert nicht "0",
=	A 4.1 //schaltet sich die Anzeigeleuchte für die Meldung "Lagerbereich nicht //leer" ein.
L	50
L	Z1
<=I	// st 50 kleiner oder gleich Zählwert,
=	A 4.2 //dann schaltet sich die Anzeigeleuchte für die Meldung "Lagerbereich zu //50% voll" ein.
L	90
>=I	//Ist der Zählwert größer oder gleich 90,
=	A 4.3 //dann schaltet sich die Anzeigeleuchte für die Meldung "Lagerbereich zu //90% voll" ein.
L	Z1
L	100
>=I	//Ist der Zählwert größer oder gleich 100,
=	A 4.4 //dann schaltet sich die Anzeigeleuchte für die Meldung "Lagerbereich //voll" ein. (Sie könnten auch über Ausgang A 4.4 das Förderband 1 //blockieren.)

B.5 Arithmetische Operationen mit Ganzzahlen Beispiel

Berechnen einer Gleichung

Das folgende Programmbeispiel zeigt, wie Sie mit drei arithmetischen Operationen für Ganzzahlen und den Operationen L und T das gleiche Ergebnis erzielen, wie die folgende Gleichung:

$$MD4 = ((EW0 + DB5.DBW3) \times 15) / MW2$$

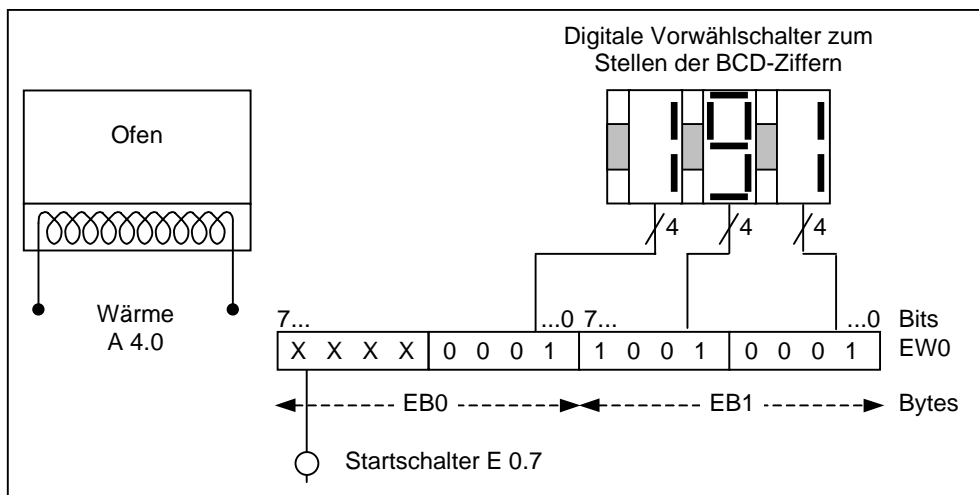
Anweisungsliste

AWL	Erläuterung
L	EW0 //Lade den Wert von Eingangswort EW0 in AKKU 1. Lade den Wert des Globaldatenworts DBW3 aus dem DB 5 in AKKU 1. Der alte Inhalt von AKKU 1 wird in AKKU 2 geschoben.
L	DB5.DBW3 //Lade den Wert des Globaldatenworts DBW3 aus dem DB 5 in AKKU 1. Der alte Inhalt von AKKU 1 wird in AKKU 2 geschoben.
+I	//Addiere den Inhalt der niederwertigen Wörter von AKKU 1 und AKKU 2. Das Ergebnis wird im niederwertigen Wort von AKKU 1 gespeichert. Der Inhalt von AKKU 2 und das höherwertige Wort von AKKU 1 bleiben unverändert.
L	15 //Lade den konstanten Wert +15 in AKKU 1. Der alte Inhalt von AKKU 1 wird in AKKU 2 geschoben.
*I	//Multipliziere den Inhalt des niederwertigen Worts von AKKU 2 mit dem Inhalt desniederwertigen Worts von AKKU 1. Das Ergebnis wird in AKKU 1 gespeichert. Der Inhalt von AKKU 2 bleibt unverändert.
L	MW2 //Lade den Wert von Merkerwort MW2 in AKKU 1. Der alte Inhalt von AKKU 1 wird in AKKU 2 geschoben.
/I	//Dividiere den Inhalt des niederwertigen Worts von AKKU 2 durch den Inhalt des niederwertigen Worts von AKKU 1. Das Ergebnis wird in AKKU 1 gespeichert. Der Inhalt von AKKU 2 bleibt unverändert.
T	MD4 //Transferiere das Endergebnis ins Merkerdoppelwort MD4. Der Inhalt beider Akkumulatoren bleibt unverändert.

