

SIEMENS

SIMATIC

S7-300 および S7-400 プログラミング 用ステートメントリスト (STL)

リファレンスマニュアル

ビットロジック命令

1

比較命令

2

変換命令

3

カウンタ命令

4

データブロック命令

5

論理制御命令

6

整数演算命令

7

浮動小数点数値演算命令

8

ロード命令と転送命令

9

プログラム制御命令

10

シフト命令および回転命令

11

タイマ命令

12

ワード論理命令

13

アキュムレータ命令

14

すべての STL 命令の概要

A

プログラミング例

B

パラメータ転送

C

03/2012

A5E03681272-01

法律上の注意

警告事項

本書には、ユーザーの安全を守るため、および製品や接続された機器の損傷を防ぐために遵守すべき注意事項が記載されています。ユーザーの安全性に関する注意事項は、安全警告サインで強調表示されています。このサインは、物的損傷に関する注意事項には表示されません。

危険

適切な予防措置を講じなければ、きわめて高い可能性で、死亡、重傷、または機器の重大な損傷を引き起こす恐れがあります。

警告

適切な予防措置を講じない場合、死亡、重傷、または機器の重大な損傷を引き起こす恐れがあります。

注意

回避しなければ、軽度または中度の人身傷害を引き起こすおそれのある危険な状況を示します（安全警告サイン付き）。

注意

回避しなければ、物的損傷を引き起こすおそれのある危険な状況を示します（安全警告サインなし）。

通知

回避しなければ、望ましくない結果や状態が生じ得る状況を示します（安全警告サインなし）。

複数の危険度が存在する場合、一番高い危険度を表す警告が使用されます。安全警告シンボルを伴う人的傷害に関する警告には、物的損害に関する警告も含まれる場合があります。

有資格者

装置/システムのセットアップおよび使用にあたっては必ず本マニュアルを参照してください。機器のインストールおよび操作は**有資格者**のみが行うものとします。有資格者とは、法的な安全規制/規格に準拠してアースの取り付け、電気回路、設備およびシステムの設定に携わることを承認されている技術者のことをいいます。

Siemens 製品を正しくお使いいただくために

以下の点に注意してください。

警告

Siemens 製品は、カタログおよび付属の技術説明書の指示に従ってお使いください。他社の製品または部品との併用は、弊社の推奨もしくは許可がある場合に限りです。Siemens 製品を正しく安全にご使用いただくには、適切な運搬、保管、組み立て、据え付け、配線、始動、操作、保守を行ってください。ご使用になる場所は、許容された範囲を必ず守ってください。付属の技術説明書に記述されている指示を遵守してください。

商標

本書において®で識別されるすべての名称は、Siemens AG の登録商標です。その他、この文書に記載されている会社名や製品名は各社の商標であるため、第三者が自己の目的のためにこれらの名前を使用すると、商標所有者の権利を侵害する恐れがあります。

免責事項

本書の内容は、記載されているハードウェアやソフトウェアとの齟齬がないよう見直されています。しかしながら、相違点をすべて取り除くことはできないため、完全な一致を保証するものではありません。本マニュアルの内容は定期的に見直され、必要な修正は次回の版で行われます。

はじめに

目的

このマニュアルは STL（ステートメントリストプログラミング言語）のユーザプログラムを作成するための手引きです。

STL 言語要素の構文、ファンクションを説明する参照ページも含まれています。

必要な基本知識

このマニュアルの対象者は、S7 プログラマ、オペレータ、保守/サービス要員です。

本マニュアルを理解するには、自動化技術の全般的な知識が必要です。

さらに、コンピュータについての知識、オペレーティングシステム MS Windows XP、MS Windows Server 2003、MS Windows 7 の下での PC（プログラミングデバイス等）同様やその他の動作機器の知識が必須です。

本マニュアルの対応バージョン

本マニュアルは STEP 7 プログラミングソフトウェアパッケージの 5.5 版用です。

標準との適合

STL は IEC（国際電気標準会議）-1131-3 で定義された「命令リスト」に対応します。ただし、運用に関してはかなりの差異があります。詳細は、STEP 7 ファイル NORM_TBL.RTF の標準表を参照してください。

必要条件

ステートメントリストマニュアルを効果的に使用するには、STEP 7 のオンラインヘルプに記載されている S7 プログラムの背後にある理論に精通する必要があります。また、言語パッケージは STEP 7 の標準ソフトウェアを使用するので、本ソフトウェアの操作に慣れ、付属マニュアルをよく読んでおいてください。

本マニュアルは「STEP 7 リファレンス」ドキュメントパッケージの一部です。

次の表で、STEP 7 のマニュアルの概要をご覧ください。

マニュアル	目的	オーダ番号
STEP 7 基本情報 - STEP 7 V5.3 の使い方 入門編 - STEP 7 によるプログラミング - STEP 7 によるハードウェアと通信接続のコンフィグレーション - S5 から S7 へ、コンバータマニュアル	STEP 7 と S7-300/400 プログラマブルコントローラによるコントロールタスクの実装方法を記述した、技術担当要員向けの基本情報	6ES7810-4CA10-8BW0
STEP 7 リファレンスと - S7-300/400 用ラダーロジック (LAD) / ファンクションブロックダイアグラム (FBD) / ステートメントリスト (STL) マニュアル 『S7-300/400 用標準およびシステムファンクション』ボリューム 1 とボリューム 2	参照情報を提供し、STEP 7 の基本情報の範囲を越えるプログラミング言語 LAD、FBD、STL および標準ファンクション、システムファンクションについて説明	6ES7810-4CA10-8BW1

オンラインヘルプ	目的	オーダ番号
STEP 7 のヘルプ	オンラインヘルプ形式で STEP 7 によるプログラミングとハードウェアのコンフィグレーションの基本情報を提供	STEP 7 標準ソフトウェアに付属
STL/LAD/FBD の参照ヘルプ SFB/SFC の参照ヘルプ オーガニゼーションブロックの参照ヘルプ	状況に応じた参照情報	STEP 7 標準ソフトウェアに付属

オンラインヘルプ

本マニュアルはソフトウェアに組み込まれているオンラインヘルプと併せてご使用ください。このオンラインヘルプは、STEP 7 を使用する際に詳細なサポートを提供することを目的としています。

ソフトウェアに組み込まれたヘルプシステムでは、いくつかのインターフェースが利用できます。

- 状況に応じたヘルプは、現在のコンテキスト、たとえば開いたダイアログ、アクティブなウィンドウについての情報を提供します。状況に応じたヘルプは、メニューコマンドの[ヘルプ]状況に応じたヘルプを選択するか、[F1]キーを押すか、ツールバーの疑問符マークの記号を使います。
- [STEP 7 のヘルプ]を呼び出すには、メニューコマンドから[ヘルプ]目次または状況に応じたヘルプウィンドウの[STEP 7 のヘルプ]ボタンを使用します。
- [用語集]ボタンで、STEP 7 アプリケーションの用語集を呼び出すことができます。

このマニュアルは、「ステートメントリストのヘルプ」からの抜粋です。マニュアルとオンラインヘルプがほぼ同一構造なので、マニュアルとオンラインヘルプを交互に容易に利用できます。

その他のサポート

技術的な質問がある場合、シーメンスの担当者または代理店の担当者に連絡をとってください。

連絡先は下記のアドレスで検索できます。

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

各 SIMATIC 製品およびシステムの技術文書のガイドは、以下でご覧になれます。

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

オンラインカタログおよび注文システムは以下にあります。

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

トレーニングセンター

Siemens 社は、SIMATIC S7 オートメーションシステムに精通していただくためのたくさんのトレーニングコースを用意しております。詳しくは、該当地区のトレーニングセンターか、下記のドイツ D 90026 ニュルンベルクの中央トレーニングセンターにご連絡ください。

インターネット: <http://www.sitrain.com>

技術サポート

すべての産業用オートメーション&ドライブテクノロジー製品のテクニカルサポートは次のとおりです。

- サポートリクエスト Web フォーム経由
<http://www.siemens.com/automation/support-request>

テクニカルサポートについての追加情報は、インターネットの
<http://www.siemens.com/automation/service> ページにあります。

インターネットによるサービスとサポート

マニュアルの他に、ノウハウを次のインターネットで提供します。

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

内容:

- ニュースレター、製品に関する最新情報を提供
- 「Service & Support」の検索機能を利用して必要な文書を検索
- フォーラム、世界中のユーザや専門家がその経験を交換
- 産業用オートメーション&ドライブテクノロジーを担当するお客様の最寄りのお問い合わせ先
- フィールドサービス、修理、スペアパーツ、コンサルティングについての情報

目次

1	ビットロジック命令.....	13
1.1	ビット論理命令の概要.....	13
1.2	A AND.....	15
1.3	AN AND NOT.....	16
1.4	O OR.....	17
1.5	ON OR NOT.....	18
1.6	X 排他的 OR.....	19
1.7	XN 排他的 OR NOT.....	20
1.8	O AND の後に OR.....	21
1.9	A (AND/ネストを開く.....	22
1.10	AN (AND NOT/ネストを開く.....	23
1.11	O (OR/ネストを開く.....	23
1.12	ON (OR NOT/ネストを開く.....	24
1.13	X (排他的 OR/ネストを開く.....	24
1.14	XN (排他的 OR NOT/ネストを開く.....	25
1.15	) ネストを閉じる.....	25
1.16	= 割り付け.....	27
1.17	R リセット.....	28
1.18	S セット.....	29
1.19	NOT RLO の否定.....	30
1.20	SET RLO (=1) の設定.....	30
1.21	CLR RLO (=0) のクリア.....	32
1.22	SAVE RLO を BR レジスタに保存.....	33
1.23	FN 信号立ち下がり.....	34
1.24	FP 信号立ち上がり.....	36
2	比較命令.....	39
2.1	比較命令の概要.....	39
2.2	?I 整数 (16 ビット) の比較.....	40
2.3	?D 倍長整数 (32 ビット) の比較.....	41
2.4	?R 浮動小数点数 (32 ビット) の比較.....	42
3	変換命令.....	43
3.1	変換命令の概要.....	43
3.2	BTI BCD から整数 (16 ビット) への変換.....	44
3.3	ITB 整数 (16 ビット) から BCD への変換.....	45
3.4	BTD BCD から整数 (32 ビット) への変換.....	46
3.5	ITD 整数 (16 ビット) から倍長整数 (32 ビット) への変換.....	47
3.6	DTB 倍長整数 (32 ビット) から BCD への変換.....	48
3.7	DTR 倍長整数 (32 ビット) から浮動小数点数への変換 (32 ビット、IEEE 754).....	49
3.8	INVI 整数の 1 の補数 (16 ビット).....	50
3.9	INVD 倍長整数の 1 の補数 (32 ビット).....	51
3.10	NEGI 整数の 2 の補数 (16 ビット).....	52
3.11	NEGD 倍長整数の 2 の補数 (32 ビット).....	53

3.12	NEGR	浮動小数点数 (32 ビット、IEEE 754) の否定	54
3.13	CAW	ACCU 1-L のバイトシーケンス (16 ビット) を変更	55
3.14	CAD	ACCU 1 のバイトシーケンス (32 ビット) を変更	56
3.15	RND	丸め	57
3.16	TRUNC	切り捨て	58
3.17	RND+	倍長整数の切り上げ	59
3.18	RND-	倍長整数の切り下げ	60
4	カウンタ命令		61
4.1		カウンタ命令の概要	61
4.2	FR	カウンタの有効化 (フリー)	62
4.3	L	現在のカウンタ値を ACCU 1 へロード	63
4.4	LC	現在のカウンタ値を BCD で ACCU 1 にロード	64
4.5	R	カウンタのリセット	66
4.6	S	カウンタプリセット値の設定	67
4.7	CU	カウンタアップ	68
4.8	CD	カウンタダウン	69
5	データブロック命令		71
5.1		データブロック命令の概要	71
5.2	OPN	データブロックを開く	72
5.3	CDB	共有 DB とインスタンス DB の交換	73
5.4	L DBLG	共有 DB のサイズを ACCU 1 にロード	73
5.5	L DBNO	共有 DB の数を ACCU 1 にロード	74
5.6	L DILG	インスタンス DB のサイズを ACCU 1 にロード	74
5.7	L DINO	インスタンス DB の数を ACCU 1 へロード	75
6	論理制御命令		77
6.1		論理制御命令の概要	77
6.2	JU	無条件ジャンプ	79
6.3	JL	ラベルへジャンプ	80
6.4	JC	RLO = 1 のときにジャンプ	82
6.5	JCN	RLO = 0 のときにジャンプ	83
6.6	JCB	RLO = 1 のときに BR に保存してジャンプ	84
6.7	JNB	RLO = 0 のときに BR に保存してジャンプ	85
6.8	JB	BR = 1 のときにジャンプ	86
6.9	JNBI	BR = 0 のときにジャンプ	87
6.10	JO	OV = 1 のときにジャンプ	88
6.11	JOS	OS = 1 のときにジャンプ	89
6.12	JZ	ゼロのときにジャンプ	91
6.13	JN	ゼロ以外のときにジャンプ	92
6.14	JP	プラスのときにジャンプ	93
6.15	JM	マイナスのときにジャンプ	94
6.16	JPZ	プラスまたはゼロのときにジャンプ	95
6.17	JMZ	マイナスまたはゼロのときにジャンプ	96
6.18	JUO	比較不能のときにジャンプ	97
6.19	LOOP	ループ	99

7	整数演算命令	101
7.1	整数値演算命令の概要	101
7.2	整数値演算命令におけるステータスワードのビットの評価	102
7.3	+I ACCU 1 と ACCU 2 の加算 (16 ビット整数)	103
7.4	-I ACCU 2 から ACCU 1 を減算 (16 ビット整数)	104
7.5	*I ACCU 1 と ACCU 2 の乗算 (16 ビット整数)	105
7.6	I ACCU 2 を ACCU 1 で除算 (16 ビット整数)	106
7.7	+ 整数値の加算 (16 ビット、32 ビット)	108
7.8	+D ACCU 1 と ACCU 2 の加算 (32 ビット倍長整数)	110
7.9	-D ACCU 2 から ACCU 1 を減算 (32 ビット倍長整数)	111
7.10	*D ACCU 1 と ACCU 2 の乗算 (32 ビット倍長整数)	112
7.11	D ACCU 2 を ACCU 1 で除算 (32 ビット倍長整数)	113
7.12	MOD 除算により余りを生成 (32 ビット倍長整数)	114
8	浮動小数点数値演算命令	115
8.1	浮動小数点数値演算命令の概要	115
8.2	浮動小数点数値演算命令におけるステータスワードのビットの評価	116
8.3	浮動小数点数値演算命令: 基本	117
8.3.1	+R ACCU 1 と ACCU 2 の加算 (32 ビットの IEEE 754 浮動小数点数)	117
8.3.2	-R ACCU 2 から ACCU 1 を減算 (32 ビットの IEEE 754 浮動小数点数)	119
8.3.3	*R ACCU 1 と ACCU 2 の乗算 (32 ビットの IEEE 754) 浮動小数点数	120
8.3.4	R ACCU 2 を ACCU 1 で除算 (32 ビットの IEEE 754 浮動小数点数)	121
8.3.5	ABS 浮動小数点数の絶対値 (32 ビットの IEEE 754)	122
8.4	浮動小数点数値演算命令: 拡張	123
8.4.1	SQR 浮動小数点数 (32 ビット) の 2 乗を生成	123
8.4.2	SQRT 浮動小数点数 (32 ビット) の平方根を生成	124
8.4.3	EXP 浮動小数点数 (32 ビット) の指数値を生成	125
8.4.4	LN 浮動小数点数 (32 ビット) の自然対数を生成	126
8.4.5	SIN 角度のサイン (32 ビット浮動小数点数) を生成	127
8.4.6	COS 角度のコサイン (32 ビット浮動小数点数) を生成	128
8.4.7	TAN 角度のタンジェント (32 ビット浮動小数点数) を生成	129
8.4.8	ASIN 浮動小数点数 (32 ビット) のアークサインを生成	130
8.4.9	ACOS 浮動小数点数 (32 ビット) のアークコサインを生成	131
8.4.10	ATAN 浮動小数点数 (32 ビット) のアークタンジェントを生成	132
9	ロード命令と転送命令	133
9.1	ロード命令と転送命令の概要	133
9.2	L ロード	134
9.3	L STW ステータスワードを ACCU 1 にロード	136
9.4	LAR1 ACCU 1 からアドレスレジスタ 1 をロード	137
9.5	LAR1 <D> アドレスレジスタ 1 を倍長整数 (32 ビットポインタ) と共にロード	138
9.6	LAR1 AR2 アドレスレジスタ 2 からアドレスレジスタ 1 をロード	139
9.7	LAR2 ACCU 1 からアドレスレジスタ 2 をロード	139
9.8	LAR2 <D> 倍長整数 (32 ビットポインタ) と共にアドレスレジスタ 2 をロード	140
9.9	T 転送	141
9.10	T STW ACCU 1 をステータスワードへ転送	142
9.11	CAR アドレスレジスタ 1 をアドレスレジスタ 2 と交換	143
9.12	TAR1 アドレスレジスタ 1 を ACCU 1 へ転送	143

9.13	TAR1 <D>	アドレスレジスタ 1 を宛先へ転送 (32 ビットポインタ)	144
9.14	TAR1 AR2	アドレスレジスタ 1 をアドレスレジスタ 2 へ転送	145
9.15	TAR2	アドレスレジスタ 2 を ACCU 1 へ転送	145
9.16	TAR2 <D>	アドレスレジスタ 2 を宛先へ転送 (32 ビットポインタ)	146
10	プログラム制御命令		147
10.1	プログラム制御命令の概要		147
10.2	BE	ブロックの終了	148
10.3	BEC	条件付きのブロックの終了	149
10.4	BEU	無条件ブロックの終了	150
10.5	CALL	ブロック呼び出し	151
10.6	FB	の呼び出し	154
10.7	FC	の呼び出し	156
10.8	SFB	の呼び出し	158
10.9	SFC	の呼び出し	160
10.10	複数インスタンスの呼び出し		161
10.11	ライブラリからのブロックの呼び出し		161
10.12	CC	条件付き呼び出し	162
10.13	UC	無条件呼び出し	163
10.14	MCR (マスタコントロールリレー)		164
10.15	MCR	ファンクションの使用方法に関する重要注記	166
10.16	MCR (	MCR オン	167
10.17	) MCR	MCR 終了	169
10.18	MCRA	MCR 領域の有効化	170
10.19	MCRD	MCR 領域の無効化	171
11	シフト命令および回転命令		173
11.1	シフト命令		173
11.1.1	シフト命令の概要		173
11.1.2	SSI	符号付きシフト整数 (16 ビット)	174
11.1.3		符号付きシフトダブル整数 (32 ビット)	176
11.1.4	SLW	左シフトワード (16 ビット)	178
11.1.5	SRW	右シフトワード (16 ビット)	180
11.1.6	SLD	左シフトダブルワード (32 ビット)	182
11.1.7	SRD	右シフトダブルワード (32 ビット)	184
11.2	回転命令		186
11.2.1	回転命令の概要		186
11.2.2	RLD	左循環ダブルワード (32 ビット)	187
11.2.3	RRD	右循環ダブルワード (32 ビット)	189
11.2.4	RLDA	CC 1 により ACCU 1 を左へ循環 (32 ビット)	191
11.2.5	RRDA	CC 1 により ACCU 1 を右へ循環 (32 ビット)	192
12	タイマ命令		193
12.1	タイマ命令の概要		193
12.2	メモリ内のタイマの場所およびタイマのコンポーネント		194
12.3	FR	タイマの有効化 (フリー)	197
12.4	L	現在のタイマ値を ACCU 1 に整数でロード	199
12.5	LC	現在のタイマ値を BCD として ACCU 1 にロード	201

12.6	R	リセットタイマ	203
12.7	SP	パルスタイマ	204
12.8	SE	拡張パルスタイマ	206
12.9	SD	オンディレイタイマ	208
12.10	SS	保持型オンディレイタイマ	210
12.11	SF	オフディレイタイマ	212
13	ワード論理命令		215
13.1	ワード論理命令の概要		215
13.2	AW	AND ワード (16 ビット)	216
13.3	OW	OR ワード (16 ビット)	218
13.4	XOW	排他的 OR ワード (16 ビット)	220
13.5	AD	AND ダブルワード (32 ビット)	222
13.6	OD	OR ダブルワード (32 ビット)	224
13.7	XOD	排他的 OR ダブルワード (32 ビット)	226
14	アキュムレータ命令		229
14.1	アキュムレータの演算とアドレスレジスタ命令の概要		229
14.2	TAK	ACCU 1 と ACCU 2 の切り替え	230
14.3	POP	ACCU が 2 つの CPU	231
14.4	POP	ACCU が 4 つの CPU	232
14.5	PUSH	ACCU が 2 つの CPU	233
14.6	PUSH	ACCU が 4 つの CPU	234
14.7	ENT	ACCU スタックに入る	235
14.8	LEAVE	ACCU スタックから出る	235
14.9	INC	ACCU 1-L-L に加算	236
14.10	DEC	ACCU 1-L-L からの減算	237
14.11	+AR1	ACCU 1 をアドレスレジスタ 1 に追加	238
14.12	+AR2	ACCU 1 をアドレスレジスタ 2 に追加	239
14.13	BLD	プログラム表示命令 (Null)	240
14.14	NOP 0	Null 命令	240
14.15	NOP 1	Null 命令	241
A	すべての STL 命令の概要		243
A.1	ドイツ語のプログラム表記法 (SIMATIC) に従ってソートされた STL 命令		243
A.2	英語のプログラム表記法 (インターナショナル) に従ってソートされた STL 命令		248
B	プログラミング例		253
B.1	プログラミングの概要例		253
B.2	例: ビットロジック命令		254
B.3	例: タイマ命令		257
B.4	例: カウンタ命令と比較命令		260
B.5	例: 整数値演算命令		262
B.6	例: ワードロジック命令		263
C	パラメータ転送		265
索引			267

1 ビットロジック命令

1.1 ビット論理命令の概要

説明

ビット論理命令は、1 と 0 の 2 つの桁で動作します。これらの 2 つの桁は、2 進法という記数法の基本となります。2 つの数字 1 と 0 は 2 進数字またはビットと呼ばれます。接点やコイルにおいて、1 は有効または通電状態であることを示し、0 は有効ではない、通電状態ではないことを示します。

ビットロジック命令は、信号状態 1 と 0 を解釈し、ブールロジックに応じてこれらの状態を結合します。これらの結合により、“論理演算結果”（RLO）と呼ばれる 1 または 0 の結果が生成されます。

ブールビットロジックは、以下の基本命令に適用されます。

- A AND
- AN AND NOT
- O OR
- ON OR NOT
- X 排他的 OR
- XN 排他的 OR NOT
- O AND の後に OR

以下の命令を使用すれば、ネスト構造表現を実行できます。

- A (AND/ネストを開く
- AN (AND NOT/ネストを開く
- O (OR/ネストを開く
- ON (OR NOT/ネストを開く
- X (排他的 OR/ネストを開く
- XN (排他的 OR NOT/ネストを開く
-) ネストを閉じる

以下の命令の 1 つを使用すれば、ブールビットロジック文字列を終了できます。

- = 割り付け
- R リセット
- S セット

1.1 ビット論理命令の概要

以下の命令を 1 つ使用すれば、論理演算の結果（RLO）を変更できます。:

- NOT RLO の否定
- SET RLO (=1) の設定
- CLR RLO (=0) のクリア
- SAVE RLO を BR レジスタに保存

他の命令は、正または負の信号遷移に応答します。

- FN 信号立ち下がり
- FP 信号立ち上がり

1.2 A AND

フォーマット

A <Bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D、T、C

説明

A は、アドレス指定されたビットの状態が"1"かどうかをチェックし、テスト結果と RLO の AND を求めます。

ステータスワードビットのチェック:

AND 命令を使用し、以下のアドレスを使用すれば、ステータスワードを直接チェックすることもできます。==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	x	x	x	1

例

STL Program	Relay Logic	
	Power rail	
A I 1.0	I 1.0 signal state 1	NO contact
A I 1.1	I 1.1 signal state 1	NC contact
= Q 4.0	Q 4.0 signal state 1	Coil
 Displays closed switch		

1.3 AN AND NOT

フォーマット

N <Bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D、T、C

説明

AN は、アドレス指定されたビットの状態が"0"かどうかをチェックし、テスト結果と RLO の論理積を求めます。

ステータスワードビットのチェック:

AND NOT 命令を使用して、以下のアドレスを使用すれば、ステータスワードを直接チェックすることもできます。==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	x	x	x	1

例

STL Program		Relay Logic	
		Power rail	
A	I 1.0	I 1.0 Signal state 0	NO contact 
AN	I 1.1	I 1.1 Signal state 1	NC contact 
=	Q 4.0	Q 4.0 Signal state 0	Coil 

1.4 O OR

フォーマット

O <Bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D、T、C

説明

O は、アドレス指定されたビットが"1"かどうかをチェックし、テスト結果と RLO の OR を求めます。

ステータスワードビットのチェック:

OR 命令を使用して、以下のアドレスを使用すれば、ステータスワードを直接チェックすることもできます。==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	x	1

例

STL Program	Relay Logic
O I 1.0	
O I 1.1	
= Q 4.0	

1.5 ON OR NOT

フォーマット

ON <Bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D、T、C

説明

ON は、アドレス指定されたビットの状態が"0"かどうかをチェックし、テスト結果と RLO の OR を求めます。

ステータスワードビットのチェック:

OR NOT 命令を使用して、以下のアドレスを使用すれば、ステータスワードを直接チェックすることもできます。==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-								

例

STL Program		Relay Logic	
		Power rail	
O	I 1.0	I 1.0 Signal state 0	NO contact
ON	I 1.1	I 1.1 Signal state 1	NC contact
=	Q 4.0	Q 4.0 Signal state 1	Coil

1.6 X 排他的 OR

フォーマット

X <Bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D、T、C

説明

X は、アドレス指定されたビットの状態が"1"かどうかをチェックし、テスト結果と RLO の排他的 OR を求めます。

排他的 OR ファンクションは複数回使用することもできます。チェックされたアドレスの不良数が"1"の場合、相互の論理演算の結果が"1"になります。

ステータスワードビットのチェック:

EXCLUSIVE OR 命令を使用して、以下のアドレスを使用すれば、ステータスワードを直接チェックすることもできます。==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	x	1

例

Statement List Program		Relay Logic	
		Power rail	
X	I 1.0	Contact I 1.0	
X	I 1.1	Contact I 1.1	
=	Q 4.0	Q 4.0 Coil	

1.7 XN 排他的OR NOT

フォーマット

XN <Bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D、T、C

説明

XN は、アドレス指定されたビットの状態が"0"かどうかをチェックし、テスト結果と RLO の排他的OR NOT を求めます。

ステータスワードビットのチェック:

EXCLUSIVE OR NOT 命令を使用して、以下のアドレスを使用すれば、ステータスワードを直接チェックすることもできます。==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	x	1

例

Statement List Program			Relay Logic
			Power rail
X	I 1.0	Contact I 1.0	
XN	I 1.1	Contact I 1.1	
=	Q 4.0	Q 4.0 Coil	

1.8 O AND の後に OR

フォーマット

O

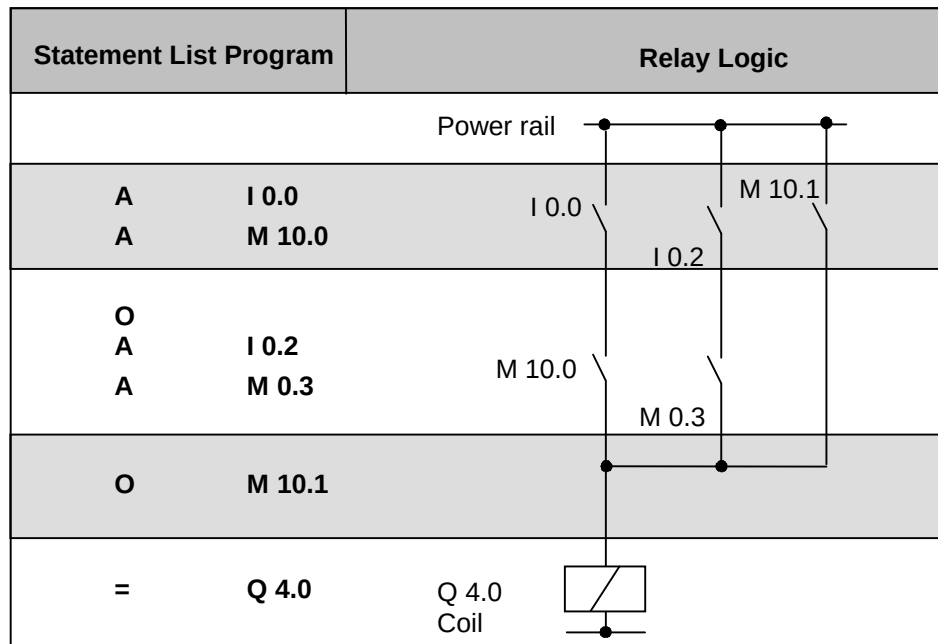
説明

O ファンクションは、AND 後に OR のルールに従って、AND ファンクションで論理 OR 命令を実行します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	x	1	-	x

例



1.9 A (AND/ネストを開く

フォーマット

A (

説明

A ((AND/ネストを開く) は、RLO と OR の各ビット、およびファンクションコードをネスティングスタックに保存します。最大 7 つのエントリまでネストできます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

例

Statement List Program		Relay Logic	
		Power rail	
A(	I 0.0	I 0.0	M 10.0
O	M 10.0		
)			
A(	I 0.2	I 0.2	M10.3
O	M 10.3		
)			
A	M 10.1	M 10.1	
=	Q 4.0	Q 4.0 Coil	

1.10 AN (AND NOT/ネストを開く

フォーマット

AN (

説明

AN ((AND NOT/ネストを開く) は、RLO と OR の各ビット、およびファンクションコードをネスティングスタックに保存します。最大 7 つのエントリまでネストできます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

1.11 O (OR/ネストを開く

フォーマット

O (

説明

O ((OR/ネストを開く) は、RLO と OR の各ビット、およびファンクションコードをネスティングスタックに保存します。最大 7 つのエントリまでネストできます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

1.12 ON (OR NOT/ネストを開く

フォーマット

ON (

説明

ON ((OR NOT/ネストを開く) は、RLO と OR の各ビット、およびファンクションコードをネスティングスタックに保存します。最大 7 つのエントリまでネストできます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

1.13 X (排他的 OR/ネストを開く

フォーマット

X (

説明

X ((排他的 OR/ネストを開く) は、RLO と OR の各ビット、およびファンクションコードをネスティングスタックに保存します。最大 7 つのエントリまでネストできます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

1.14 XN (排他的 OR NOT/ネストを開く

フォーマット

XN (

説明

XN ((排他的 OR NOT/ネストを開く) は、RLO と OR の各ビット、およびファンクションコードをネスティングスタックに保存します。最大 7 つのエントリまでネストできます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

1.15) ネストを閉じる

フォーマット

)

説明

) (ネストを閉じる) は、ファンクションコードに従って、ネスティングスタックからエントリを削除し、OR ビットを復元し、スタックエントリに含まれる RLO を現在の RLO とリンクしたあと、その結果を RLO に割り付けます。ファンクションコードが"AND"または"AND NOT"の場合は、OR ビットも含まれます。

括弧を開くステートメント

- U (AND/ ネストを開く
- UN (AND NOT/ネストを開く
- O (OR/ネストを開く
- ON (OR NOT/ネストを開く
- X (排他的 OR/ネストを開く
- XN (排他的 OR NOT/ネストを開く

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	x	1	x	1

例

Statement List Program		Relay Logic	
		Power rail	
A(	I 0.0	I 0.0	M 10.0
O	M 10.0		
)			
A(	I 0.2	I 0.2	M10.3
O	M 10.3		
)			
A	M 10.1	M 10.1	
=	Q 4.0	Q 4.0 Coil	

1.16 = 割り付け

フォーマット

<Bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D

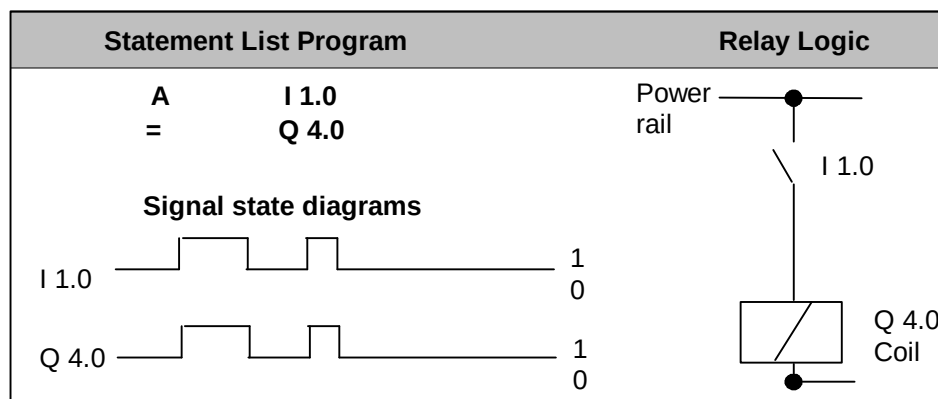
説明

= <Bit>は、MCR が 1 の場合、スイッチオンされたマスタコントロールリレーのアドレス指定ビットに RLO を書き込みます。MCR が 0 の場合、RLO の代わりに値 0 がアドレス指定ビットに書き込まれます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	-	0

例



1.17 R リセット

フォーマット

R <bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D

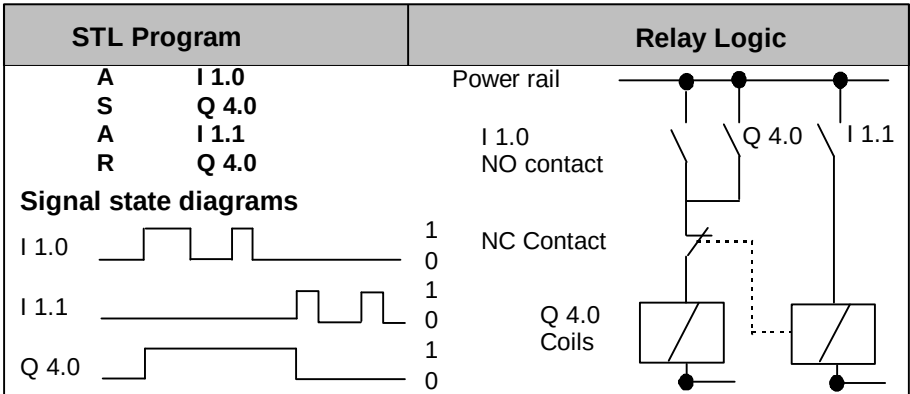
説明

R（ビットのリセット）は、RLO が 1 で、マスタコントロールリレーMCR が 1 の場合、アドレス指定ビットに「0」を配置します。
MCR が 0 の場合、アドレス指定ビットは変更されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	-	0

例



1.18 S セット

フォーマット

S <bit>

アドレス	データタイプ	メモリ領域
<Bit>	BOOL	I、Q、M、L、D

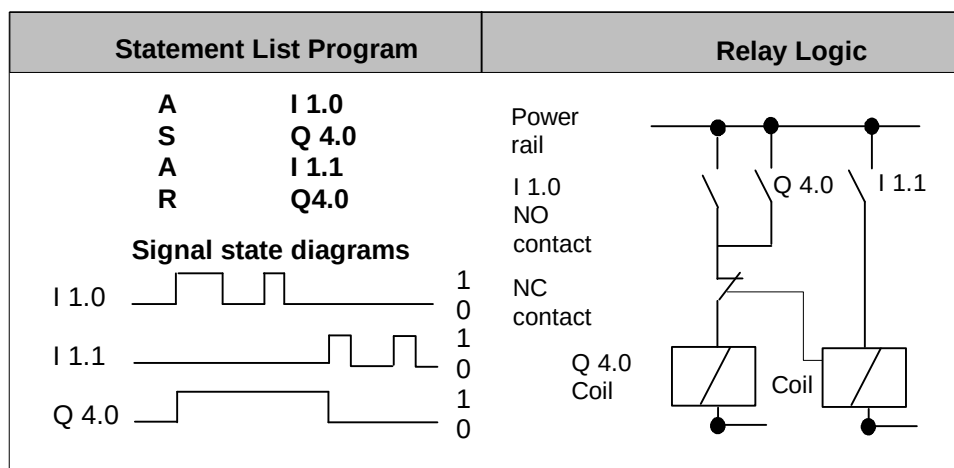
命令の説明

S（ビットのセット）は、RLO が 1 で、スイッチオンされたマスタコントロールリレーMCR が 1 の場合、アドレス指定ビットに「1」を配置します。MCR が 0 の場合、アドレス指定ビットは変更されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	-	0

例



1.19 NOT RLO の否定

フォーマット

NOT

説明

NOT は、RLO を否定します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	-	1	x	-

1.20 SET RLO (=1) の設定

フォーマット

SET

説明

SET は、RLO の信号状態を"1"に設定します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	1	0

例

STL Program	Signal State	Result of Logic Operation (RLO)
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 RLO (=0) のクリア

フォーマット

CLR

説明

CLR は、RLO を信号状態"0"に設定します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	0	0	0

例

Statement List	Signal State	Result of Logic Operation (RLO)
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 RLO を BR レジスタに保存

フォーマット

SAVE

命令の説明

SAVE は、BR ビットに RLO を保存します。最初のチェックビット/FC はリセットされません。このため、BR ビットの状態は、次のネットワークの AND 論理演算に含まれます。

BR ビットの **SAVE** とこれ以降の BR ビットの問い合わせを同じブロックまたは 2 次ブロックで使用することはお勧めできません。これは、この 2 つの操作の間に数多くの命令があると、BR ビットが変更される恐れがあるからです。ブロックを終了するときには、その前に **SAVE** 命令を使用してください。これは、この操作をすると、ENO 出力 (= BR ビット) が、RLO ビットの値に設定されるため、ブロックのエラー処理を追加できるからです。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	x	-	-	-	-	-	-	-	-

1.23 FN 信号立ち下がり

フォーマット

FN <Bit>

アドレス	データタイプ	メモリ領域	説明
<Bit>	BOOL	I、Q、M、L、D	エッジフラグ。RLO の前の信号状態を保存する。

説明

FN <Bit>（RLO の立ち下がり）は、RLO が"1"から"0"になると信号立ち下がりを検出し、これにより RLO は 1 になります。

各プログラムスキャンサイクルで、RLO ビットの信号状態を前のサイクルの RLO ビットの信号状態と比較して、状態に変化があったかどうかを確認します。前の RLO の状態は、比較を行えるように、エッジフラグアドレス（<Bit>）に保存する必要があります。現在の RLO"1"（信号立ち下がり検出）と前の RLO"1"に差がある場合、RLO ビットはこの命令の後で"1"になります。

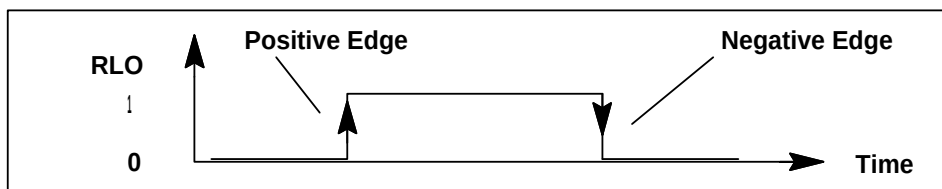
注記

モニタするビットがプロセスイメージ内に指定されていれば、この命令に問題はありません。これは、ブロックのローカルデータが有効になるのが、ブロックのランタイム時だけだからです。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	x	1

定義



例

プログラマブルロジックコントローラが、接点 I 1.0 で信号立ち下がりを検出すると、その OB1 スキャンサイクルで Q 4.0 のコイルが起動します。

Statement List		Signal State Diagram									
A	I 1.0	I 1.0									
FN	M 1.0	M 1.0									
=	Q 4.0	Q 4.0									
OB1 Scan Cycle No:			1	2	3	4	5	6	7	8	9

1.24 FP 信号立ち上がり

フォーマット

FP <Bit>

アドレス	データタイプ	メモリ領域	説明
<Bit>	BOOL	I、Q、M、L、D	エッジフラグ。RLO の前の信号状態を保存する。

説明

FP <Bit>（RLO の立ち上がり）は、RLO が"0"から"1"になると信号立ち上りを検出し、これにより RLO は 1 になります。

各プログラムスキャンサイクルで、RLO ビットの信号状態を前のサイクルの RLO ビットの信号状態と比較して、状態に変化があったかどうかを確認します。前の RLO の状態は、比較を行えるように、エッジフラグアドレス（<Bit>）に保存する必要があります。現在の RLO"0"（信号立ち下がり検出）と前の RLO"0"に差がある場合、RLO ビットはこの命令の後に"1"になります。

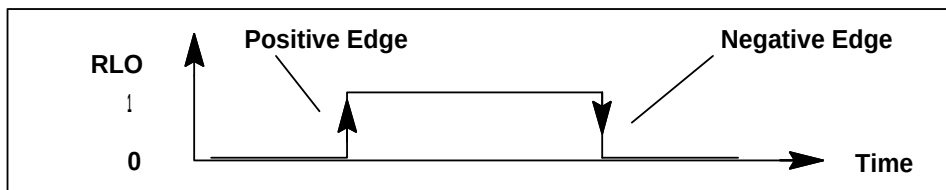
注記

モニタするビットがプロセスイメージ内に指定されていれば、この命令に問題はありません。これは、ブロックのローカルデータが有効になるのが、ブロックのランタイム時だけだからです。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容	-	-	-	-	-	0	x	x	1