B.6 Wortverknüpfungsoperationen Beispiel

Heizen eines Ofens

Der Bediener startet das Heizen des Ofens, indem er den Startschalter drückt. Mit den digitalen Vorwählschaltern kann er die Dauer der Heizzeit festlegen. Der Wert, den er setzt, gibt die Sekunden im binär-codierten Dezimalformat (BCD) an.



Systemkomponente	Absolute Adresse
Startschalter	E 0.7
Digitale Vorwählschalter für Einer	E 1.0 bis E 1.3
Digitale Vorwählschalter für Zehner	E 1.4 bis E 1.7
Digitale Vorwählschalter für Hunderter	E 0.0 bis E 0.3
Beginn Heizvorgang	A 4.0

Anweisungsliste

AWL	Erläuterung
U T1 = A 4.0	//Wenn die Zeit läuft, //dann beginne den Heizvorgang.
BEB	//Wenn die Zeit läuft, dann beende die Bearbeitung hier. Dadurch wird ein Neustart der Zeit T1 verhindert, wenn der Schalter gedrückt wird.
L EW0	
UW W#16#0FFF	//Maskiere die Eingangsbits E 0.4 bis E 0.7 (d.h. setze sie auf "0" zurück). Der Zeitwert in Sekunden befindet sich in binär-codiertem Dezimalformat im niederwertigen Wort von AKKU 1.
OW W#16#2000	//Ordne die Zeitbasis als Sekunden in Bits 12 und 13 des niederwertigen Wort von AKKU 1 zu.
U E 0.7	
SV T1	//Starte die Zeit T1 als verlängerten Impuls, wenn der Schalter gedrückt wird.

C Parameterübergabe

Die Parameter eines Bausteins werden als Wert übergeben. Bei Funktionsbausteinen wird innerhalb des aufgerufenen Bausteins eine Kopie des Aktualparameterwertes im Instanz-DB verwendet. Bei Funktionen liegt eine Kopie des Aktualwertes im Lokaldatenstack. Zeiger werden nicht kopiert. Vor dem Aufruf werden die INPUT-Werte in den Instanz-DB bzw auf den L-Stack kopiert. Nach dem Aufruf werden die OUTPUT-Werte zurück in die Variablen kopiert. Innerhalb des aufgerufenen Baustein arbeitet man nur auf einer Kopie. Die dafür notwendigen AWL-Befehlen befinden sich im aufrufenden Baustein und bleiben dem Anwender verborgen.

Hinweis

Wenn Merker, Eingänge, Ausgänge, Peripherieeingänge oder Peripherieausgänge als Aktualoperanden an einer Funktion verwendet werden, werden diese anders behandelt als die anderen Operanden. Die Aktualisierung erfolgt hier nicht über den L-Stack, sondern direkt.



Warnung

Sorgen Sie bei der Programmierung des aufgerufenen Bausteins dafür, daß die als OUTPUT deklarierten Parameter auch beschrieben werden. Sonst sind die ausgegebenen Werte zufällig! Bei Funktionsbausteinen bekommt man eben den vom letzten Aufruf gemerkten Wert aus dem Instanz-DB, bei Funktionen den zufällig auf dem L-Stack liegenden Wert.

Beachten Sie folgende Punkte:

- Initialisieren Sie wenn möglich alle OUTPUT Parameter.
 - Verwenden Sie möglichst keine Setze- und Rücksetze-Befehle. Diese Befehle sind VKE-abhängig. Wenn das VKE den Wert 0 hat, bleibt der zufällige Wert erhalten!
 - Wenn Sie innerhalb des Bausteins springen, so achten Sie darauf, daß Sie keine Stellen überspringen, in denen OUTPUT-Parameter beschrieben werden. Denken Sie dabei auch an BEB und die Wirkung der MCR-Befehle.
-

Index

)

) 1-16

*

*D 7-12

*I 7-5

*R 8-7

/

/D 7-13, 7-14

/I 7-6, 7-7

/R 8-8

?