定義



例

プログラマブルロジックコントローラが接点 I 1.0 で信号立ち上がりを検出すると、その OB1 スキャンサイクルで Q 4.0 のコイルが起動します。

Statement List		Signal State Diagram									
A	I 1.0	I 1.0									1 0
FP	M 1.0	M 1.0									1 0
=	Q 4.0	Q 4.0									1 0
OB1 Scan Cycle No:			1	2	3	4	5	6	7	8	9

2 比較命令

2.1 比較命令の概要

説明

ACCU1 と ACCU2 は、選択した比較のタイプに応じて比較されます。以下の比較タイプがあります。

== ACCU1 は ACCU2 と等しい。
<> ACCU1 は ACCU2 と等しくない。
> ACCU1 は ACCU2 より大きい。
< ACCU1 は ACCU2 より小さい。
>= ACCU1 は ACCU2 以上である。
<= ACCU1 は ACCU2 以下である。

比較結果が true の場合、ファンクションの RLO は"1"になります。ステータスワードビット CC 1 と CC 0 は、"小さい"、"等しい"、または"大きい"の各関係を示します。

以下のファンクションを実行する比較命令があります。

- ?I 整数（16 ビット）の比較
- ?D 倍長整数（32 ビット）の比較
- ?R 浮動小数点数（32 ビット）の比較

2.2 ?I 整数 (16 ビット) の比較

フォーマット

==I、<>I、>I、<I、>=I、<=I

命令の説明

"整数 (16 ビット) の比較"命令は、ACCU 2-L の内容を ACCU 1-L の内容と比較します。ACCU 2-L と ACCU 1-L の内容は、16 ビット整数と解釈されます。比較結果は、RLO および対応するステータスワードビットの設定で示されます。RLO が 1 の場合は比較結果は true、0 の場合は false になります。ステータスワードビット CC 1 と CC 0 は、“小さい”、“等しい”、または“大きい”の各関係を示します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	x	x	0	-	0	x	x	1

RLO 値

実行する比較命令	RLO の値 ACCU 2 > ACCU 1	RLO の値 ACCU 2 = ACCU 1	RLO の値 ACCU 2 < ACCU 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

例

STL	説明
L MW10	//MW10 (16 ビット整数) の内容をロードする。
L IW24	//IW24 (16 ビット整数) の内容をロードする。
>I	//ACCU 2-L (MW10) が ACCU 1 (IW24) よりも大きいかどうかを比較する。
= M 2.0	//MW10 > IW24 の場合、RLO は 1 になる。

2.3 ? D 倍長整数 (32 ビット) の比較

フォーマット

==D、<>D、>D、<D、>=D、<=D

命令の説明

"倍長整数 (32 ビット) の比較"命令は、ACCU 2 の内容を ACCU 1 の内容と比較します。ACCU 2 と ACCU 1 の内容は、32 ビット整数と解釈されます。比較結果は、RLO および対応するステータスワードビットの設定で示されます。RLO が 1 の場合は比較結果は true、0 の場合は false になります。ステータスワードビット CC 1 と CC 0 は、“小さい”、“等しい”、または“大きい”の各関係を示します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	0	-	0	x	x	1

RLO 値

実行する比較命令	RLO の値 ACCU 2 > ACCU 1	RLO の値 ACCU 2 = ACCU 1	RLO の値 ACCU 2 < ACCU 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

例

STL	説明
L MD10	//MD10 (倍長整数、32 ビット) の内容をロードする。
L ID24	//ID24 (倍長整数、32 ビット) の内容をロードする。
>D	//ACCU 2 (MD10) が ACCU 1 (ID24) より大きいかどうか比較する。
= M 2.0	//MD10 > ID24 の場合、RLO は 1 になる。

2.4 ? R 浮動小数点数 (32 ビット) の比較

フォーマット

==R、<>R、>R、<R、>=R、<=R

命令の説明

浮動小数点数の比較 (32 ビット、IEEE 754) 命令は、ACCU 2 の内容を ACCU 1 の内容と比較します。ACCU 1 と ACCU 2 の内容は、浮動小数点数として解釈されます (32 ビット、IEEE 754)。比較結果は、RLO および対応するステータスワードビットの設定で示されます。RLO が 1 の場合は比較結果は true、0 の場合は false になります。ステータスワードビット CC 1 と CC 0 は、“小さい”、“等しい”、または“大きい”の各関係を示します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	x	x	x	x	0	x	x	1

RLO 値

実行する比較命令	RLO の値 ACCU 2 > ACCU 1	RLO の値 ACCU 2 = ACCU 1	RLO の値 ACCU 2 < ACCU 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

例

STL	説明
L MD10	//MD10 (浮動小数点数) の内容をロードする。
L 1.359E+02	//定数 1.359E+02 をロードする。
>R	//ACCU 2 (MD10) が ACCU 1 (1.359-E+02) よりも大きいかどうかを比較する。
= M 2.0	//MD10 > 1.359E+02 の場合、RLO は 1 になる。

3 変換命令

3.1 変換命令の概要

説明

以下の命令を使用すれば、2 進数 10 進数と整数を他のタイプの数に変換できます。

- BTI BCD から整数（16 ビット）への変換
- ITB 整数（16 ビット）から BCD への変換
- BTD BCD から整数（32 ビット）への変換
- ITD 整数（16 ビット）から倍長整数（32 ビット）への変換
- DTB 倍長整数（32 ビット）から BCD への変換
- DTR 倍長整数（32 ビット）から浮動小数点数への変換（32 ビット、IEEE 754）

以下の命令を使用すれば、整数の補数を生成したり、浮動小数点数の符号を反転したりできます。

- INVI 整数の 1 の補数（16 ビット）
- INVD 倍長整数の 1 の補数（32 ビット）
- NEGI 整数の 2 の補数（16 ビット）
- NEGD 倍長整数の 2 の補数（32 ビット）
- NEGR 浮動小数点数（32 ビット、IEEE 754）の否定

アキュムレータ 1 で、以下のビットシーケンス変更命令を使用すれば、アキュムレータ 1 の下位ワードのバイトの順序、またはアキュムレータ全体のバイトの順序を反転できます。

- CAW ACCU 1-L のバイトシーケンス（16 ビット）を変更
- CAD ACCU 1 のバイトシーケンス（32 ビット）を変更

以下の命令を使用すれば、アキュムレータ 1 の 32 ビット浮動小数点数を 32 ビット整数（倍長整数）に変換できます。個々の命令で、丸め方法が異なります。

- RND 丸め
- TRUNC 切り捨て
- RND+ 倍長整数の切り上げ
- RND 倍長整数の切り下げ

3.2 BTI BCD から整数 (16 ビット) への変換

フォーマット

BTI

説明

BTI (3 桁の BCD 数を 10 進数から 2 進数へ変換) は、ACCU 1-L の内容を 3 桁の 2 進化 10 進数 (BCD) と解釈し、16 ビット倍長整数に変換します。この結果は、アキュムレータ 1 の下位ワードに保存されます。アキュムレータ 1 の上位ワードとアキュムレータ 2 は変更されません。

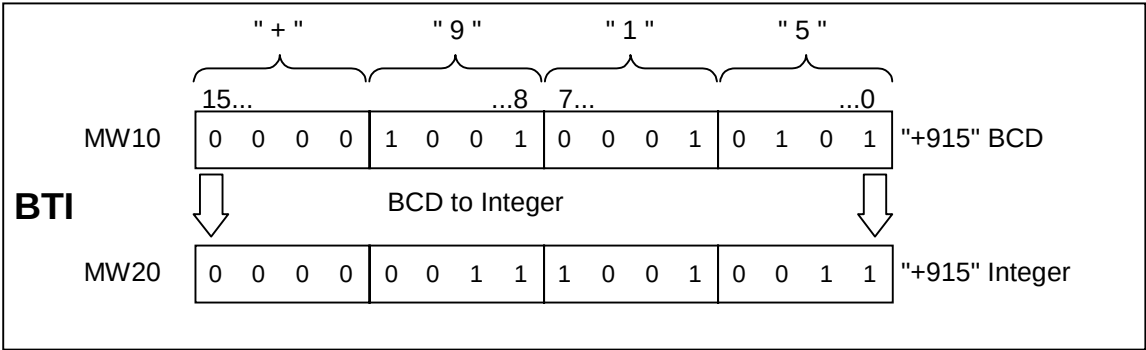
ACCU 1-L の BCD 数: BCD 数の許容値範囲は"-999"~"+999"です。ビット 0~11 は値として、ビット 15 は BCD の符号 (0 = 正、1 = 負) として解釈されます。ビット 12~14 は、この変換には使用されません。10 進数字 (4 ビット) が無効な範囲 (10~15) にある場合、変換の実行中に BCDF エラーが発生します。通常、CPU は STOP に移行します。ただし、この同期プログラミングエラーには、OB121 をプログラミングして別のエラー応答を設定することができます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MW10	//BCD 数を ACCU 1-L にロードする。
BTI	//BCD を整数に変換し、その結果を ACCU 1-L に保存する。
T MW20	//結果 (整数) を MW20 に転送する。



3.3 ITB 整数 (16 ビット) から BCD への変換

フォーマット

ITB

説明

ITB (16 ビット数を 2 進数から 10 進数へ変換) は、ACCU 1-L の内容を 16 ビット整数と解釈し、3 桁の 2 進化 10 進数 (BCD) に変換します。この結果は、アキュムレータ 1 の下位ワードに保存されます。ビット 0~11 には、BCD 数の値が格納されます。ビット 12~15 は、BCD の符号 (0000 = 正、1111 = 負) の状態に設定されます。アキュムレータ 1 の上位ワードとアキュムレータ 2 は変更されません。

BCD の値の有効範囲は "-999" ~ "+999" です。この範囲内にない場合、ステータスビット OV および OS が 1 に設定されます。

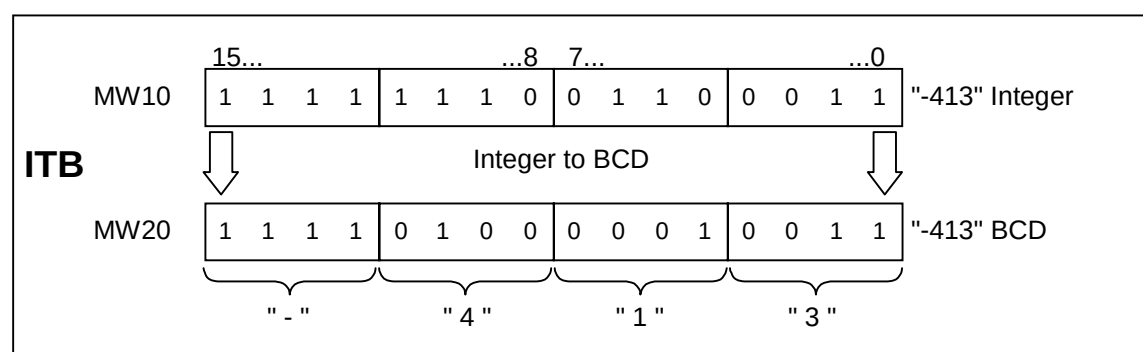
この命令は、RLO に関係なく実行され、これにより RLO が変更されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	X	X	-	-	-	-

例

STL	説明
L MW10	//整数を ACCU 1-L にロードする。
ITB	//整数を BCD (16 ビット) に変換し、その結果を ACCU 1-L に保存する。
T MW20	//結果 (BCD 数) を MW20 に転送する。



3.4 BT D BCD から整数 (32 ビット) への変換

フォーマット

BT D

説明

BT D (7 桁の BCD を 10 進数から 2 進数へ変換) は、ACCU 1 の内容を 7 桁の 2 進化 10 進数 (BCD) として解釈し、32 ビット倍長整数に変換します。この結果は、アキュムレータ 1 に保存されます。アキュムレータ 2 は変更されません。

ACCU 1BCD:BCD の値の有効範囲は"-9,999,999"~"+9,999,999"です。ビット 0~27 は値として、ビット 31 は BCD の符号 (0 = 正、1 = 負) として解釈されます。ビット 28~30 は、この変換には使用されません。

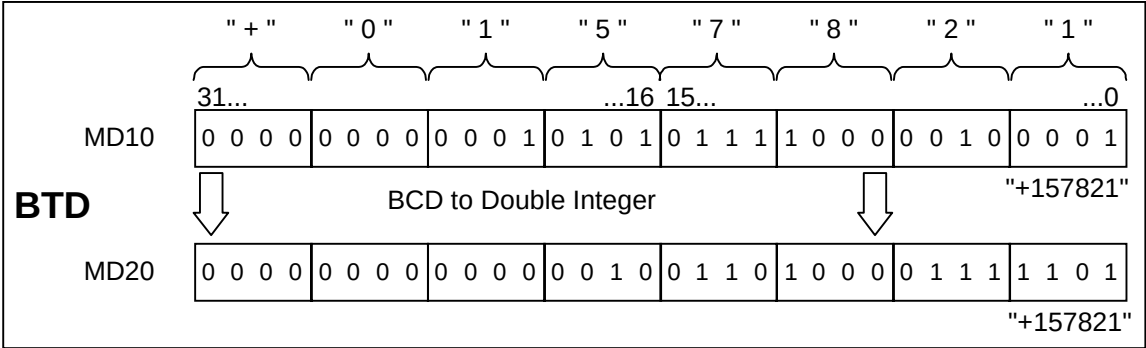
10 進数字 (4 ビット) が無効な範囲 (10~15) にある場合、変換の実行中に BCDF エラーが発生します。通常、CPU は STOP に移行します。ただし、この同期プログラミングエラーには、OB121 をプログラミングして別のエラー応答を設定することができます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MD10	//BCD 数を ACCU 1 にロードする。
BT D	//BCD を整数に変換し、その結果を ACCU 1 に保存する。
T MD20	//結果 (倍長整数) を MD20 に転送する。



3.5 ITD 整数（16 ビット）から倍長整数（32 ビット）への変換

フォーマット

ITD

説明

ITD（16 ビット整数を 32 ビット整数へ変換）は、ACCU 1-L 内容を 16 ビット整数として解釈し、32 ビット倍長整数に変換します。この結果は、アキュムレータ 1 に保存されます。アキュムレータ 2 は変更されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MW12	//整数を ACCU 1 にロードする。
ITD	//整数（16 ビット）を倍長整数（32 ビット）に変換し、その結果を ACCU 1 に保存する。
T MD20	//結果（倍長整数）を MD20 に転送する。

例: MW12 = "-10"（整数、16 ビット）

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
ITD 命令実行前の値	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
ITD 命令実行後の値	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 または 1、変換にはビットは使用されない)							

3.6 DTB 倍長整数 (32 ビット) から BCD への変換

フォーマット

DTB

説明

DTB (32 ビット整数を 2 進数から 10 進数へ変換) は、ACCU 1 の内容を 32 ビット整数と解釈し、7 桁の 2 進化 10 進数 (BCD) に変換します。この結果はアキュムレータ 1 に保存されます。ビット 0 ~ 27 には、BCD 数の値が格納されます。ビット 28 ~ 31 は BCD の符号 (0000 = 正、1111 = 負) の状態に設定されます。アキュムレータ 2 は変更されません。

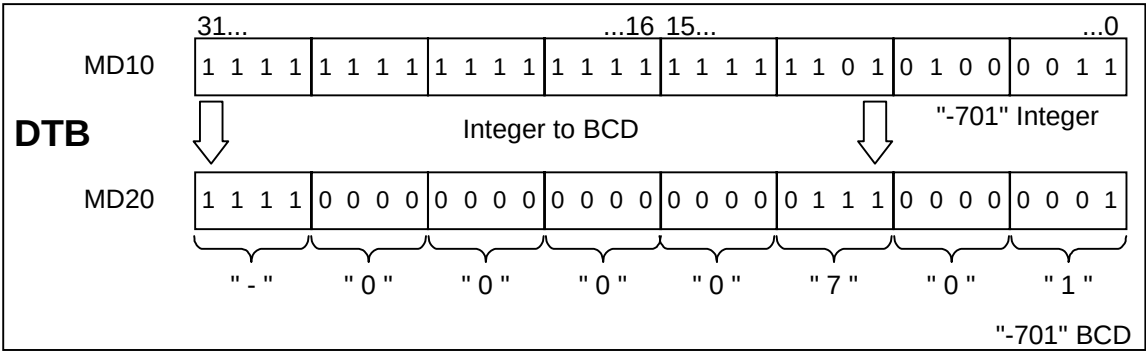
BCD の値の有効範囲は、"-9,999,999" ~ "+9,999,999" です。この範囲内にない場合、ステータスビット OV および OS が 1 に設定されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	X	X	-	-	-	-

例

STL	説明
L MD10	//32 ビット整数を ACCU 1 にロードする。
DTB	//整数 (32 ビット) を BCD に変換し、その結果を ACCU 1 に保存する。
T MD20	//結果 (BCD 数) を MD20 に転送する。



3.7 DTR 倍長整数 (32 ビット) から浮動小数点数への変換 (32 ビット、IEEE 754)

フォーマット

DTR

説明

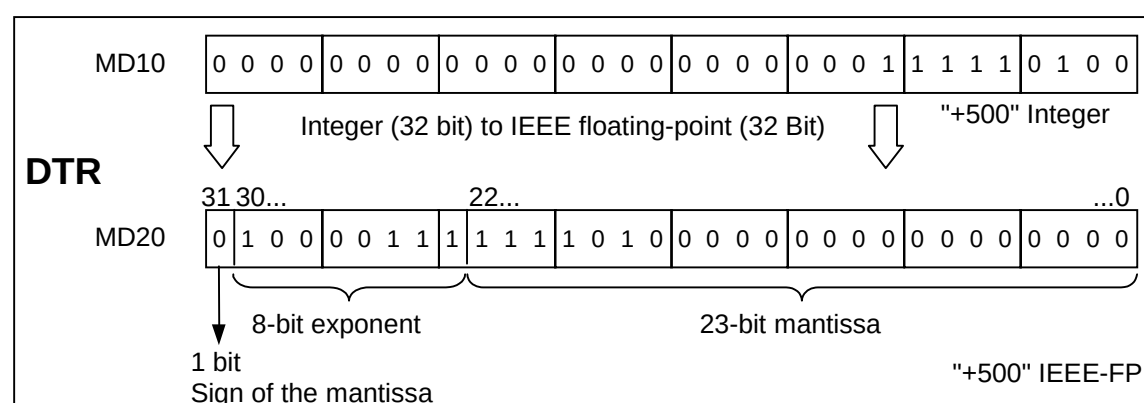
DTR (32 ビット整数を 32 ビット IEEE 浮動小数点数へ変換) は、ACCU 1 の内容を 32 ビット倍長整数として解釈し、32 ビット IEEE 浮動小数点数に変換します。この結果は、必要に応じて丸めが行われます (32 ビット整数は、32 ビット浮動小数点数よりも正確度が高い)。この結果は、アキュムレータ 1 に保存されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MD10	//32 ビット整数を ACCU 1 にロードする。
DTR	//倍長整数を浮動小数点数 (32 ビット IEEE FP) へ変換し、その結果を ACCU 1 に保存する。
T MD20	//結果 (BCD 数) を MD20 に転送する。



3.8 INVI 整数の1の補数 (16 ビット)

フォーマット

INVI

説明

INVI (整数の1の補数) は、16 ビット値の1の補数を ACCU 1-L に生成します。1の補数が生成されると、値はビットごとに反転し、1は0に、0は1になります。この結果は、アキュムレータ1の下位ワードに保存されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L IW8	//値を ACCU 1-L にロードする。
INVI	//1の補数 (16 ビット) を生成する。
T MW10	//結果を MW10 に転送する。

目次	ACCU1-L			
ビット	15 ..	..	..	... 0
INVI 命令実行前の値	0110	0011	1010	1110
INVI 命令実行後の値	1001	1100	0101	0001

3.9 INVD 倍長整数の1の補数 (32 ビット)

フォーマット

INVD

説明

INVD (倍長整数の1の補数) は、32 ビット値の1の補数を ACCU 1 に生成します。1の補数が生成されると、値はビットごとに反転し、1は0に、0は1になります。この結果は、アキュムレータ 1 に保存されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L ID8	//値を ACCU 1 にロードする。
INVD	//1 の補数 (32 ビット) を生成する。
T MD10	//結果を MD10 に転送する。

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
INVD 命令実行前の値	0110	1111	1000	1100	0110	0011	1010	1110
INVD 命令実行後の値	1001	0000	0111	0011	1001	1100	0101	0001

3.10 NEGI 整数の2の補数 (16 ビット)

フォーマット

NEGI

説明

NEGI (整数の2の補数) は、16 ビット値の2の補数を ACCU 1-L に生成します。2の補数が生成されると、値はビットごとに反転し、1は0に、0は1になり、"1"が追加されます。この結果は、アキュムレータ 1 の下位ワードに保存されます。2の補数の命令は、-1 による乗算と同じです。ステータスビット CC 1、CC 0、OS、および OV は、演算結果のファンクションとして設定されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	X	-	-	-	-

ステータスワードの生成	CC 1	CC 0	OV	OS
結果 = 0	0	0	0	-
-32768 <= 結果 <= -1	0	1	0	-
32767 >= 結果 >= 1	1	0	0	-
結果 = 2768	0	1	1	1

例

STL	説明
L IW8	//値を ACCU 1-L にロードする。
NEGI	//2の補数 (16 ビット) を生成する。
T MW10	//結果を MW10 に転送する。

目次	ACCU1-L			
ビット	15 ...	..	..	... 0
NEGI 命令実行前の値	0101	1101	0011	1000
NEGI 命令実行後の値	1010	0010	1100	1000

3.11 NEGD 倍長整数の2の補数 (32 ビット)

フォーマット

NEGD

説明

NEGD (2の補数の倍長整数) は、32 ビット値の2の補数を ACCU 1 に生成します。2の補数が生成されると、値はビットごとに反転し、1は0に、0は1になり、"1"が追加されます。この結果は、アキュムレータ 1 に保存されます。2の補数の命令は、-1 による乗算と同じです。命令は RLO に関係なく、また RLO に影響を与えずに実行されます。ステータスビット CC 1、CC 0、OS、および OV は、演算結果のファンクションとして設定されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

ステータスワードの生成	CC 1	CC 0	OV	OS
結果 = 0	0	0	0	-
-2.147.483.647 <= 結果 <= -1	0	1	0	—
2.147.483.647 >= 結果 >= 1	1	0	0	-
結果 = -2 147 483 648	0	1	1	1

例

STL	説明
L ID8	//値を ACCU 1 にロードする。
NEGD	//2の補数 (32 ビット) を生成する。
T MD10	//結果を MD10 に転送する。

目次	ACCU1-H				ACCU1-L			
ビット	31...	...	...	...16	15...	...	...	...0
NEGD 命令実行前の値	0101	1111	0110	0100	0101	1101	0011	1000
NEGD 命令実行後の値	1010	0000	1001	1011	1010	0010	1100	1000
	(X = 0 または 1、変換にはビットは使用されない)							

3.12 NEGR 浮動小数点数 (32 ビット、IEEE 754) の否定

フォーマット

NEGR

命令の説明

NEGR (32 ビット IEEE 浮動小数点数の否定) は、ACCU 1 に格納された浮動小数点数 (32 ビット、IEEE 754) を否定します。この命令は、ACCU 1 のビット 31 の状態 (仮数の符号) を反転します。この結果は、アキュムレータ 1 に保存されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L ID8	//値を ACCU 1 にロードする (例: ID 8 = 1.5E+02)。
NEGR	//浮動小数点数 (32 ビット、IEEE 754) を否定し、その結果を ACCU 1 に保存する。
T MD10	//結果を MD10 に転送する (例: result = -1.5E+02)。

3.13 CAW ACCU 1-L のバイトシーケンス (16 ビット) を変更

フォーマット

CAW

説明

CAW は、ACCU 1-L のバイトシーケンスを反転します。この結果は、アキュムレータ 1 の下位ワードに保存されます。アキュムレータ 1 の上位ワードとアキュムレータ 2 は変更されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MW10	//MW10 の値を ACCU 1 にロードする。
CAW	//ACCU 1-L のバイトシーケンスを反転する。
T MW20	//結果を MW20 に転送する。

目次	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
CAW 命令実行前の値	A	B	C	D
CAW 命令実行後の値	A	B	D	C

3.14 CAD ACCU 1 のバイトシーケンス (32 ビット) を変更

フォーマット

CAD

説明

CAD は、ACCU 1 のバイトシーケンスを反転します。この結果は、アキュムレータ 1 に保存されます。アキュムレータ 2 は変更されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MD10	//MD10 の値を ACCU 1 にロードする。
CAD	//ACCU 1 のバイトシーケンスを反転する。
T MD20	//結果を MD20 に転送する。

目次	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
CAD 命令実行前の値	A	B	C	D
CAD 命令実行後の値	D	C	B	A

3.15 RND 丸め

フォーマット

RND

説明

RND（32 ビット IEEE 浮動小数点を 32 ビット整数へ変換）は、ACCU 1 の内容を、32 ビット IEEE 浮動小数点数（32 ビット、IEEE 754）として解釈します。32 ビット IEEE 浮動小数点数は 32 ビット整数（倍長整数）に変換され、最も近い整数に丸められます。変換後の数字の端数部分が偶数と奇数の間にある場合、自動的に偶数が選択されます。この範囲内にない場合、ステータスビット OV および OS が 1 に設定されます。この結果は、アキュムレータ 1 に保存されます。

障害が発生すると（NaN、または 32 ビット整数で表現できない浮動小数点数を使用）、変換は実行されず、オーバーフローが示されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	X	X	-	-	-	-

例

STL	説明
L MD10	//浮動小数点数を ACCU 1-L にロードする。
RND	//浮動小数点数（32 ビット、IEEE 754）を整数（32 ビット）に変換し、その結果を丸める。
T MD20	//結果（倍長整数）を MD20 に転送する。

変換前の値		変換後の値
MD10 = "100.5"	=> RND =>	MD20 = "+100"
MD10 = "-100.5"	=> RND =>	MD20 = "-100"

3.16 TRUNC 切り捨て

フォーマット

TRUNC

説明

TRUNC (32 ビット IEEE 浮動小数点数を 32 ビット整数に変換) は、ACCU 1 の内容を 32 ビット IEEE 浮動小数点数として解釈します。32 ビット IEEE 浮動小数点数を 32 ビット整数 (倍長整数) に変換します。変換された浮動小数点数の整数部分がこの命令の結果になります (IEEE 丸めモード"ゼロに丸め")。この範囲内にない場合、ステータスビット OV および OS が 1 に設定されます。この結果は、アキュムレータ 1 に保存されます。

障害が発生すると (NaN、または 32 ビット整数で表現できない浮動小数点数を使用)、変換は実行されず、オーバーフローが示されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	X	X	-	-	-	-

例

STL	説明
L MD10	//浮動小数点数を ACCU 1-L にロードする。
TRUNC	//浮動小数点数 (32 ビット、IEEE 754) を整数 (32 ビット) に変換し、その結果を丸める。 //結果を ACCU 1 に保存する。
T MD20	//結果 (倍長整数) を MD20 に転送する。

変換前の値		変換後の値
MD10 = "100.5"	=> TRUNC =>	MD20 = "+100"
MD10 = "-100.5"	=> TRUNC =>	MD20 = "-100"

3.17 RND+ 倍長整数の切り上げ

フォーマット

RND+

説明

RND+（32 ビット IEEE 浮動小数点を 32 ビット整数へ変換）は、ACCU 1 の内容を、32 ビット IEEE 浮動小数点数として解釈します。32 ビット IEEE 浮動小数点数は 32 ビット整数（倍長整数）に変換され、この結果は、変換された浮動小数点数以上で、最も近い整数に変換されます（IEEE 丸めモード "+infinity に丸め"）。数値が有効範囲にない場合、ステータスビット OV および OS は 1 に設定されます。この結果はアキュムレータ 1 に保存されます。

障害が発生すると（NaN、または 32 ビット整数で表現できない浮動小数点数を使用）、変換は実行されず、オーバーフローが示されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	X	X	-	-	-	-

例

STL	説明
L MD10	//浮動小数点数（32 ビット、IEEE 754）を ACCU 1-L にロードする。
RND+	//浮動小数点数（32 ビット、IEEE 754）を整数（32 ビット）に変換し、その結果を丸める。 //出力を ACCU 1 に保存する。
T MD20	//結果（倍長整数）を MD20 に転送する。

変換前の値		変換後の値
MD10 = "100.5"	=> RND+ =>	MD20 = "+101"
MD10 = "-100.5"	=> RND+ =>	MD20 = "-100"

3.18 RND- 倍長整数の切り下げ

フォーマット

RND-

説明

RND- (32 ビット IEEE 浮動小数点数を 32 ビット整数に変換) は、ACCU 1 の内容を、32 ビット IEEE 浮動小数点数として解釈します。32 ビット IEEE 浮動小数点数は 32 ビット整数 (倍長整数) に変換され、その結果は、変換された浮動小数点数以下で、最も近い整数に丸められます (IEEE 丸めモード "-infinity へ丸め")。この範囲内にない場合、ステータスビット OV および OS が 1 に設定されます。この結果は、アキュムレータ 1 に保存されます。

エラーが発生すると (NaN、または 32 ビット整数で表現できない浮動小数点数を使用)、変換は実行されず、オーバーフローが示されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	X	X	-	-	-	-

例

STL	説明
L MD10	//浮動小数点数を ACCU 1-L にロードする。
RND-	//浮動小数点数 (32 ビット、IEEE 754) を整数 (32 ビット) に変換し、その結果を丸める。 //結果を ACCU 1 に保存する。
T MD20	//結果 (倍長整数) を MD20 に転送する。

変換前の値		変換後の値
MD10 = "100.5"	=> RND- =>	MD20 = "+100"
MD10 = "-100.5"	=> RND- =>	MD20 = "-100"

4 カウンタ命令

4.1 カウンタ命令の概要

説明

カウンタは、カウントを実行する STEP 7 プログラム言語のファンクションエレメントです。カウンタは、CPU のメモリ内にカウンタ用に確保された領域を持っています。このメモリ領域では、カウンタごとに 16 ビットワードが 1 つ確保されています。ステートメントリスト命令では、256 個のカウンタが使用できます。使用している CPU で使うことのできるカウンタ数に関しては、CPU の技術データを参照してください。

カウンタ命令は、メモリ領域にアクセスできる唯一のファンクションです。

以下のカウンタ命令を使用すれば、この範囲内でカウント値を変更できます。

- FR カウンタの有効化（フリー）
- L 現在のカウンタ値を ACCU 1 へロード
- LC 現在のカウンタ値を BCD で ACCU 1 にロード
- R カウンタのリセット
- S カウンタプリセット値の設定
- CU カウントアップ
- CD カウントダウン

4.2 FR カウンタの有効化（フリー）

フォーマット

FR <counter>

アドレス	データタイプ	メモリ領域	説明
<counter>	COUNTER	C	カウンタ。範囲は CPU に より異なる。

説明

FR <counter>では、RLO が"0"から"1"になると、アドレス指定されているカウンタのカウント方向の設定および選択に使用するエッジ検出フラグがクリアされます。カウンタを設定したり、通常のカウントを行う場合は、カウンタの有効化は必要ありません。つまり、カウンタプリセット値の設定、カウントアップ、またはカウントダウンの RLO が常に 1 であっても、カウンタの有効化の後にこれらの命令が実行されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.0	//入力 I 2.0 の信号状態をチェックする。
FR C3	//RLO が 0 から 1 に変わると、カウンタ C3 が有効になる。

4.3 L 現在のカウンタ値を ACCU 1 へロード

フォーマット

L <counter>

アドレス	データタイプ	メモリ領域	説明
<counter>	COUNTER	C	カウンタ。範囲は CPU により異なる。

説明

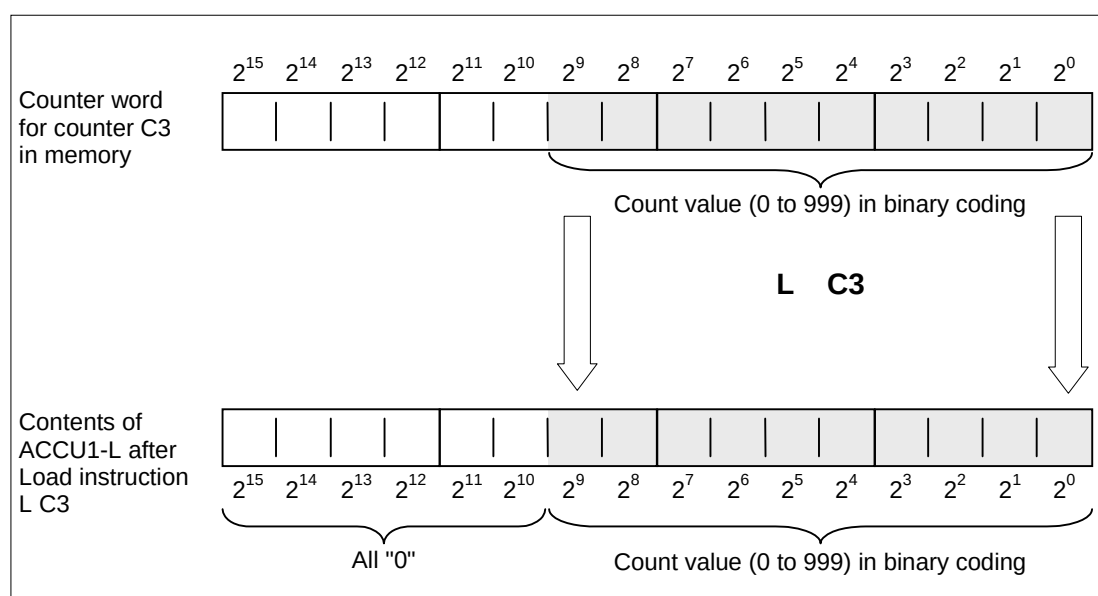
L <counter>は、ACCU 1 の内容を ACCU 2 に保存した後に、アドレス指定されたカウンタの現在のカウンタ値を整数として ACCU 1-L にロードします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L C3	//カウンタ C3 のカウンタ値をバイナリフォーマットで ACCU 1-L にロードする。



4.4 LC 現在のカウンタ値をBCDでACCU 1にロード

フォーマット

LC <counter>

アドレス	データタイプ	メモリ領域	説明
<counter>	COUNTER	C	カウンタ。範囲はCPUにより異なる。

説明

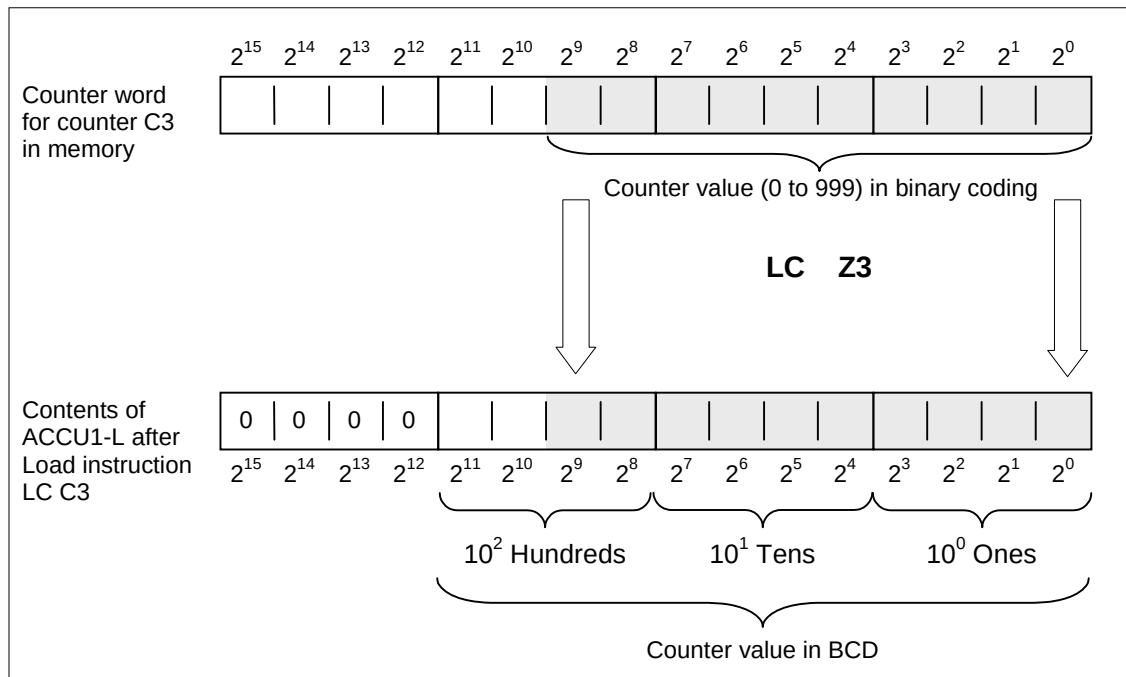
LC <counter>は、ACCU 1の内容をACCU 2へ保存した後に、アドレス指定されたカウンタのカウンタをBCDでCCU 1にロードします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
LC C3	//カウンタ C3 のカウンタ値を2 進化 10 進数フォーマットで ACCU 1-L にロードします。



4.5 R カウンタのリセット

フォーマット

R<counter>

アドレス	データタイプ	メモリ領域	説明
<counter>	COUNTER	C	プリセットされるカウンタ。範囲は CPU により異なる。

説明

R <counter>は、RLO が 1 のときに、アドレス指定されたカウンタに"0"をロードします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.3	//入力 I 2.3 の信号状態をチェックする。
R C3	//RLO が 0 から 1 になると、カウンタ C3 は値 0 にリセットされる。

4.6 S カウンタプリセット値の設定

フォーマット

S <counter>

アドレス	データタイプ	メモリ領域	説明
<counter>	COUNTER	C	プリセットされるカウンタ。範囲は CPU により異なる。

説明

S <counter>は、RLO が"0"から"1"になると、アドレス指定されたカウンタに、ACCU 1-L のカウントをロードします。ACCU 1 のカウントは、"0"～"999"の BCD でなければなりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.3	//入力 I 2.3 の信号状態をチェックする。
L C#3	//カウント値 3 を ACCU 1-L にロードする。
S C1	//RLO が 0 から 1 になると、カウンタ C1 がカウント値に設定される。

4.7 CU カウントアップ

フォーマット

CU <counter>

アドレス	データタイプ	メモリ領域	説明
<counter>	COUNTER	C	カウンタ。範囲は CPU に より異なる。

説明

CU <counter>では、RLO が"0"から"1"になった場合およびカウントが"999"未満の場合に、アドレス指定されたカウンタの値が1だけ増えます。カウントが上限値の"999"に達すると、それ以上増えません。RLO がさらに変化しても、カウントは行われず、オーバーフローOV ビットも設定されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.1	//信号立ち上がりがある場合、入力 I 2.1 で変化する。
CU C3	//RLO が 0 から 1 になると、カウンタ C3 が 1 だけ増える。

4.8 CD カウントダウン

フォーマット

CD <counter>

アドレス	データタイプ	メモリ領域	説明
<counter>	COUNTER	C	カウンタ。範囲は CPU により異なる。

説明

CD <counter>では、RLO が"0"から"1"に変化し、かつカウンタが"0"より大きい場合に、アドレス指定されたカウンタの値が 1 だけ減ります。カウンタが下限値の"0"に達すると、それ以上減りません。RLO がさらに変化しても、負数をカウントすることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
L C#14	//カウンタプリセット値
A I 0.1	//I 0.1 で信号立ち上がりを検出すると、カウンタをプリセットする。
S C1	//有効になると、プリセットされたカウンタ 1 をロードする。
A I 0.0	//I 0.0 の信号立ち上がりのたびに、カウンタが 1 だけ減る。
CD C1	//入力 I 0.0 の状態に応じて RLO が 0 から 1 になると、カウンタ C1 が 1 だけ減る。
AN C1	//C1 ビットを使用して 0 を検出する。
= Q 0.0	//カウンタ 1 の値が 0 の場合、Q 0.0 は 1 になる。

5 データブロック命令

5.1 データブロック命令の概要

説明

"データブロックを開く" (OPN) 命令を使用すれば、データブロックを共有データブロックまたはインスタンスデータブロックとして開くことができます。プログラムでは、共有データブロック 1 つとインスタンスデータブロック 1 つを同時に開くことができます。

以下のデータブロック命令を使用できます。

- OPN データブロックを開く
- CDB 共有 DB とインスタンス DB の交換
- L DBLG 共有 DB のサイズを ACCU 1 にロード
- L DBNO 共有 DB の数を ACCU 1 にロード
- L DILG インスタンス DB のサイズを ACCU 1 にロード
- L DINO インスタンス DB の数を ACCU 1 へロード

5.2 OPN データブロックを開く

フォーマット

OPN <data block>

アドレス	データブロックタイプ	ソースアドレス
<data block>	DB, DI	1~65535

命令の説明

OPN <data block>は、共有データブロックまたはインスタンスデータブロックとしてデータブロックを開きます。どちらのブロックも、1 度に関ける数は 1 つです。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
OPN DB10	//データブロック DB10 を共有データブロックとして開く。
L DBW35	//開いているデータブロックのデータワード 35 を ACCU 1-L にロードする。
T MW22	//ACCU 1-L の内容を MW22 に転送する。
OPN DI20	//データブロック DB20 をインスタンスデータブロックとして開く。
L DIB12	//開いているインスタンスデータブロックのデータバイト 12 を ACCU 1-L にロードする。
T DBB37	//ACCU 1-L の内容を、開いている共有データブロックのデータバイト 37 に転送する。

5.3 CDB 共有 DB とインスタンス DB の交換

フォーマット

CDB

命令の説明

CDB は、共有データブロックとインスタンスデータブロックの交換に使用します。この命令は、データブロックレジスタの交換を実行します。これにより、共有データブロックはインスタンスデータブロックに、インスタンスブロックは共有データブロックになります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