? D 2-3

? I 2-2

? R 2-4

+

+ 7-9

+AR1 14-11

+AR2 14-12, 14-13

+D 7-10

+I 7-3

+R 8-3, 8-4

<

<=I 2-2

<>I 2-2

<I 2-2

=

= 1-18

==D 2-3

<>D 2-3

>D 2-3

<D 2-3

>=D 2-3

<=D 2-3

==I 2-2

==R 2-4

<>R 2-4

>R 2-4

<R 2-4

>=R 2-4

<=R 2-4

>

>=I 2-2

>I 2-2

1

1-Komplement Ganzzahl (16 Bit) 3-8

1-Komplement Ganzzahl (32 Bit) 3-9

2

2-Komplement Ganzzahl (16 Bit) 3-10

2-Komplement Ganzzahl (32 Bit) 3-11

A

ABS 8-9

Absolutwert Gleitpunktzahl (32 Bit IEEE-FP) .. 8-9

ACOS 8-18

Addiere AKKU 1 und 2 als Ganzzahl
(16 Bit) 7-3

Addiere AKKU 1 und 2 als Ganzzahl
(32 Bit) 7-10

Addiere AKKU 1 und 2 als Gleitpunktzahl
(32 Bit) 8-3

Addiere AKKU 1 zum Adreßregister 1 14-11

Addiere AKKU 1 zum Adreßregister 2 14-12

Addiere Ganzzahlkonstante (16 oder 32 Bit) .. 7-8

Akkumulatoroperationen Übersicht 14-1

Aktiviere MCR-Bereich 10-25

Arithmetische Operationen mit
Ganzzahlen Beispiel B-12

ASIN 8-17

ATAN 8-19

AUF 5-2

Aufrufen eines FBs 10-8

Aufrufen eines FCs 10-10

Aufrufen eines SFBs 10-12

Aufrufen eines SFCs 10-14

Auswerten der Bits im Statuswort bei
Festpunkt-Funktionen 7-2

Auswerten der Bits im Statuswort bei
Gleitpunkt-Funktionen 8-2

AWL-Operationen sortiert nach deutscher Mnemonik (SIMATIC)	A-1
AWL-Operationen sortiert nach englischer Mnemonik (International)	A-6

B

Baustein aus einer Bibliothek aufrufen.....	10-16
Bausteinaufruf.....	10-5
Bausteinende.....	10-2
Bausteinende absolut	10-4
Bausteinende bedingt	10-3
BCD wandeln in Ganzzahl (16 Bit).....	3-2
BCD wandeln in Ganzzahl (32 Bit).....	3-4
BE	10-2
BEA.....	10-4
BEB.....	10-3
Bedingter Bausteinaufruf	10-17
Beende MCR-Bereich	10-24
Beginn MCR-Bereich	10-22
Beispiele zur Programmierung.....	B-1
Bildbefehl (Nulloperation).....	14-14
Bilden der Quadratwurzel einer Gleitpunktzahl (32 Bit).....	8-11
Bilden des Arcuscossinus einer Gleitpunktzahl (32 Bit).....	8-18
Bilden des Arcussinus einer Gleitpunktzahl (32 Bit).....	8-17
Bilden des Arcustangens einer Gleitpunktzahl (32 Bit).....	8-19
Bilden des Cosinus eines Winkels als Gleitpunktzahl (32 Bit).....	8-15
Bilden des natürlichen Logarithmus einer Gleitpunktzahl (32 Bit).....	8-13
Bilden des Quadrats einer Gleitpunktzahl (32 Bit)	8-10
Bilden des Sinus eines Winkels als Gleitpunktzahlen (32 Bit).....	8-14
Bilden des Tangens eines Winkels als Gleitpunktzahlen (32 Bit).....	8-16
Bitverknüpfungsoperationen Beispiel.....	B-2
Bitverknüpfungsoperationen Übersicht	1-1
BLD	14-14
BTD.....	3-4
BTI	3-2

C

CALL.....	10-5, 10-6, 10-7
CC.....	10-17
CLR.....	1-23
COS	8-15

D

-D	7-11
Das MCR (Master Control Relay)	10-19
Datenbaustein öffnen.....	5-2
Datenbausteinoperationen Übersicht.....	5-1
Deaktiviere MCR-Bereich	10-26

DEC.....	14-10
Dekrementiere AKKU 1-L-L.....	14-10
Dividiere AKKU 2 durch 1 als Ganzzahl (16 Bit).....	7-6
Dividiere AKKU 2 durch 1 als Ganzzahl (32 Bit).....	7-13
Dividiere AKKU 2 durch 1 als Gleitpunktzahl (32 Bit)	8-8
Divisionsrest Ganzzahl (32 Bit)	7-15
DTB	3-6
DTR.....	3-7

E

Einer-Komplement Ganzzahl (16 Bit)	3-8
Einer-Komplement Ganzzahl (32 Bit)	3-9
ENT	14-7
Enter AKKU-Stack.....	14-7
Exklusiv Oder	1-7
Exklusiv Oder mit Verzweigung.....	1-14
Exklusiv Oder Nicht	1-8
Exklusiv Oder Nicht mit Verzweigung.....	1-15
EXKLUSIV-ODER-Doppelwort (32 Bit)	13-12
EXKLUSIV-ODER-Wort (16 Bit)	13-6
EXP Bilden des Exponentialwerts einer Gleitpunktzahl (32 Bit)	8-12

F

FB aufrufen.....	10-8
FC aufrufen	10-10
Festpunkt-Funktionen Übersicht.....	7-1
Flanke Negativ	1-25
Flanke Positiv	1-27
FN	1-25
FP	1-27
FR	4-2, 12-5, 12-6
Freigabe Timer	12-5
Freigabe Zähler	4-2

G

Ganzzahl (16 Bit) wandeln in BCD	3-3
Ganzzahl (16 Bit) wandeln in Ganzzahl (32 Bit).....	3-5
Ganzzahl (32 Bit) wandeln in BCD	3-6
Ganzzahl (32 Bit) wandeln in Gleitpunktzahl (32 Bit IEEE-FP)	3-7
Gleitpunkt-Funktionen Übersicht	8-1

I

-I	7-4
INC	14-8, 14-9
Inkrementiere AKKU 1-L-L	14-8
INVD.....	3-9
INVI	3-8
ITB.....	3-3
ITD	3-5

L	
L	4-3, 9-2
L DBLG	5-4
L DBNO	5-5
L DILG	5-6
L DINO	5-7
L STW	9-4
Lade	9-2
Lade Adreßregister 1 mit Inhalt von Adressregister 2	9-7
Lade Adreßregister 1 mit Inhalt von AKKU 1	9-5
Lade Adreßregister 2 mit Ganzzahl (32 Bit)	9-9
Lade Adreßregister 2 mit Inhalt von AKKU 1	9-8
Lade aktuellen Zählwert als BCD in AKKU 1	4-5
Lade aktuellen Zählwert als Ganzzahl in AKKU 1	4-3
Lade aktuellen Zeitwert als BCD in AKKU 1	12-9
Lade aktuellen Zeitwert als Ganzzahl in AKKU 1	12-7
Lade Länge Global-DB in AKKU 1	5-4
Lade Länge Instanz-DB in AKKU 1	5-6
Lade Nummer Global-DB in AKKU 1	5-5
Lade Nummer Instanz-DB in AKKU 1	5-7
Lade Statuswort in AKKU 1	9-4
Lade- und Transferoperationen Übersicht	9-1
LAR1	9-5
LAR1 <D> Lade Adreßregister 1 mit Pointer (32 Bit-Format)	9-6
LAR1 AR2	9-7
LAR2	9-8
LAR2 <D>	9-9
LC	4-5, 4-6, 12-9, 12-10
LEAVE	14-8
Leave AKKU-Stack	14-8
LN	8-13
LOOP	6-24, 6-25
M	
MCR	10-22, 10-23, 10-24, 10-25, 10-26
MCR(.....	10-22, 10-23
MCR)	10-24
MCRA	10-25
MCR-Bereich	10-25
MCRD	10-26
Mnemonic englisch/international	A-6
Mnemonic deutsch/SIMATIC	A-1
MOD	7-15
Multiinstanz aufrufen	10-16
Multipliziere AKKU 1 und 2 als Ganzzahl (16 Bit)	7-5
Multipliziere AKKU 1 und 2 als Ganzzahl (32 Bit)	7-12
Multipliziere AKKU 1 und 2 als Gleitpunktzahl (32 Bit)	8-7
N	
NEGD	3-11
NEGI	3-10
Negiere Gleitpunktzahl	3-12
Negiere VKE	1-21
NEGR	3-12
NOP 0	14-15
NOP 1	14-16
NOT	1-21
Nulloperation	14-15, 14-16
O	
O	1-5, 1-9
O(.....	1-12
OD	13-10, 13-11
Oder	1-5
Oder mit Verzweigung	1-12
Oder Nicht	1-6
Oder Nicht mit Verzweigung	1-13
ODER-Doppelwort (32 Bit)	13-10
ODER-Wort (16 Bit)	13-4
ON	1-6
ON(.....	1-13
OW	13-4, 13-5
P	
Parameterübergabe	C-1
POP	14-5, 14-6
CPU mit vier Akkus	14-6
CPU mit zwei Akkus	14-5
Praktische Anwendung B-1, B-2, B-7, B-12, B-13	
Programmierbeispiele Übersicht	B-1
Programmschleife	6-24
Programmsteuerungsoperationen Übersicht	10-1
PUSH	14-2, 14-3, 14-4
CPU mit vier Akkus	14-4
CPU mit zwei Akkus	14-2
R	
R	1-19, 4-7, 12-11
-R	8-5
-R	8-5
-R	8-6
RLD	11-14, 11-15
RLDA	11-18
RND	3-15
RND-	3-18
RND-	3-18
RND-	3-18
RND-	3-18
RND+	3-17
Rotiere Akku 1 links über A1-Anzeige (32 Bit)	11-18
Rotiere Akku 1 rechts über A1-Anzeige (32 Bit)	11-19