5.4 L DBLG 共有 DB のサイズを ACCU 1 にロード

フォーマット

L DBLG

命令の説明

L DBLG（共有データブロックのサイズをロード）は、ACCU 1 の内容を ACCU 2 に保存してから、共有データブロックのサイズを ACCU 1 にロードします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
OPN DB10	//データブロック DB10 を共有データブロックとして開く。
L DBLG	//共有データブロックの長さ (DB10 の長さ) をロードする。
L MD10	//データブロックが十分長い場合は比較値。
<D	
JC ERRO	//長さが MD10 の値より短い場合、ERRO ジャンプラベルへジャンプする。

5.5 L DBNO 共有 DB の数を ACCU 1 にロード

フォーマット

L DBNO

命令の説明

L DBNO（共有データブロックの数をロード）は、ACCU 1 の内容を ACCU 2 に保存してから、開いている共有データブロックの数を ACCU 1-L にロードします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

5.6 L DILG インスタンス DB のサイズを ACCU 1 にロード

フォーマット

L DILG

命令の説明

L DILG（インスタンスデータブロックのサイズをロード）は、ACCU 1 の内容を ACCU 2 に保存してから、インスタンスデータブロックのサイズを ACCU 1-L にロードします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
OPN D120	//データブロック DB20 をインスタンスデータブロックとして開く。
L DILG	//インスタンスデータブロックの長さ（DB20 の長さ）をロードする。
L MW10	//データブロックが十分長い場合は比較値。
<1	
JC	//長さが MW10 の値より短い場合、ERRO ジャンプラベルへジャンプする。

5.7 L DINO インスタンス DB の数を ACCU 1 へロード

フォーマット

L DINO

命令の説明

L DINO (インスタンスデータブロックの数をロード) は、ACCU 1 の内容を ACCU 2 に保存してから、開いているインスタンスデータブロックの数を ACCU 1 にロードします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

6 論理制御命令

6.1 論理制御命令の概要

説明

以下のジャンプ命令を使用すれば、ロジックのフローを制御できます。このため、プログラムは、そのリニアフローに割り込んで、異なるポイントでスキャンを再開できます。LOOP 命令を使用すれば、任意のプログラムセグメントを複数回呼び出すことができます。

ジャンプ命令やループ命令のアドレスはラベルです。ジャンプラベルは 4 文字で指定し、最初の 1 文字には英字を使用します。ジャンプラベルの後にコロン":"を指定する必要があります。また、各行内でプログラムステートメントの前にジャンプラベルを指定する必要があります。

注

S7-300 CPU プログラムでは、ジャンプ命令の場合、ジャンプ先は必ず (318-2 の場合は除く)、ブールロジック文字列の先頭を形成します。ロジック文字列に、ジャンプ先を指定しないでください。

以下のジャンプ命令を使用すれば、無条件で、プログラムの通常フローに割り込むことができます。

- JU 無条件ジャンプ
- JL ラベルへジャンプ

以下のジャンプ命令は、直前の命令ステートメントで生成された論理演算の結果 (RLO) に基づいて、プログラム内の論理のフローに割り込みます。

- JC RLO = 1 のときにジャンプ
- JCN RLO = 0 のときにジャンプ
- JCB RLO = 1 のときに BR に保存してジャンプ
- JNB RLO = 0 のときに BR に保存してジャンプ

以下のジャンプ命令は、ステータスワード内のビットの信号状態に基づいて、プログラム内のロジックのフローに割り込みます。

- JBI BR = 1 のときにジャンプ
- JNBI BR = 0 のときにジャンプ
- JO OV = 1 のときにジャンプ
- JOS OS = 1 のときにジャンプ

6.1 論理制御命令の概要

以下のジャンプ命令は、計算結果に基づいて、プログラム内のロジックのフローに割り込みます。

- JZ ゼロのときにジャンプ
- JN ゼロ以外のときにジャンプ
- JP プラスのときにジャンプ
- JM マイナスのときにジャンプ
- JPZ プラスまたはゼロのときにジャンプ
- JMZ マイナスまたはゼロのときにジャンプ
- JUO 比較不能のときにジャンプ

6.2 JU 無条件ジャンプ

フォーマット

JU <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JU <jump label> は、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは1つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768ワードまたは+32767ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1ワード、2ワード、または3ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明	
A	I 1.0	
A	I 1.2	
JC	DELE	//RLO が 1 のときに、ジャンプラベル DELE へジャンプする。
L	MB10	
INC	1	
T	MB10	
JU	FORW	//ジャンプラベル FORW へ無条件ジャンプする。
DELE: L	0	
T	MB10	
FORW: A	I 2.1	//ジャンプラベル FORW にジャンプした後、ここからプログラムスキャンが再開される。

6.3 JL ラベルへジャンプ

フォーマット

JL <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JL <jump label> (ジャンプリストによりジャンプ) は、複数のジャンプを設定できます。ジャンプターゲットリストには最大 255 のエントリを指定でき、**JL** 命令の次の行から開始し、JL アドレスで参照されるジャンプラベルの前の行で終了します。各ジャンプ先は、1 つの **JU** 命令で構成されます。ジャンプ先の数 (0~255) は、ACCU 1-L-L の値になります。

JL 命令は、ACCU の内容が **JL** 命令とジャンプラベルの間にあるジャンプ先の数よりも少なければ、**JU** 命令のうちの 1 つにジャンプします。ACCU 1-L-L=0 の場合は最初の **JU** 命令にジャンプし、ACCU 1-L-L=1 の場合は 2 番目の **JU** 命令にジャンプします (以下同様)。ジャンプ先の数が大きすぎる場合、**JL** 命令はジャンプ先リスト内で最後の **JU** 命令にジャンプした後、再び最初の命令にジャンプします。

ジャンプ先リストは、JL 命令のアドレスで参照されるジャンプラベルの後に **JU** 命令が続きます。ジャンプリスト内の他の命令はすべて不正です。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明		
	L	MB0	//ジャンプ先番号を ACCU 1-L-L にロードする。
	JL	LSTX	//ACCU 1-L-L > 3 のときのジャンプ先
	JU	SEG0	//ACCU 1-L-L = 0 のときのジャンプ先
	JU	SEG1	//ACCU 1-L-L = 1 のときのジャンプ先
	JU	COMM	//ACCU 1-L-L = 2 のときのジャンプ先
	JU	SEG3	//ACCU 1-L-L = 3 のときのジャンプ先
LSTX:	JU	COMM	
SEG0:	*		//有効な命令
	*		
	JU	COMM	
SEG1:	*		//有効な命令
	*		
	JU	COMM	
SEG3:	*		//有効な命令
	*		
	JU	COMM	
COMM:	*		
	*		

6.4 JC RLO = 1 のときにジャンプ

フォーマット

JC <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JC <jump label>は、論理演算の結果が1のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは1つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768ワードまたは+32767ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1ワード、2ワード、または3ワードのステートメント）により異なります。

論理演算の結果が0の場合、ジャンプは実行されません。RLOは1に設定され、次のステートメントからプログラムスキャンが実行されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	1	0

例

STL	説明	
A I 1.0		
A I 1.2		
JC JOVR	//RLO が 1 のとき、ジャンプラベル JOVR にジャンプする。	
L IW8	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。	
T MW22		
JOVR: A I 2.1	//ジャンプラベル JOVR へジャンプした後、ここからプログラムスキャンが再開される。	

6.5 JCN RLO = 0 のときにジャンプ

フォーマット

JCN <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JCN <jump label> は、論理演算の結果が 0 のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

論理演算の結果が 1 の場合、ジャンプは実行されません。次のステートメントからプログラムスキャンが実行されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	1	0

例

STL	説明
A I 1.0	
A I 1.2	
JCN JOVR	//RLO が 0 のとき、ジャンプラベル JOVR にジャンプする。
L IW8	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
T MW22	
JOVR: A I 2.1	//ジャンプラベル JOVR へジャンプした後、ここからプログラムスキャンが再開される。

6.6 JCB RLO = 1 のときに BR に保存してジャンプ

フォーマット

JCB <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JCB <jump label>は、論理演算の結果が 1 のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

論理演算の結果が 0 の場合、ジャンプは実行されません。RLO は 1 に設定され、次のステートメントからプログラムスキャンが実行されます。

RLO は、その状態に関係なく、JCB <jump label>命令の BR にコピーされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	x	-	-	-	-	0	1	1	0

例

STL	説明
A I 1.0	
A I 1.2	
JCB JOVR	//RLO が 1 のとき、ジャンプラベル JOVR にジャンプする。 //RLO ビットの内容は BR ビットにコピーされる。
L IW8	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
T MW22	
JOVR: A I 2.1	//ジャンプラベル JOVR へジャンプした後、ここからプログラムスキャンが再開される。

6.7 JNB RLO = 0 のときに BR に保存してジャンプ

フォーマット

JNB <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JNB <jump label>は、論理演算の結果が0のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは1つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768ワードまたは+32767ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1ワード、2ワード、または3ワードのステートメント）により異なります。

論理演算の結果が1の場合、ジャンプは実行されません。RLOは1に設定され、次のステートメントからプログラムスキャンが実行されます。

RLOは、その状態に関係なく、JNB <jump label>命令のBRにコピーされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	x	-	-	-	-	0	1	1	0

例

STL	説明
A I 1.0	
A I 1.2	
JNB JOVR	//RLO が 0 のとき、ジャンプラベル JOVR にジャンプする。 //RLO ビットの内容を BR ビットにコピーする。
L IW8	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
T MW22	
JOVR: A I 2.1	//ジャンプラベル JOVR へジャンプした後、ここからプログラムスキャンが再開される。

6.8 JBI BR = 1 のときにジャンプ

フォーマット

JBI <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JBI <jump label>は、ステータスビット BR が 1 のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプラベルは 4 文字で指定し、最初の 1 文字には英字を使用します。ジャンプラベルの後にコロン":"を指定する必要があります。また、各行内でプログラムステートメントの前にジャンプラベルを指定する必要があります。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	0	1	-	0

6.9 JNBI BR = 0 のときにジャンプ

フォーマット

JNBI <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JNBI <jump label>は、ステータスビット BR が 0 のとき、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

6.10 JO OV = 1 のときにジャンプ

フォーマット

JO <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JO <jump label>は、ステータスビット OV が 1 のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。数値演算命令を結合する場合、各数値演算命令の後にオーバーフローのチェックを行って、中間結果が有効範囲内にあるかどうかを確認するか、または、JOS 命令を使用します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明	
L	MW10	
L	3	
*I		//MW10 の内容に"3"を掛ける。
JO	OVER	//計算結果が最大範囲（OV=1）を超えるとジャンプする。
T	MW10	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
A	M 4.0	
R	M 4.0	
JU	NEXT	
OVER:	AN	M 4.0 //ジャンプラベル OVER にジャンプした後、ここからプログラムスキャンが再開される。
	S	M 4.0
NEXT:	NOP 0	//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

6.11 JOS OS = 1 のときにジャンプ

フォーマット

JOS <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JOS <jump label>は、ステータスビット OS が 1 のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	0	-	-	-	-

6.11 JOS OS = 1 のときにジャンプ

例

STL	説明	
	L	IW10
	L	MW12
	*I	
	L	DBW25
	+I	
	L	MW14
	-I	
	JOS	OVER //計算 OS=1 の実行中に 3 つの命令の //1 つでオーバーフローが発生するとジャンプする。(注記参照)
	T	MW16 //ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
	A	M 4.0
	R	M 4.0
	JU	NEXT
OVER:	AN	M 4.0 //ジャンプラベル OVER にジャンプした後、ここからプログラムスキャンが再開される。
	S	M 4.0
NEXT:	NOP 0	//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

注

このケースでは、JO 命令は使用しないでください。JO 命令は、オーバーフローが発生した場合に、直前の-I 命令のみをチェックします。

6.12 JZ ゼロのときにジャンプ

フォーマット

JZ <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JZ <jump label>（結果が0のときにジャンプ）は、ステータスビットが CC 1 = 0 および CC 0 = 0 のとき、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは1つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MW10	
SRW 1	
JZ ZERO	//シフトアウトしたビットが0の場合、ジャンプラベル ZERO へジャンプする。
L MW2	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
INC 1	
T MW2	
JU NEXT	
ZERO: L MW4	//ジャンプラベル ZERO にジャンプした後、ここからプログラムスキャンが再開される。
INC 1	
T MW4	
NEXT: NOP 0	//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

6.13 JN ゼロ以外のときにジャンプ

フォーマット

JN <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

ステータスビット CC 1 および CC 0 の示す結果が 0 より大きいか 0 未満 (CC 1=0/CC 0=1 または CC 1=1/CC 0=0) のとき、JN <jump label> (結果が 0 以外のときにジャンプ) は、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ (1 ワード、2 ワード、または 3 ワードのステートメント) により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明	
L	IW8	
L	MW12	
XOW		
JN	NOZE	//ACCU 1-L 内容が 0 でないときにジャンプする。
AN	M 4.0	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
S	M 4.0	
JU	NEXT	
NOZE: AN	M 4.1	//ジャンプラベル NOZE にジャンプした後、ここからプログラムスキャンが再開される。
S	M 4.1	
NEXT: NOP	0	//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

6.14 JP プラスのときにジャンプ

フォーマット

JP <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JP <jump label>（結果が0未満のときにジャンプ）は、ステータスビット CC 1 が1、CC 0 が0のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは1つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768ワードまたは+32767ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1ワード、2ワード、または3ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明	
L I W8		
L M W12		
-I		//IW8 の内容から MW12 の内容を減算する。
JP POS		//計算結果が0を超えると（つまり、ACCU 1 > 0 の場合）、ジャンプする。
AN M 4.0		//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
S M 4.0		
JU NEXT		
POS: AN M 4.1		//ジャンプラベル POS にジャンプした後、ここからプログラムスキャンが再開される。
S M 4.1		
NEXT: NOP 0		//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

6.15 JM マイナスのときにジャンプ

フォーマット

JM <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JM <jump label>（結果が0未満のときにジャンプ）は、ステータスビット CC 1 が 0、CC 0 が 1 のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは1つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明	
L	IW8	
L	MW12	
-I		//IW8 の内容から MW12 の内容を減算する。
JM	NEG	//計算結果が 0 より小さいとき（つまり、ACCU 1 の内容が 0 より小さい場合）、ジャンプする。
AN	M 4.0	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
S	M 4.0	
JU	NEXT	
NEG:	AN	M 4.1
	S	M 4.1
NEXT:	NOP 0	

6.16 JPZ プラスまたはゼロのときにジャンプ

フォーマット

JPZ <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

ステータスビット CC 1 および CC 0 の示す結果が 0 以上のとき（CC 1=0/CC 0=0 または CC 1=1/CC 0=0）、JPZ <jump label>（結果が 0 以上のときにジャンプ）は、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	

例

STL	説明	
L	IW8	
L	MW12	
-I		//IW8 の内容から MW12 の内容を減算する。
JPZ	REG0	//計算結果が 0 以上のとき（つまり、ACCU 1 >= 0 の場合）、ジャンプする。
AN	M 4.0	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
S	M 4.0	
JU	NEXT	
REG0:	AN	M 4.1
	S	M 4.1
NEXT:	NOP	0
		//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

6.17 JMZ マイナスまたはゼロのときにジャンプ

フォーマット

JMZ <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

ステータスビット CC 1 および CC 0 の示す結果が 0 以下（CC 1=0/CC 0=0 または CC 1=0/CC 0=1）のとき、JMZ <jump label>（結果が 0 以下のときにジャンプ）は、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L IW8	
L MW12	
-I	//IW8 の内容から MW12 の内容を減算する。
JMZ RGE0	//計算結果が 0 以下のとき（つまり、ACCU 1 の内容 0 以下の場合）、ジャンプする。
AN M 4.0	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
S M 4.0	
JU NEXT	
RGE0: AN M 4.1	//ジャンプラベル RGE0 にジャンプした後、ここからプログラムスキャンが再開される。
S M 4.1	
NEXT: NOP 0	//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

6.18 JUO 比較不能のときにジャンプ

フォーマット

JUO <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

JUO <jump label>は、ステータスビット CC 1 が 1、CC 0 が 1 のときに、リニアプログラムスキャンに割り込みを実行し、宛先へジャンプします。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスビット CC 1 が 1、CC 0 が 1 になるのは次の場合です。

- ゼロ除算が実行された場合
- 不正な命令が使用された場合
- 浮動小数点の比較結果が"比較不能"、つまりフォーマットが無効な場合

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

6.18 JUO 比較不能のときにジャンプ

例

STL	説明
L MD10	
L ID2	
/D	//MD10 の内容を ID2 の内容で除算する。
JUO ERRO	//ゼロによる除算が実行されると（つまり ID2 = 0 の場合）、ジャンプする。
T MD14	//ジャンプが実行されない場合、ここからプログラムスキャンが継続される。
A M 4.0	
R M 4.0	
JU NEXT	
ERRO: AN M 4.0	//ジャンプラベル ERRO にジャンプした後、ここからプログラムスキャンが再開される。
S M 4.0	
NEXT: NOP 0	//ジャンプラベル NEXT にジャンプした後、ここからプログラムスキャンが再開される。

6.19 LOOP ループ

フォーマット

LOOP <jump label>

アドレス	説明
<jump label>	ジャンプ先のシンボル名

説明

LOOP <jump label>（ACCU 1-L が 0 以外するとき、ACCU 1-L が 1 つ減り、ジャンプする）を使用すると、ループプログラミングが簡単になります。ACCU 1-L で、ループカウンタが加算されます。この命令は、指定されたジャンプ先へジャンプします。ACCU 1-L の内容が 0 でない限り、ジャンプは実行されます。そして、ジャンプ先でリニアプログラムスキャンを再開します。ジャンプ先は、ジャンプラベルで指定します。ジャンプは順方向と逆方向のどちらも可能です。また、ジャンプは 1 つのブロック内でのみ実行できるので、ジャンプ命令およびジャンプ先は同一のブロック内で指定します。ジャンプ先は、このブロック内でユニークでなければなりません。最大ジャンプ距離はプログラムコードの-32768 ワードまたは+32767 ワードです。ジャンプ可能なステートメントの最大数は、プログラムで使用されているステートメントの組み合わせ（1 ワード、2 ワード、または 3 ワードのステートメント）により異なります。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

ファクタ 5 の計算例

STL	説明		
NEXT:	L	L#1	// 整数 (32 ビット) を ACCU 1 にロードする。
	T	MD20	// ACCU 1 の内容を MD20 に転送する (初期化)。
	L	5	// ループサイクルの数を ACCU 1-L にロードする。
	T	MW10	// ジャンプレベル=ループ開始/ ACCU 1-L をループカウンタへ転送。
	L	MD20	
	*	D	// MD20 の現在の内容に、MB10 の現在の内容を乗算する。
	T	MD20	// 計算結果を MD20 に転送する。
	L	MW10	// ループカウンタの内容を ACCU 1 にロードする。
	LOOP	NEXT	// ACCU 1-L > 0 の場合、ACCU 1 の内容を 1 つ減らし、 // NEXT ジャンプレベルへジャンプする。
	L	MW24	// ループが終了すると、ここからプログラムスキャンが再開される。
L	200		
>I			

7 整数演算命令

7.1 整数値演算命令の概要

説明

数値演算は、アキュムレータ 1 とアキュムレータ 2 の内容を結合します。アキュムレータが 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

アキュムレータが 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の元の内容は変わりません。

整数演算では、**2 つの整数**（16 ビットおよび 32 ビット）を使用して次の演算を実行することができます。

- +I ACCU 1 と ACCU 2 の加算（16 ビット整数）
- -I ACCU 2 から ACCU 1 を減算（16 ビット整数）
- *I ACCU 1 と ACCU 2 の乗算（16 ビット整数）
- /I ACCU 2 を ACCU 1 で除算（16 ビット整数）
- + 整数の加算（16 ビット、32 ビット）
- +D ACCU 1 と ACCU 2 の加算（32 ビット倍長整数）
- -D ACCU 2 から ACCU 1 を減算（32 ビット倍長整数）
- *D ACCU 1 と ACCU 2 の乗算（32 ビット倍長整数）
- /D ACCU 2 を ACCU 1 で除算（32 ビット倍長整数）
- MOD 除算により余りを生成（32 ビット倍長整数）

7.2 整数値演算命令におけるステータスワードのビットの評価

説明

整数値演算命令は、ステータスワードビット CC 1 と CC 0、OV と OS に影響を及ぼします。

以下の表に、整数（16 ビットと 32 ビット）を使用した命令の結果に対応するステータスワードのビットの信号状態を示します。

結果の有効範囲	CC 1	CC 0	OV	OS
0	0	0	0	*
16 ビット: -32 768 <= 結果 < 0 (負数) 32 ビット: -2 147 483 648 <= 結果 < 0 (負数)	0	1	0	*
16 ビット: 32 767 >= 結果 > 0 (正数) 32 ビット: 2 147 483 647 >= 結果 > 0 (正数)	1	0	0	*