Rotiere links Doppelwort (32 Bit).....	11-14
Rotiere rechts Doppelwort (32 Bit).....	11-16
Rotieroperationen Übersicht	11-14
RRD.....	11-16, 11-17
RRDA.....	11-19
Rücksetze	1-19
Rücksetze Timer	12-11
Rücksetze VKE (=0)	1-23
Rücksetze Zähler	4-7
Runden einer Gleitpunktzahl durch	
Abschneiden	3-16
Runden einer Gleitpunktzahl zur Ganzzahl..	3-15
Runden einer Gleitpunktzahl zur	
nächsthöheren Ganzzahl	3-17
Runden einer Gleitpunktzahl zur	
nächstniederen Ganzzahl	3-18
S	
S	1-20, 4-8
SA	12-20, 12-21
SAVE	1-24
Schiebe links Doppelwort (32 Bit)	11-10
Schiebe links Wort (16 Bit).....	11-6
Schiebe rechts Doppelwort (32 Bit).....	11-12
Schiebe rechts Wort (16 Bit)	11-8
Schiebe Vorzeichen rechts Ganzzahl	
(16 Bit)	11-2
Schiebe Vorzeichen rechts Ganzzahl	
(32 Bit)	11-4
Schiebeoperationen Übersicht.....	11-1
SE	12-16, 12-17
SET	1-22
Setze.....	1-20
Setze VKE (=1)	1-22
Setze Zählerstartwert.....	4-8
SFB aufrufen.....	10-12
SFC aufrufen.....	10-14
SI	12-12, 12-13
Sichere VKE im BIE-Bit.....	1-24
Sichere VKE im MCR-Stack.....	10-22
SIN.....	8-14
SLD.....	11-10, 11-11
SLW	11-6, 11-7
SPA.....	6-3
SPB.....	6-6
SPBB	6-8
SPBI.....	6-10
SPBIN	6-11
SPBN	6-7
SPBNB.....	6-9
Speicherbereiche und Komponenten	
einer Zeit.....	12-2
SPL.....	6-4, 6-5
SPM.....	6-19
SPMZ.....	6-21
SPN	6-17
SPO	6-12, 6-13
SPP.....	6-18
SPPZ	6-20
Springe	
wenn BIE = 0.....	6-11
wenn BIE = 1	6-10
wenn Ergebnis < 0.....	6-19
wenn Ergebnis <= 0	6-21
wenn Ergebnis <> 0	6-17
wenn Ergebnis = 0.....	6-16
wenn Ergebnis > 0.....	6-18
wenn Ergebnis >= 0	6-20
wenn Ergebnis ungültig	6-22
wenn OS = 1.....	6-14
wenn OV = 1.....	6-12
wenn VKE = 0.....	6-7
wenn VKE = 0 und rette VKE ins BIE	6-9
wenn VKE = 1.....	6-6
wenn VKE = 1 und rette VKE ins BIE	6-8
Springe absolut	6-3
Sprungleiste	6-4
Sprungoperationen Übersicht.....	6-1
SPS	6-14, 6-15
SPU	6-22, 6-23
SPZ	6-16
SQR	8-10
SQRT	8-11
SRD.....	11-12, 11-13
SRW	11-8, 11-9
SS	12-18, 12-19
SSD.....	11-4, 11-5
SSI	11-2, 11-3
Subtrahiere AKKU 1 von 2 als	
Ganzzahl (16 Bit).....	7-4
Subtrahiere AKKU 1 von 2 als	
Ganzzahl (32 Bit).....	7-11
Subtrahiere AKKU 1 von 2 als	
Gleitpunktzahl (32 Bit)	8-5
SV	12-14, 12-15
T	
T	9-10
T STW	9-11
TAD	3-14
TAK	14-2
TAN	8-16
TAR	9-12
TAR1	9-13
TAR1 <D>	9-14
TAR1 AR2	9-15
TAR2	9-16
TAR2 <D>	9-17
Tausche Adreßregister 1 mit 2	9-12
Tausche AKKU 1 mit AKKU 2	14-2
Tausche Global-DB und Instanz-DB	5-3
Tausche Reihenfolge der Bytes im	
AKKU 1 (32 Bit)	3-14
Tausche Reihenfolge der Bytes im	
AKKU 1-L (16 Bit).....	3-13
TAW	3-13
TDB	5-3
Transferiere	9-10