* OS ビットは、この命令結果による影響を受けません。

無効な結果範囲	A1	A0	OV	OS
アンダーフロー (加算) 16 ビット: 結果 = -65536 32 ビット: 結果 = -4 294 967 296	0	0	1	1
アンダーフロー (乗算) 16 ビット: 結果 < -32 768 (正数) 32 ビット: 結果 < -2 147 483 648 (正数)	0	1	1	1
オーバーフロー (加算、除算) 16 ビット: 結果 > 32 767 (正数) 32 ビット: 結果 > 2 147 483 647 (正数)	0	1	1	1
オーバーフロー (乗算、除算) 16 ビット: 結果 > 32 767 (正数) 32 ビット: 結果 > 2 147 483 647 (正数)	1	0	1	1
アンダーフロー (加算、除算) 16 ビット: 結果 < -32 768 (負数) 32 ビット: 結果 < -2 147 483 648 (負数)	1	0	1	1
0 による除算	1	1	1	1

演算	A1	A0	OV	OS
+D: 結果 = -4 294 967 296	0	0	1	1
/D または MOD: 0 による除算	1	1	1	1

7.3 +I ACCU 1 と ACCU 2 の加算 (16 ビット整数)

フォーマット

+I

説明

+I (16 ビット整数の加算) は、ACCU 2-L に ACCU 1-L を加算して、その結果を ACCU 1-L に保存します。ACCU 1-L と ACCU 2-L の内容は、16 ビット整数として解釈されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。この命令では、オーバーフロー/アンダーフローが発生すると、32 ビット整数の代わりに 16 ビット整数が生成されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
合計 = 0	0	0	0	-
-32768 <= 合計 < 0	0	1	0	-
32767 >= 合計 > 0	1	0	0	-
合計 = -65536	0	0	1	1
65534 >= 合計 > 32767	0	1	1	1
-65535 <= 合計 < -32768	1	0	1	1

例

STL	説明
L IW10	//IW10 の値を ACCU 1-L にロードする。
L MW14	//ACCU 1-L の内容を ACCU 2-L にロードする。MW14 の値を //ACCU 1-L にロードする。
+I	//ACCU 2-L と ACCU 1-L を加算し、結果を ACCU 1-L に保存する。
T DB1.DBW25	//ACCU 1-L の内容 (結果) を DB1 の DBW25 に転送する。

7.4 -I ACCU 2 から ACCU 1 を減算 (16 ビット整数)

7.4 -I ACCU 2 から ACCU 1 を減算 (16 ビット整数)

フォーマット

-I

説明

-I (16 ビット整数の減算) は、ACCU 2-L から ACCU 1-L を引き、その結果を ACCU 1-L に保存します。ACCU 1-L と ACCU 2-L の内容は、16 ビット整数として解釈されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることはありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。この命令では、オーバーフロー/アンダーフローが発生すると、32 ビット整数の代わりに 16 ビット整数が生成されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	X	X	X	X	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
差 = 0	0	0	0	-
-32768 <= 差 < 0	0	1	0	-
32767 >= 差 > 0	1	0	0	-
65535 >= 差 > 32767	0	1	1	1
-65535 <= 差 < -32768	1	0	1	1

例

STL	説明
L IW10	//IW10 の値を ACCU 1-L にロードする。
L MW14	//ACCU 1-L の内容を ACCU 2-L にロードする。MW14 の値を //ACCU 1-L にロードする。
-I	//ACCU 2-L から ACCU 1-L を減算し、その結果を ACCU 1-L に保存する。
T DB1.DBW25	//ACCU 1-L の内容 (結果) を DB1 の DBW25 に転送する。

7.5 *I ACCU 1 と ACCU 2 の乗算 (16 ビット整数)

フォーマット

*I

説明

*I (16 ビット整数の乗算) は、ACCU 2-L に ACCU 1-L を掛けます。ACCU 1-L と ACCU 2-L の内容は、16 ビット整数として解釈されます。計算結果は、32 ビット整数としてアキュムレータ 1 に保存されます。ステータスワードが OV1 = 1 と OS = 1 の場合、計算結果は 16 ビット整数の範囲外になります。

この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
積 = 0	0	0	0	-
-32768 <= 積 < 0	0	1	0	-
32767 >= 積 > 0	1	0	0	-
1073741824 >= 積 > 32767	1	0	1	1
-1073709056 <= 積 < -32768	0	1	1	1

例

STL	説明
L IW10	//IW10 の値を ACCU 1-L にロードする。
L MW14	//ACCU 1-L の内容を ACCU 2-L にロードする。MW14 の内容を ACCU 1-L にロードする。
*I	//ACCU 2-L と ACCU 1-L を乗算して、その結果を ACCU 1 に保存する。
T DB1.DB25	//ACCU 1 の内容 (結果) を DB1 の DBW25 に転送する。

7.6 // ACCU 2 を ACCU 1 で除算 (16 ビット整数)

フォーマット

//

説明

// (16 ビット整数の除算) は、ACCU 2-L を ACCU 1-L で割ります。ACCU 1-L と ACCU 2-L の内容は、16 ビット整数として解釈されます。命令の結果は、アキュムレータ 1 に保存されます。この結果は、2 つの 16 ビット整数、指数と余りから成ります。指数は ACCU 1-L に、余りは ACCU 1-H に保存されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	X	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
商 = 0	0	0	0	-
-32768 <= 指数 < 0	0	1	0	-
32767 >= 指数 > 0	1	0	0	-
指数 = 32768	1	0	1	1
ゼロ除算	1	1	1	1

例

STL	説明
L IW10	//IW10 の値を ACCU 1-L にロードする。
L MW14	//ACCU 1-L の内容を ACCU 2-L にロードする。MW14 の値を //ACCU 1-L にロードする。
//	//ACCU 2-L を ACCU 1-L で除算する。その結果を ACCU 1 に保存する。ここで、ACCU 1-L: 商、 //ACCU 1-H: 余り
T MD20	//ACCU 1 の内容 (結果) を MD20 に転送する。

例: 13 を 4 で割る

命令 (IW10) の前の ACCU 2-L の内容:	"13"
命令 (MW14) の前の ACCU 1-L の内容:	"4"
命令 /I (ACCU 2-L / ACCU 1) -L:	"13/4"
命令後の ACCU 1-L の内容 (商):	"3"
命令後の ACCU 1-H の内容 (余り):	"1"

7.7 + 整定数の加算 (16 ビット、32 ビット)

フォーマット

+ <integer constant>

アドレス	データタイプ	説明
<integer constant>	(16 ビット、または 32 ビットの整数)	加算される定数

説明

+ <整定数> ACCU 1 の内容に整定数を加算し、その結果を ACCU 1 に保存します。命令は、ステータスワードビットに関係なく実行されます。ステータスビットも影響されません。

+ <16 ビット整定数>: 16 ビット整定数 (範囲-32768 ~+32767) を ACCU 1-L の内容に加算し、その結果を ACCU 1-L に保存します。

+ <32 ビット整定数>: 32 ビット整定数 (範囲 2,147,483,648 ~2,147,483,647) を ACCU 1-L の内容に加算し、その結果を ACCU 1-L に保存します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例 1

STL	説明
L IW10	//IW10 の値を ACCU 1-L にロードする。
L MW14	//ACCU 1-L の内容を ACCU 2-L にロードする。MW14 の値を ACCU 1-L にロードする。
+I	//ACCU 2-L と ACCU 1-L を加算し、結果を ACCU 1-L に保存する。
+ 25	//ACCU 1-L に 25 を加算し、その結果を ACCU 1-L に保存する。
T DB1.DBW25	//ACCU 1-L の内容 (結果) を DB1 の DBW25 に転送する。

例 2

STL	説明
L IW12	
L IW14	
+ 100	//ACCU 1-L に 100 を加算し、その結果を ACCU 1-L に保存する。
>I	//ACCU 2 > ACCU 1、または IW12 > (IW14 + 100) の場合
JC NEXT	//その後、ジャンプラベル NEXT へ条件付きジャンプする。

例 3

STL		説明
L	MD20	
L	MD24	
+D		//ACCU 1 と ACCU 2 を加算し、その結果を ACCU 1 に保存する。
+	L#-200	//ACCU 1 に-200 を加算し、その結果を ACCU 1 に保存する。
T	MD28	

7.8 +D ACCU 1 と ACCU 2 の加算 (32 ビット倍長整数)

フォーマット

+D

説明

+D (32 ビット整数の加算) は、ACCU 2 に ACCU 1 を加算して、その結果を ACCU 1 に保存します。ACCU 1 と ACCU 2 の内容は、32 ビット整数として解釈されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることはありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	x	x	x	x	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
合計 = 0	0	0	0	-
-2147483648 <= 合計 < 0	0	1	0	-
2147483647 >= 合計 > 0	1	0	0	-
合計 = -4294967296	0	0	1	1
4294967294 >= 合計 > 2147483647	0	1	1	1
-4294967295 <= 合計 < -2147483648	1	0	1	1

例

STL	説明
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
+D	//ACCU 2 と ACCU 1 を加算し、その結果を ACCU 1 に保存する。
T DB1.DBD25	//ACCU 1 の内容 (結果) を DB1 の DBD25 に転送する。

7.9 -D ACCU 2 から ACCU 1 を減算 (32 ビット倍長整数)

フォーマット

-D

説明

-D (32 ビット整数の減算) は、ACCU 2 の内容から ACCU 1 の内容を減算し、その結果を ACCU 1 に保存します。ACCU 1 と ACCU 2 の内容は、32 ビット整数として解釈されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
差異 = 0	0	0	0	-
-2147483648 <= 差 < 0	0	1	0	-
2147483647 >= 差 > 0	1	0	0	-
4294967295 >= 差 > 2147483647	0	1	1	1
-4294967295 <= 差 < -2147483648	1	0	1	1

例

STL	説明
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
-D	//ACCU 2 から ACCU 1 を減算し、その結果を ACCU 1 に保存する。
T DB1.DB25	//ACCU 1 の内容 (結果) を DB1 の DB25 に転送する。

7.10 *D ACCU 1 と ACCU 2 の乗算 (32 ビット倍長整数)

フォーマット

*D

説明

*D (32 ビット整数の乗算) は、ACCU 2 の内容に ACCU 1 の内容を掛けます。ACCU 1 と ACCU 2 の内容は、32 ビット整数として解釈されます。計算結果は、32 ビット整数としてアキュムレータ 1 に保存されます。ステータスワードビットが OV1 = 1 および OS = 1 の場合、計算結果は、32 ビット整数の範囲外になります。

この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	X	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
積 = 0	0	0	0	-
-2147483648 <= 積 < 0	0	1	0	-
2147483647 >= 積 > 0	1	0	0	-
積 > 2147483647	1	0	1	1
積 < -2147483648	0	1	1	1

例

STL	説明
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
*D	//ACCU 2 と ACCU 1 を乗算する。結果を ACCU 1 に保存する。
T DB1.DBD25	//ACCU 1 の内容 (結果) を DB1 の DBD25 に転送する。

7.11 /D ACCU 2 を ACCU 1 で除算 (32 ビット倍長整数)

フォーマット

/D

説明

/D (32 ビット整数の除算) は、ACCU 2 の内容を ACCU 1 の内容で割ります。ACCU 1 と ACCU 2 の内容は、32 ビット整数として解釈されます。この命令の結果は、アキュムレータ 1 に保存されます。結果には商のみが含まれ、剰余は含まれません。(MOD 命令を使用すれば、余りを求めることができます)。

この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
商 = 0	0	0	0	-
-2147483648 <= 指数 < 0	0	1	0	-
2147483647 >= 指数 > 0	1	0	0	-
指数 = 2147483648	1	0	1	1
ゼロ除算	1	1	1	1

例

STL	説明
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
/D	//ACCU 2 を ACCU 1 で除算し、その結果 (商) を ACCU 1 に保存する。
T MD20	//ACCU 1 の内容 (結果) を MD20 に転送する。

例: 13 を 4 で割る

命令 (ID10) の前の ACCU 2 の内容:	"13"
命令 (MD14) の前の ACCU 1 の内容:	"4"
命令/D (ACCU 2 / ACCU 1) :	"13/4"
命令後の ACCU 1 の内容 (商) :	"3"

7.12 MOD 除算により余りを生成 (32 ビット倍長整数)

フォーマット

MOD

説明

MOD (32 ビット整数の除算) は、ACCU 2 の内容を ACCU 1 の内容で割ります。ACCU 1 と ACCU 2 の内容は、32 ビット整数として解釈されます。この命令の結果は、アキュムレータ 1 に保存されます。結果には剰余のみが含まれ、商は含まれません。(/D 命令を使用すれば、余りを求めることができます)。

この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	X	-	-	-	-

ステータスビットの生成	CC 1	CC 0	OV	OS
余り = 0	0	0	0	-
-2147483648 <= 余り < 0	0	1	0	-
2147483647 >= 余り > 0	1	0	0	-
ゼロ除算	1	1	1	1

例

STL	説明
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
MOD	//ACCU 2 を ACCU 1 で除算し、その結果 (余り) を ACCU 1 に保存する。
T MD20	//ACCU 1 の内容 (結果) を MD20 に転送する。

例: 13 を 4 で割る

命令 (ID10) の前の ACCU 2 の内容: "13"
 命令 (MD14) の前の ACCU 1 の内容: "4"
 命令 MOD (ACCU 2 / ACCU 1) : "13/4"
 命令後の ACCU 1 の内容 (余り) : "1"

8 浮動小数点数値演算命令

8.1 浮動小数点数値演算命令の概要

説明

数値演算命令は、アキュムレータ 1 とアキュムレータ 2 の内容を結合します。アキュムレータが 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

アキュムレータが 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の元の内容は変わりません。

IEEE 32 ビット浮動小数点数は、REAL と呼ばれるデータタイプに属します。浮動小数点数値演算命令を使用し、**32 ビット IEEE 浮動小数点数を 2 つ**使用すれば、以下の数値演算命令を実行できます。

- +R ACCU 1 と ACCU 2 の加算
- -R ACCU 2 から ACCU 1 を減算
- *R ACCU 1 と ACCU 2 を乗算
- /R ACCU 2 を ACCU 1 で除算

浮動小数点数値演算を使用し、**32 ビット IEEE 浮動小数点数を 1 つ**使用すれば、以下の操作を実行できます。

- ABS 絶対値
- SQR 2 乗の生成
- SQRT 平方根の生成
- EXP 指数値の生成
- LN 自然対数の生成
- SIN 角度のサインの生成
- COS 角度のコサインの生成
- TAN 角度のタンジェントの生成
- ASIN アークサインの生成
- ACOS アークコサインの生成
- ATAN アークタンジェントの生成

8.2 浮動小数点数値演算命令におけるステータスワードのビットの評価

説明

基本算術型は、ステータスワードビット CC 1 と CC 0、OV と OS に影響を及ぼします。

以下の表に、浮動小数点数（32 ビット）を使用した命令の結果に対応するステータスワードのビットの信号状態を示します。

結果の有効範囲	CC 1	CC 0	OV	OS
+0、-0 (Null)	0	0	0	*
-3.402823E+38 < 結果 < -1.175494E-38 (負の数)	0	1	0	*
+1.175494E-38 < 結果 < 3.402824E+38 (正の数)	1	0	0	*

* OS ビットは、この命令結果による影響を受けません。

結果の有効範囲	CC 1	CC 0	OV	OS
アンダーフロー -1.175494E-38 < 結果 < -1.401298E-45 (負数)	0	0	1	1
アンダーフロー +1.401298E-45 < 結果 < +1.175494E-38 (正数)	0	0	1	1
オーバーフロー 結果 < -3.402823E+38 (負の数)	0	1	1	1
オーバーフロー 結果 > 3.402823E+38 (正数)	1	0	1	1
無効な実数または不正な命令 (有効範囲にない入力値)	1	1	1	1

8.3 浮動小数点数値演算命令: 基本

8.3.1 +R ACCU 1 と ACCU 2 の加算 (32 ビットの IEEE 754 浮動小数点数)

フォーマット

+R

命令の説明

+R (32 ビットの IEEE 浮動小数点数の加算) は、アキュムレータ 2 の内容をアキュムレータ 1 の内容に加算して、その結果をアキュムレータ 1 に保存します。アキュムレータ 1 とアキュムレータ 2 の内容は、32 ビットの IEEE 浮動小数点数として解釈されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-infinite	0	1	1	1	オーバーフロー
-qNaN	1	1	1	1	

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

8.3 浮動小数点数値演算命令: 基本

例

STL	説明	
OPN DB10		
L ID10	//ID10 の値を ACCU 1 にロードする。	
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。	
+R	//ACCU 2 と ACCU 1 を加算し、その結果を ACCU 1 に保存する。	
T DBD25	//ACCU 1 の内容（結果）を DB10 の DBD25 に転送する。	

8.3.2 -R ACCU 2 から ACCU 1 を減算 (32 ビットの IEEE 754 浮動小数点数)

フォーマット

-R

説明

-R (32 ビットの IEEE 浮動小数点数の減算) は、アキュムレータ 2 の内容からアキュムレータ 1 の内容を減算して、その結果をアキュムレータ 1 に保存します。アキュムレータ 1 とアキュムレータ 2 の内容は、32 ビットの IEEE 浮動小数点数として解釈されます。この結果は、アキュムレータ 1 に保存されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-infinite	0	1	1	1	オーバーフロー
-qNaN	1	1	1	1	

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

例

STL	説明
OPN DB10	
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
-R	//ACCU 2 から ACCU 1 を減算し、その結果を ACCU 1 に保存する。
T DBD25	//ACCU 1 の内容 (結果) を DB10 の DBD25 に転送する。

8.3.3 *R ACCU 1 と ACCU 2 の乗算 (32 ビットの IEEE 754) 浮動小数点数

フォーマット

*R

命令の説明

*R (32 ビット IEEE 浮動小数点数の乗算) は、アキュムレータ 2 の内容をアキュムレータ 1 の内容で乗算します。アキュムレータ 1 とアキュムレータ 2 の内容は、32 ビットの IEEE 浮動小数点数として解釈されます。計算結果は、32 ビット IEEE 浮動小数点数としてアキュムレータ 1 に保存されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスワードビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。アキュムレータ 4 の内容は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-infinite	0	1	1	1	オーバーフロー
-qNaN	1	1	1	1	

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

例

STL	説明
OPN DB10	
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
*R	//ACCU 2 と ACCU 1 を乗算する。結果を ACCU 1 に保存する。
T DBD25	//ACCU 1 の内容 (結果) を DB10 の DBD25 に転送する。

8.3.4 /R ACCU 2 を ACCU 1 で除算 (32 ビットの IEEE 754 浮動小数点数)

フォーマット

/R

命令の説明

/R (32 ビット IEEE 浮動小数点数の除算) は、アキュムレータ 2 の内容をアキュムレータ 1 の内容で除算します。アキュムレータ 1 とアキュムレータ 2 の内容は、32 ビットの IEEE 浮動小数点数として解釈されます。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスビット CC 1、CC 0、OS、および OV は、命令の結果のファンクションとして設定されます。

ACCU が 2 つの CPU の場合、アキュムレータ 2 の内容は変わりません。

ACCU が 4 つの CPU の場合、アキュムレータ 3 の内容はアキュムレータ 2 にコピーされ、アキュムレータ 4 の内容はアキュムレータ 3 にコピーされます。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-infinite	0	1	1	1	オーバーフロー
-qNaN	1	1	1	1	

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	x	-	-	-	-

例

STL	説明
OPN DB10	
L ID10	//ID10 の値を ACCU 1 にロードする。
L MD14	//ACCU 1 の値を ACCU 2 にロードする。MD14 の値を ACCU 1 にロードする。
/R	//ACCU 2 を ACCU 1 で除算し、その結果を ACCU 1 に保存する。
T DBD20	//ACCU 1 の内容 (結果) を DB10 の DBD20 に転送する。

8.3.5 ABS 浮動小数点数の絶対値 (32 ビットの IEEE 754)

フォーマット

ABS

説明

ABS (32 ビットの IEEE 浮動小数点数の絶対値) は、ACCU 1 に格納された浮動小数点数の絶対値を生成します (32 ビットの IEEE 浮動小数点数)。この結果は、アキュムレータ 1 に保存されます。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L ID8	//値を ACCU 1 にロードする (例: ID8= -1.5E+02)。
ABS	//絶対値を形成し、その結果を ACCU 1 に保存する。
T MD10	//結果を MD10 に転送する (例: result = 1.5E+02)。

8.4 浮動小数点数値演算命令: 拡張

8.4.1 SQR 浮動小数点数（32 ビット）の 2 乗を生成

フォーマット

SQR

命令の説明

SQR（32 ビットの IEEE 754 浮動小数点数の 2 乗を生成）は、ACCU 1 に格納された浮動小数点数の 2 乗を計算します（32 ビット、IEEE 754）。この結果は、アキュムレータ 1 に保存されます。この命令により、CC 1、CC 0、OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-qNaN	1	1	1	1	

例

STL	説明
OPN DB17	//データブロック DB17 を開く。
L DBD0	//データダブルワード DBD0 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。
SQR	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）の //2 乗を計算する。結果を ACCU 1 に保存する。
AN OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。
JC OK	//SQR 命令の実行中にエラーが発生しなければ、 //OK ジャンプラベルへジャンプする。
BEU	//SQR 命令の実行中にエラーが発生しなければ、無条件でブロックの最後へ。
OK: T DBD4	//結果を ACCU 1 からデータダブルワード DBD4 に転送する。

8.4.2 Sqrt 浮動小数点数（32 ビット）の平方根を生成

フォーマット

Sqrt

命令の説明

Sqrt（32 ビットの IEEE 754 浮動小数点数の平方根を生成）は、ACC0 に格納された浮動小数点数の平方根を計算します（32 ビット、IEEE 754）。この結果は、アキュムレータ 0 に保存されます。入力値は 0 以上でなければなりません。これにより、正の結果が得られます。唯一の例外: -0 の平方根は -0 です。この命令により、CC 0、CC 1、OV、および OS のステータスワードビットは変更されます。アキュムレータ 1 の内容 (ACC0 が 4 つの CPU の場合はさらにアキュムレータ 2 と 3) は変わりません。

結果

ACC0 の結果	CC 0	CC 1	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-qNaN	1	1	1	1	

例

STL	説明	
L MD10	//メモリダブルワード MD10 の値が ACC0 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。	
Sqrt	//ACC0 に格納された浮動小数点数（32 ビット、IEEE 754）の //平方根を計算する。結果を ACC0 に保存する。	
AN OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。	
JC OK	//Sqrt 命令の実行中にエラーが発生しなければ、 //OK ジャンプラベルへジャンプする。	
BEU	//Sqrt 命令の実行中にエラーが発生しなければ、無条件でブロックの最後へ。	
OK: T MD20	//結果を ACC0 からメモリダブルワード MD20 に転送する。	

8.4.3 EXP 浮動小数点数（32 ビット）の指数値を生成

フォーマット

EXP

命令の説明

EXP（32 ビットの IEEE 754 浮動小数点数の指数値を生成）は、ACCU 1 に格納された浮動小数点数の指数値（e を底とする指数）を計算します（32 ビット、IEEE 754）。この結果は、アキュムレータ 1 に保存されます。この命令により、CC 1, CC 0、OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-qNaN	1	1	1	1	

例

STL			説明
	L	MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。
	EXP		//（この値は浮動小数点数形式にする必要がある）。 //ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）の、 //e を底とする指数を計算する。結果を ACCU 1 に保存する。
	AN	OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。
	JC	OK	//EXP 命令の実行中にエラーが発生しなければ、 //OK ジャンプラベルへジャンプする。
BEU			//EXP 命令の実行中にエラーが発生しなければ、無条件でブロックの最後へ。
OK:	T	MD20	//結果を ACCU 1 からメモリダブルワード MD20 に転送する。

8.4.4 LN 浮動小数点数（32 ビット）の自然対数を生成

フォーマット

LN

命令の説明

LN（32 ビットの IEEE 754 浮動小数点数の自然対数を生成）は、ACCU 1 に格納された浮動小数点数の自然対数（e を底とする対数）を計算します（32 ビット、IEEE 754）。この結果は、アキュムレータ 1 に保存されます。入力値は 0 より大きい必要があります。この命令により、CC 1、CC 0、UO、および OV のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-infinite	0	1	1	1	オーバーフロー
-qNaN	1	1	1	1	

例

STL	説明	
BEU OK:	L MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。
	LN	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）の //自然対数を計算する。結果を ACCU 1 に保存する。
	AN OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。
	JC OK	//命令の実行中にエラーが発生しなければ、OK ジャンプラベルへジャンプする。
	T MD20	//命令の実行中にエラーが発生しなければ、無条件でブロック最後へ。 //結果を ACCU 1 からメモリダブルワード MD20 に転送する。

8.4.5 SIN 角度のサイン（32 ビット浮動小数点数）を生成

フォーマット

SIN

命令の説明

SIN（角度のサインを 32 ビットの IEEE 754 浮動小数点数で生成）は、角度のサインをラジアン単位で計算します。角度は浮動小数点数として ACCU 1 に存在する必要があります。この結果は、アキュムレータ 1 に保存されます。この命令により、CC 1, CC 0、OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	オーバーフロー
+zero	0	0	0	-	
+infinite	1	0	1	1	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

関連項目: 浮動小数点命令によるステータスワードのビットの評価

例

STL	説明
L MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。
SIN	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）のサインを計算する。 //結果を ACCU 1 に保存する。
T MD20	//結果を ACCU 1 からメモリダブルワード MD20 に転送する。

8.4.6 COS 角度のコサイン（32 ビット浮動小数点数）を生成

フォーマット

COS

命令の説明

COS（角度のコサインを 32 ビットの IEEE 754 浮動小数点数で生成）は、角度のコサインをラジアン単位で計算します。角度は浮動小数点数として ACCU 1 に存在する必要があります。この結果は、アキュムレータ 1 に保存されます。この命令により、CC 1, CC 0、OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	オーバーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

例

STL	説明
L MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。
COS	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）のコサインを計算する。 //結果を ACCU 1 に保存する。
T MD20	//結果を ACCU 1 からメモリダブルワード MD20 に転送する。

8.4.7 TAN 角度のタンジェント（32 ビット浮動小数点数）を生成

フォーマット

TAN

命令の説明

TAN（角度のタンジェントを 32 ビットの IEEE 754 浮動小数点数で生成）は、角度のタンジェントをラジアン単位で計算します。角度は浮動小数点数として ACCU 1 に存在する必要があります。この結果は、アキュムレータ 1 に保存されます。この命令により、CC 1, CC 0, OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+infinite	1	0	1	1	オーバーフロー
+normalized	1	0	0	-	
+denormalized	0	0	1	1	アンダーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-infinite	0	1	1	1	オーバーフロー
-qNaN	1	1	1	1	

例

STL	説明	
L MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。	
TAN	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）の //タンジェントを計算する。結果を ACCU 1 に保存する。	
AN OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。	
JC OK	//TAN 命令の実行中にエラーが発生しなければ、OK ジャンプラベルへジャンプする。	
BEU	//命令の実行中にエラーが発生しなければ、無条件でブロックの最後へ。	
OK: T MD20	//結果を ACCU 1 からメモリダブルワード MD20 に転送する。	

8.4.8 ASIN 浮動小数点数（32 ビット）のアークサインを生成

フォーマット

ASIN

命令の説明

ASIN（浮動小数点数のアークサインを生成、32 ビット、IEEE 754）は、ACCU 1 に格納された浮動小数点数のアークサインを計算します。入力値の許容範囲

$-1 \leq \text{入力値} \leq +1$

計算結果はラジアン単位の角度です。値の範囲は以下のとおりです。

$-\pi / 2 \leq \text{arc sine (ACCU1)} \leq +\pi / 2$, with $\pi = 3.14159...$

この命令により、CC 1, CC 0、OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	オーバーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

例

STL	説明	
L MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。	
ASIN	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）の //アークサインを計算する。結果を ACCU 1 に保存する。	
AN OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。	
JC OK	//ASIN 命令の実行中にエラーが発生しなければ、OK ジャンプラベルへジャンプする。	
BEU	//ASIN 命令の実行中にエラーが発生しなければ、無条件でブロックの最後へ。	
OK: T MD20	//結果を ACCU 1 からメモリダブルワード MD20 に転送する。	

8.4.9 ACOS 浮動小数点数（32 ビット）のアーコサインを生成

フォーマット

ACOS

命令の説明

ACOS（浮動小数点数のアーコサインを生成、32 ビット、IEEE 754）は、ACCU 1 に格納された浮動小数点数のアーコサインを計算します。入力値の許容範囲

$-1 \leq \text{入力値} \leq +1$

計算結果はラジアン単位の角度です。値の範囲は以下のとおりです。

$0 \leq \text{arc cosine (ACCU1)} \leq \pi$, with $\pi = 3.14159\dots$

この命令により、CC 1, CC 0, OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	オーバーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

例

STL	説明	
L MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。	
ACOS	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）の //アーコサインを計算する。結果を ACCU 1 に保存する。	
AN OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。	
JC OK	//ACOS 命令の実行中にエラーが発生しなければ、OK ジャンプラベルへジャンプする。	
BEU	//ACOS 命令の実行中にエラーが発生しなければ、無条件でブロック最後へ。	
OK: T MD20	//結果を ACCU 1 からメモリダブルワード MD20 に転送する。	

8.4.10 ATAN 浮動小数点数（32 ビット）のアークタンジェントを生成

フォーマット

ATAN

命令の説明

ATAN（浮動小数点数、32 ビット、IEEE 754 のアークタンジェントを生成）は、ACCU 1 に格納された浮動小数点数のアークタンジェントを計算します。計算結果はラジアン単位の角度になります。値の範囲は、以下のとおりです。

$$-\pi / 2 \leq \text{arc tangent (ACCU1)} \leq +\pi / 2, \text{ with } \pi = 3.14159...$$

この命令により、CC 1、CC 0、OV、および OS のステータスワードビットは変更されます。

アキュムレータ 2 の内容 (ACCU が 4 つの CPU の場合はさらにアキュムレータ 3 と 4) は変わりません。

結果

ACCU 1 の結果	CC 1	CC 0	OV	OS	注記
+ qNaN	1	1	1	1	
+normalized	1	0	0	-	
+denormalized	0	0	1	1	オーバーフロー
+zero	0	0	0	-	
-zero	0	0	0	-	
-denormalized	0	0	1	1	アンダーフロー
-normalized	0	1	0	-	
-qNaN	1	1	1	1	

例

STL	説明		
L MD10	//メモリダブルワード MD10 の値が ACCU 1 にロードされる。 //（この値は浮動小数点数形式にする必要がある）。		
ATAN	//ACCU 1 に格納された浮動小数点数（32 ビット、IEEE 754）の //アークタンジェントを計算する。結果を ACCU 1 に保存する。		
AN OV	//ステータスワードの OV ビットが"0"かどうかをスキャンする。		
JC OK	//ATAN 命令の実行中にエラーが発生しなければ、 //OK ジャンプラベルへジャンプする。		
BEU	//ATAN 命令の実行中にエラーが発生しなければ、無条件でブロック最後へ。		
OK: T MD20	//結果を ACCU 1 からメモリダブルワード MD20 に転送する。		

9 ロード命令と転送命令

9.1 ロード命令と転送命令の概要

説明

ロード命令（L）と転送命令（T）を使用すれば、入出力モジュールとメモリ領域間またはメモリ領域間での情報交換をプログラムできます。CPU は、これらの命令を無条件命令として、すなわち、ステートメントの論理演算結果には関係なく、それぞれのスキャンサイクル内で実行します。

以下のロード命令と転送命令を使用できます。

- L ロード
- L STW ステータスワードを ACCU 1 にロード
- LAR1 AR2 アドレスレジスタ 2 からアドレスレジスタ 1 をロード
- LAR1 <D> アドレスレジスタ 1 を倍長整数（32 ビットポイント）と共にロード
- LAR1 ACCU 1 からアドレスレジスタ 1 をロード
- LAR2 <D> 倍長整数（32 ビットポイント）と共にアドレスレジスタ 2 をロード
- LAR2 ACCU 1 からアドレスレジスタ 2 をロード

- T 転送
- T STW ACCU 1 をステータスワードへ転送
- TAR1 AR2 アドレスレジスタ 1 をアドレスレジスタ 2 へ転送
- TAR1 <D> アドレスレジスタ 1 を宛先へ転送（32 ビットポイント）
- TAR2 <D> アドレスレジスタ 2 を宛先へ転送（32 ビットポイント）
- TAR1 アドレスレジスタ 1 を ACCU 1 へ転送
- TAR2 アドレスレジスタ 2 を ACCU 1 へ転送
- CAR アドレスレジスタ 1 をアドレスレジスタ 2 と交換

9.2 L ロード

フォーマット

L <address>

アドレス	データタイプ	メモリ領域	ソースアドレス
<address>	BYTE	E、A、PE、M、L、D、 ポインタ、パラメータ	0...65535
	WORD		0...65534
	DWORD		0...65532

説明

L <address>は、ACCU 1 の前の内容を ACCU 2 に保存してからアドレスバイト、ワード、ダブルワードを ACCU 1 にロードし、ACCU 1 を"0"にリセットします。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L IB10	//入力バイト IB10 を ACCU 1-L-L にロードする。
L MB120	//メモリバイト MB120 を ACCU 1-L-L にロードする。
L DBB12	//データバイト DBB12 を ACCU 1-L-L にロードする。
L DIW15	//インスタンスデータワード DIW15 を ACCU 1-L にロードする。
L LD252	//ローカルデータダブルワード LD252 を ACCU 1 にロードする。
L P# I 8.7	//ポインタを ACCU 1 にロードする。
L OTTO	//パラメータ"OTTO"を ACCU 1 にロードする。
L P# ANNA	//ポインタを ACCU 1 の指定パラメータにロードする。(この命令は、 //指定パラメータの相対アドレスオフセットをロードする。複数のインスタンス FB //でインスタンスデータブロックの絶対オフセットを計算するには、この値に、 //AR2 レジスタの内容を追加する必要があります。

ACCU 1 の内容

ACCU 1 の内容	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
ロード命令実行前の値	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
命令実行後の値 L MB10 (L <Byte>)	00000000	00000000	00000000	<MB10>
命令実行後の値 L MW10 (L <word>)	00000000	00000000	<MB10>	<MB11>
命令実行後の値 L MD10 (L <double word>)	<MB10>	<MB11>	<MB12>	<MB13>
命令実行後の値 L P# ANNA (FB 内)	<86>	< FB の開始に関連した ANNA のビットオフセット。> 複数のインスタンス FB でインスタンスデータブロックの絶対オフセットを計算するには、この値に、AR2 レジスタの内容を追加する必要があります。		
命令実行後の値 L P# ANNA (FC 内)	<ANNA に転送されるデータの領域間のアドレス>			
	X = "1"または"0"			

9.3 L STW ステータスワードを ACCU 1 にロード

フォーマット

L STW

説明

L STW（アドレス STW 付きの L）は、ステータスワードの内容と共に ACCU 1 をロードします。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

注

S7-300 シリーズの CPU では、ステートメント **L STW** は、ステータスワードの FC、STA、OR の各ビットをロードしません。ビット 1、4、5、6、7、8 だけが、アキュムレータ 1 の下位ワードの該当するビットの位置へロードされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L STW	//ステータスワードの内容を ACCU 1 にロードする。

L STW を実行後の ACCU 1 の内容

ビット	31-9	8	7	6	5	4	3	2	1	0
内容	0	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC

9.4 LAR1 ACCU 1 からアドレスレジスタ 1 をロード

フォーマット

LAR1

説明

LAR1 は、ACCU 1 の内容 (32 ビットポインタ) と共にアドレスレジスタ AR1 をロードします。ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

9.5 LAR1 <D> アドレスレジスタ 1 を倍長整数 (32 ビットポインタ) と共にロード

フォーマット

LAR1 <D>

アドレス	データタイプ	メモリ領域	ソースアドレス
<D>	DWORD ポインタ定数	D、M、L	0...65532

説明

LAR1 <D>は、アドレス指定されたダブルワード<D>またはポインタ定数と共にアドレスレジスタ AR1 をロードします。ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例: 直接アドレス

STL	説明
LAR1 DBD20	//データダブルワード DBD20 のポインタを AR1 にロードする。
LAR1 DID30	//インスタンスデータダブルワード DID30 のポインタを AR1 にロードする。
LAR1 LD180	//ローカルデータダブルワード LD180 のポインタを AR1 にロードする。
LAR1 MD24	//メモリダブルワード MD24 の定数を AR1 にロードする。

例: ポインタ定数

STL	説明
LAR1 P#M100.0	//32 ビットポインタ定数を AR1 にロードする。

9.6 LAR1 AR2 アドレスレジスタ 2 からアドレスレジスタ 1 をロード

フォーマット

LAR1 AR2

説明

LAR1 AR2 (アドレス AR2 付き LAR1 命令) は、アドレスレジスタ AR2 の内容をアドレスレジスタ AR1 にロードします。ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

9.7 LAR2 ACCU 1 からアドレスレジスタ 2 をロード

フォーマット

LAR2

説明

LAR2 は、ACCU 1 (32 ビットポインタ) と共にアドレスレジスタ AR2 をロードします。

ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

9.8 LAR2 <D> 倍長整数 (32 ビットポインタ) と共にアドレスレジスタ 2 をロード

フォーマット

LAR2 <D>

アドレス	データタイプ	メモリ領域	ソースアドレス
<D>	DWORD ポインタ定数	D、M、L	0...65532

説明

LAR2 <D>は、アドレス指定されたダブルワード<D>またはポインタ定数と共にアドレスレジスタ AR2 をロードします。ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例: 直接アドレス

STL	説明
LAR2 DBD 20	//データダブルワード DBD20 のポインタを AR2 にロードする。
LAR2 DID 30	//インスタンスデータダブルワード DID30 のポインタを AR2 にロードする。
LAR2 LD 180	//ローカルデータダブルワード LD180 のポインタを AR2 にロードする。
LAR2 MD 24	//データダブルワード DBD24 のポインタを AR2 にロードする。

例: ポインタ定数

STL	説明
LAR2 P#M100.0	//32 ビットポインタ定数を AR2 にロードする。

9.9 T 転送

フォーマット

T <address>

アドレス	データタイプ	メモリ領域	ソースアドレス
<address>	BYTE	I, Q, PQ, M, L, D	0...65535
	WORD		0...65534
	DWORD		0...65532

説明

T <address>は、マスタコントロールリレーがオン（MCR = 1）の場合、ACCU 1 の内容を宛先アドレスに転送（コピー）します。MCR が 0 の場合、宛先アドレスは 0 を使用して書き込まれます。ACCU 1 からコピーされるバイト数は、宛先アドレスで表されるサイズによって異なります。ACCU 1 はさらに、転送手続きの終了後にデータの保存も行います。直接 I/O 領域（メモリタイプ PQ）の転送は、ACCU 1 の内容または"0"（MCR=0 の場合）を、プロセスイメージ出力テーブル（メモリタイプ Q）の対応するアドレスに転送します。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
T QB10	//ACCU 1-L-L の内容を出力バイト QB10 へ転送する。
T MW14	//ACCU 1-L の内容をメモリワード MW14 へ転送する。
T DBD2	//ACCU 1 の内容をデータダブルワード DBD2 へ転送する。

9.10 T STW ACCU 1 をステータスワードへ転送

フォーマット

T STW

説明

T STW（アドレス STW 付き T 命令）は、ACCU 1 のビット 0 からビット 8 をステータスワードに転送します。

この命令はステータスビットに関係なく実行され、これによりステータスビットが変更されることはありません。

注: S7-300 ファミリの CPU では、ステータスワード/ER、STA、OR のビットは、T STW 命令によって書き込まれません。ビット 1、4、5、6、7、および 8 のみが、ACCU1 のビット設定に従って書き込まれます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	x	x	x	x	x	x	x	x	x

例

STL	説明
T STW	// ACCU 1 のビット 0 からビット 8 をステータスワードへ転送する。

ACCU 1 のビットには次のステータスビットが含まれています。

ビット	31-9	8	7	6	5	4	3	2	1	0
内容	*)	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC

*) ビットは転送されない。

9.11 CAR アドレスレジスタ 1 をアドレスレジスタ 2 と交換

フォーマット

CAR

説明

CAR（アドレスレジスタの交換）は、アドレスレジスタ AR1 と AR2 の内容を交換します。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

アドレスレジスタ AR1 の内容を、アドレスレジスタ AR2 へ移動します。
 アドレスレジスタ AR2 の内容を、アドレスレジスタ AR1 へ移動します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

9.12 TAR1 アドレスレジスタ 1 を ACCU 1 へ転送

フォーマット

TAR1

説明

TAR1 は、アドレスレジスタ AR1 の内容を ACCU 1（32 ビットポインタ）へ転送します。ACCU 1 の以前の内容は、ACCU 2 に保存されます。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

9.13 TAR1 <D> アドレスレジスタ 1 を宛先へ転送 (32 ビットポインタ)

フォーマット

TAR1 <D>

アドレス	データタイプ	メモリ領域	ソースアドレス
<D>	DWORD	D、M、L	0...65532

説明

TAR1 <D> は、アドレスレジスタ AR1 の内容をアドレスダブルワード<D>に転送します。宛先領域には、メモリダブルワード (MD)、ローカルデータダブルワード (LD)、データダブルワード (DBD)、およびインスタンスデータワード (DID) があります。

ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
TAR1 DBD20	//AR1 の内容をデータダブルワード DBD20 に転送する。
TAR1 DID30	//AR1 の内容をインスタンスデータダブルワード DID30 に転送する。
TAR1 LD18	//AR1 の内容をローカルデータダブルワード LD18 に転送する。
TAR1 MD24	//AR1 の内容をメモリダブルワード MD24 に転送する。