Transferiere Adreßregister 1 in Adreßregister 2	9-15
Transferiere Adreßregister 1 in AKKU 1	9-13
Transferiere Adreßregister 1 nach Zieladresse (32-Bit-Pointer)	9-14
Transferiere Adreßregister 2 in AKKU 1	9-16
Transferiere Adreßregister 2 nach Zieladresse (32-Bit-Pointer)	9-17
Transferiere AKKU 1 in Statuswort	9-11
Transferoperationen Übersicht.....	9-1
TRUNC	3-16

U

U	1-3
U(.....	1-10
Übersicht	
Akkumulatoroperationen	14-1
Bitverknüpfungsoperationen	1-1
Datenbausteinoperationen	5-1
Festpunkt-Funktionen	7-1
Gleitpunkt-Funktionen	8-1
Lade- und Transferoperationen.....	9-1
Programmierbeispiele	B-1
Programmsteuerungsoperationen.....	10-1
Rotieroperationen	11-14
Schiebeoperationen	11-1
Sprungoperationen	6-1
Umwandlungsoperationen	3-1
Vergleichsoperationen	2-1
Wortverknüpfungsoperationen	13-1
Zähloperationen	4-1
Zeitoperationen	12-1
UC.....	10-18
UD.....	13-8, 13-9
Umwandlungsoperationen Übersicht	3-1
UN.....	1-4
UN(.....	1-11
Unbedingter Bausteinanruf	10-18
Und	1-3
Und mit Verzweigung.....	1-10
Und Nicht	1-4
Und Nicht mit Verzweigung.....	1-11
Und vor Oder	1-9
UND-Doppelwort (32 Bit)	13-8
UND-Wort (16 Bit).....	13-2
UW.....	13-2, 13-3

V

Vergleiche Ganzzahlen (16 Bit).....	2-2
Vergleiche Ganzzahlen (32 Bit).....	2-3
Vergleiche Gleitpunktzahlen (32 Bit)	2-4
Vergleichsoperationen Übersicht.....	2-1
Verzweigung schließen	1-16
Vorzeichen einer Gleitpunktzahl wechseln ...	3-12

W

Wichtige Hinweise zur Benutzung der MCR-Funktionalität.....	10-21
Wortverknüpfungsoperationen Beispiel	B-13
Wortverknüpfungsoperationen Übersicht	13-1

X

X	1-7
X(.....	1-14
XN	1-8
XN(.....	1-15
XOD	13-12, 13-13
XOW.....	13-6, 13-7

Z

Zähl- und Vergleichsoperationen Beispiel ...	B-10
Zählen rückwärts	4-10
Zählen vorwärts	4-9
Zähloperationen Übersicht	4-1
Zeit als Ausschaltverzögerung	12-20
Zeit als Einschaltverzögerung	12-16
Zeit als Impuls	12-12
Zeit als speichernde Einschaltverzögerung	12-18
Zeit als verlängerter Impuls	12-14
Zeit einstellen	12-2
Zeitbasis	12-2, 12-3
Zeitoperationen Beispiel	B-7
Zeitoperationen Übersicht	12-1
Zeitwert.....	12-2, 12-3, 12-4
ZR	4-10
Zuweisung	1-18
ZV.....	4-9
Zweier-Komplement Ganzzahl (16 Bit).....	3-10
Zweier-Komplement Ganzzahl (32 Bit).....	3-11