9.14 TAR1 AR2 アドレスレジスタ 1 をアドレスレジスタ 2 へ転送

フォーマット

TAR1 AR2

説明

TAR1 AR2（アドレス AR2 付き TAR1 命令）は、アドレスレジスタ AR1 の内容をアドレスレジスタ AR2 へ転送します。

ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

9.15 TAR2 アドレスレジスタ 2 を ACCU 1 へ転送

フォーマット

TAR2

説明

TAR2 は、アドレスレジスタ AR2 の内容を ACCU 1（32 ビットポインタ）へ転送します。ACCU 1 の以前の内容は、以前に ACCU 2 に保存されています。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

9.16 TAR2 <D> アドレスレジスタ 2 を宛先へ転送 (32 ビットポインタ)

フォーマット

TAR2 <D>

アドレス	データタイプ	メモリ領域	ソースアドレス
<D>	DWORD	D、M、L	0...65532

説明

TAR2 <D>は、アドレスレジスタ AR2 の内容を、アドレス指定されたダブルワード<D>へ転送します。宛先領域には、メモリダブルワード (MD)、ローカルデータダブルワード (LD)、データダブルワード (DBD)、およびインスタンスデータワード (DID) があります。

ACCU 1 と ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
TAR2 DBD20	//AR2 の内容をデータダブルワード DBD20 に転送する。
TAR2 DID30	//AR2 の内容をデータダブルワード DBD30 に転送する。
TAR2 LD18	//AR2 の内容をローカルデータダブルワード LD18 に転送する。
TAR2 MD24	//AR2 の内容をメモリダブルワード MD24 に転送する。

10 プログラム制御命令

10.1 プログラム制御命令の概要

説明

プログラム制御命令の実行には、以下の命令を使用できます。

- BE ブロックの終了
- BEC 条件付きのブロックの終了
- BEU 無条件ブロックの終了
- CALL ブロック呼び出し
- CC 条件付き呼び出し
- UC 無条件呼び出し

- FB の呼び出し
- FC の呼び出し
- SFB の呼び出し
- SFC の呼び出し
- 複数インスタンスの呼び出し
- ライブラリからのブロックの呼び出し

- MCR (マスタコントロールリレー)
- MCR ファンクションの使用法に関する重要注記
- MCR (MCR スタックに RLO を保存、MCR の開始
-) MCR MCR オフ
- MCRA MCR 領域の有効化
- MCRD MCR 領域の無効化

10.2 BE ブロックの終了

フォーマット

BE

説明

BE（ブロックの終了）は、現在のブロックのプログラムスキャンを終了し、このブロックを呼び出したブロックへジャンプします。そして、呼び出しプログラムのブロック呼び出しステートメントの次の命令から、プログラムスキャンを再開します。現在のローカルデータ領域が解放され、前回のローカルデータ領域が現在のローカルデータ領域になります。ブロックが呼び出された時に開いていたデータブロックが再度開きます。さらに、呼び出し側ブロックの MCR の依存性が復元され、現在のブロックから呼び出し側であるブロックへ RLO が渡されます。BE は、どの状態からとも影響を受けません。ただし、BE 命令がジャンプすると、現在のプログラムスキャンは終了せず、ブロック内のジャンプ先から実行します。

BE 命令は、S5 ソフトウェアとは異なります。S7 ハードウェアで使用する場合、この命令は機能的に BEU と同等です。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	0	0	1	-	0

例

STL	説明
A I 1.0	
JC NEXT	//RLO が 1 (I 1.0 = 1) の場合、NEXT ジャンプラベルへジャンプする。
L IW4	//ジャンプが実行されない場合、ここから継続される。
T IW10	
A I 6.0	
A I 6.1	
S M 12.0	
BE	//ブロックの終了
NEXT: NOP	//ジャンプが実行される場合、ここから継続される。
0	

10.3 BEC 条件付きのブロックの終了

フォーマット

BEC

説明

RLO が 1 になると、**BEC**（条件付きのブロックの終了）は現在のブロックでのプログラムスキャンを中断し、呼び出し側のブロックへジャンプします。プログラムスキャンは、呼び出し側プログラムのブロック呼び出しステートメントの次の命令から再開されます。現在のローカルデータ領域が解放され、前回のローカルデータ領域が現在のローカルデータ領域になります。ブロックを呼び出した時に開いていたデータブロックが再度開きます。さらに、呼び出し側ブロックの MCR の依存性が復元されます。

終了したブロックから呼び出したブロックへ RLO (= 1) が渡されます。RLO が 0 の場合、BEC は実行されません。RLO は 1 に設定され、BEC の次の命令からプログラムスキャンが実行されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	x	0	1	1	0

例

STL	説明
A I 1.0	//RLO を更新する。
BEC	//RLO が 1 の場合にブロックを終了する。
L IW4	//BEC が実行されない場合 (RLO = 0)、ここから継続される。
T MW10	

10.4 BEU 無条件ブロックの終了

フォーマット

BEU

説明

BEU（無条件ブロックの終了）は、現在のブロックのプログラムスキャンを終了し、このブロックを呼び出したブロックへジャンプします。プログラムスキャンは、呼び出し側プログラムのブロック呼び出しステートメントの次の命令から再開されます。現在のローカルデータ領域が解放され、前回のローカルデータ領域が現在のローカルデータ領域になります。ブロックが呼び出された時に開いていたデータブロックが再度開きます。さらに、呼び出し側ブロックの MCR の依存性が復元され、現在のブロックから呼び出し側であるブロックへ RLO が渡されます。BEU は、どの状態からも影響を受けません。ただし、BEU 命令がジャンプすると、現在のプログラムスキャンは終了せず、ブロック内のジャンプ先から実行します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	0	0	1	-	0

例

STL	説明	
A I 1.0	JC	NEXT
L IW4	T	IW10
A I 6.0	A	I 6.1
S M 12.0	BEU	
NEXT: NOP 0		

10.5 CALL ブロック呼び出し

フォーマット

CALL <logic block identifier>

説明

CALL <logic block identifier>を使用すれば、ファンクション（FC）またはファンクションブロック（FB）、システムファンクション（SFC）、またはシステムファンクションブロック（SFB）を呼び出したり、Siemens 提供の標準事前プログラムブロックを呼び出したりできます。CALL 命令は、RLO や他の条件に関係なく、ユーザーがアドレスを指定した FC および SFC、FB および SFB を呼び出します。FB または SFB を CALL で呼び出す場合、インスタンス DB が関連付けられているブロックを指定する必要があります。呼び出し側のブロックプログラムは、呼び出されたブロックを処理した後にロジックプロセスを引き続き実行します。ロジックブロックには、絶対アドレス指定またはシンボルアドレス指定を使用できます。SFB/SFC の呼び出し後、レジスタの内容が復元されます。

例: FB1, DB1 の呼び出しまたは FILLVAT1, RECIPE1 の呼び出し

ロジックブロック	ブロックの種類	絶対アドレスによる呼び出し構文
FC	ファンクション	CALL FCn
SFC	システムファンクション	CALL SFCn
FB	ファンクションブロック	CALL FBn1,DBn2
SFB	システムファンクションブロック	CALL SFBn1,DBn2

注記

STL エディタを使用する場合、上記の表の参照（n、n1、n2）は、有効な既存ブロックを指す必要があります。また、使用する前にシンボル名を定義しておく必要があります。

パラメータの引き渡し（インクリメンタル編集モード）

呼び出し側ブロックは、変数リストを介して、呼び出されたブロックとパラメータのやりとりを行います。有効な CALL ステートメントを入力すると、変数リストは、自動的に STL プログラムに拡張されます。

FB、SFB、FC、または SFC の呼び出しを行い、そのブロックの変数宣言テーブルに IN、OUT、および IN_OUT の宣言がある場合、これらの変数は、呼び出し側ブロックに仮パラメータリストとして追加されます。

FC および SFC を呼び出す場合、呼び出し側ロジックブロックの実パラメータを仮パラメータに割り付ける必要があります。

FB および SFB を呼び出す場合、前回の呼び出しから変更しなければならない実パラメータのみを指定します。FB が処理されると、インスタンス DB に実パラメータが保存されます。実パラメータがデータブロックの場合、完全な絶対アドレス（たとえば、DB1、DBW2）を指定する必要があります。

IN パラメータは、定数、絶対アドレス、またはシンボルアドレスで指定できます。OUT および IN_OUT パラメータは、絶対アドレスまたはシンボルアドレスで指定します。すべてのアドレスおよび定数は、転送されるデータタイプと必ず互換性をもつようにします。

CALL はリターンアドレス（セレクトアおよび相対アドレス）、2 つの現在のデータブロックのセレクトア、MA ビットを B（ブロック）スタックに保存します。さらに、CALL は、MCR の依存性を無効にし、呼び出されるブロックのローカルデータ領域を作成します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	0	0	1	-	0

例 1: FC6 呼び出しへのパラメータの割り付け

CALL	FC6	
	仮パラメータ	実パラメータ
	NO OF TOOL	:= MW100
	TIME OUT	:= MW110
	FOUND	:= Q 0.1
	エラー	:= Q 100.0

例 2: パラメータなしの SFC の呼び出し

STL	説明
CALL	SFC43 //SFC43 を呼び出して、ウォッチドッグタイマを再トリガする（パラメータなし）。

例 3: インスタンスデータブロック DB1 による FB99 の呼び出し

CALL	FB99,DB1	
	仮パラメータ	実パラメータ
	MAX_RPM	:= #RPM1_MAX
	MIN_RPM	:= #RPM1
	MAX_POWER	:= #POWER1
	MAX_TEMP	:= #TEMP1

例 4: インスタンスデータブロック DB2 による FB99 の呼び出し

CALL	FB99,DB2	
	仮パラメータ	実パラメータ
	MAX_RPM	:= #RPM2_MAX
	MIN_RPM	:= #RPM2
	MAX_POWER	:= #POWER2
	MAX_TEMP	:= #TEMP2

注

FB または SFB CALL には、インスタンスデータブロックが必要です。上記の例では、ブロック DB1 および DB2 がなければ、呼び出しを実行することはできません。

10.6 FB の呼び出し

フォーマット

CALL FB n1, DB n1

説明

この命令は、ユーザー定義のファンクションブロック（FB）を呼び出すために使用します。CALL 命令は、アドレスが指定されたファンクションブロックを呼び出します。RLO や他の条件には影響されません。CALL を使ってファンクションブロックを呼び出す場合、インスタンスデータブロックを指定する必要があります。呼び出されたブロックの処理後、呼び出し側ブロックのプログラムにより処理が実行されます。ロジックブロックには、絶対アドレス指定またはシンボルアドレス指定を使用できます。

パラメータの引き渡し（インクリメンタル編集モード）

呼び出し側ブロックは、変数リストを介して、呼び出されたブロックとパラメータのやりとりを行います。有効な CALL 命令を指定すると、変数リストは、自動的にステートメントリストプログラムに拡張されます。

ファンクションブロックの呼び出しを行い、このブロックの変数宣言テーブルに IN、OUT、および IN_OUT の宣言がある場合、これらの変数は、呼び出し側ブロックのプログラムに仮パラメータリストとして追加されます。

ファンクションブロックは、実パラメータを指定するだけで呼び出せます。実パラメータは、ファンクションブロックの処理後にインスタンスデータブロックに保存されるので、前回の呼び出しから変更する必要があります。実パラメータがデータブロックの場合、完全な絶対アドレス（たとえば、DB1、DBW2）を指定する必要があります。

IN パラメータは、定数、絶対アドレス、またはシンボルアドレスで指定できます。OUT および IN_OUT パラメータは、絶対アドレスまたはシンボルアドレスで指定します。すべてのアドレスおよび定数は、転送されるデータタイプと必ず互換性をもつようにします。

CALL は、リターンアドレス（セクタおよび相対アドレス）、2 つの開いているデータブロックのセクタ、MA ビットを B（ブロック）スタックに保存します。さらに、CALL は、MCR の依存性を無効にし、呼び出されるブロックのローカルデータ領域を作成します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	0	0	1	-	0

例 1: インスタンスデータブロック DB1 による FB99 の呼び出し

CALL	FB99,DB1	
	仮パラメータ	実パラメータ
	MAX_RPM	:= #RPM1_MAX
	MIN_RPM	:= #RPM1
	MAX_POWER	:= #POWER1
	MAX_TEMP	:= #TEMP1

例 2: インスタンスデータブロック DB2 による FB99 の呼び出し

CALL	FB99,DB2	
	仮パラメータ	実パラメータ
	MAX_RPM	:= #RPM2_MAX
	MIN_RPM	:= #RPM2
	MAX_POWER	:= #POWER2
	MAX_TEMP	:= #TEMP2

注

ファンクションブロック CALL には、インスタンスデータブロックが必要です。上記の例では、ブロック DB1 および DB2 がなければ、呼び出しを実行することはできません。

10.7 FC の呼び出し

フォーマット

CALL FC n

注記

STL エディタで作業している場合、参照 (n) を既存の有効ブロックに関連付ける必要があります。また、使用する前にシンボル名を定義しておく必要があります。

説明

この命令は、ファンクション (FC) を呼び出します。CALL 命令は、アドレスが指定されたファンクションを呼び出します。RLO や他の条件には影響されません。呼び出されたブロックの処理後、呼び出し側ブロックのプログラムにより処理が実行されます。ロジックブロックには、絶対アドレス指定またはシンボルアドレス指定を使用できます。

パラメータの引き渡し (インクリメンタル編集モード)

呼び出し側ブロックは、変数リストを介して、呼び出されたブロックとパラメータのやりとりを行います。有効な CALL 命令を指定すると、変数リストは、自動的にステートメントリストプログラムに拡張されます。

ファンクションの呼び出しを行い、呼び出されたブロックの変数宣言テーブルに IN、OUT、および IN_OUT の宣言がある場合、これらの変数は、呼び出し側ブロックのプログラムに仮パラメータリストとして追加されます。

ファンクションを呼び出す場合、呼び出し側ロジックブロックの実パラメータを仮パラメータに割り付ける必要があります。

IN パラメータは、定数、絶対アドレス、またはシンボルアドレスで指定できます。OUT および IN_OUT パラメータは、絶対アドレスまたはシンボルアドレスで指定します。すべてのアドレスおよび定数は、転送されるデータタイプと必ず互換性をもつようにします。

CALL は、リターンアドレス (セレクトタおよび相対アドレス)、2 つの開いているデータブロックのセレクトタ、MA ビットを B (ブロック) スタックに保存します。さらに、CALL は、MCR の依存性を無効にし、呼び出されるブロックのローカルデータ領域を作成します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	0	0	1	-	0

例: FC6 呼び出しへのパラメータの割り付け

CALL	FC6	
	仮パラメータ	実パラメータ
	NO OF TOOL	:= MW100
	TIME OUT	:= MW110
	FOUND	:= Q0.1
	エラー	:= Q100.0

10.8 SFB の呼び出し

フォーマット

CALL SFB n1, DB n2

説明

この命令は、Siemens が提供する標準ファンクションブロック（SFB）を呼び出すために使用します。CALL 命令は、アドレスが指定された SFB を呼び出します。RLO や他の条件には影響されません。CALL を使ってシステムファンクションブロックを呼び出す場合、インスタンスデータブロックを指定する必要があります。呼び出されたブロックの処理後、呼び出し側ブロックのプログラムにより処理が実行されます。ロジックブロックには、絶対アドレス指定またはシンボルアドレス指定を使用できます。

パラメータの引き渡し（インクリメンタル編集モード）

呼び出し側ブロックは、変数リストを介して、呼び出されたブロックとパラメータのやりとりを行います。有効な CALL 命令を指定すると、変数リストは、自動的にステートメントリストプログラムに拡張されます。

ファンクションブロックの呼び出しを行い、そのブロックの変数宣言テーブルに IN、OUT、および IN_OUT の宣言がある場合、これらの変数は、呼び出し側ブロックのプログラムに仮パラメータリストとして追加されます。

システムファンクションブロックは、実パラメータを指定するだけで呼び出すことができます。実パラメータは、システムファンクションブロックの処理後にインスタンスデータブロックに保存されるので、前回の呼び出しから変更する必要があります。実パラメータがデータブロックの場合、完全な絶対アドレス（たとえば、DB1、DBW2）を指定する必要があります。

IN パラメータは、定数、絶対アドレス、またはシンボルアドレスで指定できます。OUT および IN_OUT パラメータは、絶対アドレスまたはシンボルアドレスで指定します。すべてのアドレスおよび定数は、転送されるデータタイプと必ず互換性をもつようにします。

CALL は、リターンアドレス（セレクトおよび相対アドレス）、2 つの開いているデータブロックのセレクト、MA ビットを B（ブロック）スタックに保存します。さらに、CALL は、MCR の依存性を無効にし、呼び出されるブロックのローカルデータ領域を作成します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	0	0	1	-	0

例

CALL	SFB4,DB4	
	仮パラメータ	実パラメータ
	IN:	I0.1
	PT:	T#20s
	Q:	M0.0
	ET:	MW10

注

システムファンクションブロック CALL には、インスタンスデータブロックが必要です。上記の例では、ブロック SFB4 および DB4 がなければ、呼び出しを実行できません。

10.9 SFC の呼び出し

フォーマット

CALL SFC n

注記

STL エディタで作業している場合、参照 (n) を既存の有効ブロックに関連付ける必要があります。また、使用する前にシンボル名を定義しておく必要があります。

説明

この命令は、Siemens が提供する標準ファンクション (SFC) を呼び出すために使用します。CALL 命令は、アドレスが指定された SFC を呼び出します。RLO や他の条件には影響されません。呼び出されたブロックの処理後、呼び出し側ブロックのプログラムにより処理が実行されます。ロジックブロックには、絶対アドレス指定またはシンボルアドレス指定を使用できます。

パラメータの引き渡し (インクリメンタル編集モード)

呼び出し側ブロックは、変数リストを介して、呼び出されたブロックとパラメータのやりとりを行います。有効な CALL 命令を指定すると、変数リストは、自動的にステートメントリストプログラムに拡張されます。

システムファンクションの呼び出しを行い、このブロックの変数宣言テーブルに IN、OUT、および IN_OUT の宣言がある場合、これらの変数は、呼び出し側ブロックのプログラムに仮パラメータリストとして追加されます。

システムファンクションを呼び出す場合、呼び出し側ロジックブロックの実パラメータを仮パラメータに割り付ける必要があります。

IN パラメータは、定数、絶対アドレス、またはシンボルアドレスで指定できます。OUT および IN_OUT パラメータは、絶対アドレスまたはシンボルアドレスで指定します。すべてのアドレスおよび定数は、転送されるデータタイプと必ず互換性をもつようにします。

CALL は、リターンアドレス (セレクトタおよび相対アドレス)、2 つの開いているデータブロックのセレクトタ、MA ビットを B (ブロック) スタックに保存します。さらに、CALL は、MCR の依存性を無効にし、呼び出されるブロックのローカルデータ領域を作成します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	0	0	1	-	0

例: パラメータなしの SFC の呼び出し

STL	説明
CALL SFC43	//SFC43 を呼び出して、ウォッチドッグタイマを再トリガする (パラメータなし)。

10.10 複数インスタンスの呼び出し

フォーマット

CALL # 変数名

説明

ファンクションブロックのデータタイプを使ってスタティック変数を宣言することで、複数のインスタンスが作成されます。すでに宣言されている複数インスタンスのみが、プログラム要素のカタログに組み込まれます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	0	0	x	x	x

10.11 ライブラリからのブロックの呼び出し

SIMATIC Manager で使用可能なライブラリを使用すれば、次のブロックを選択できます。

- CPU オペレーティングシステム ("標準ライブラリ") に統合されたブロック
- 再使用のために、あらかじめライブラリに保存してあるブロック

10.12 CC 条件付き呼び出し

フォーマット

CC <logic block identifier>

説明

CC <logic block identifier>（条件付きブロック呼び出し）は、RLO=1 の場合に論理ブロックを呼び出します。CC を使用すれば、パラメータ指定せずに FC または FB タイプのロジックブロックを呼び出すことができます。CC は **CALL** 命令と同様に使用されますが、呼び出しプログラムでパラメータを転送できないという相違点があります。この命令は、リターンアドレス（セクタおよび相対アドレス）、2つの現在のデータブロックのセクタ、MA ビットを B（ブロック）スタックに保存し、MCR の依存性を無効にし、呼び出されるブロックのローカルデータ領域を作成し、呼び出されたコードの実行を開始します。ロジックブロックには、絶対アドレス指定またはシンボルアドレス指定を使用できます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	0	0	1	1	0

例

STL	説明
A I 2.0	//入力 I 2.0 の信号状態をチェックする。
CC FC6	//I 2.0 が"1"の場合、ファンクション FC6 を呼び出す。
A M 3.0	//呼び出しファンクションから復帰した直後（I 2.0 = 1）、または I 2.0 が 0 の場合は A I 2.0 ステートメントの直後に実行される。

注

CALL 命令でファンクションブロック（FB）またはシステムファンクションブロック（SFB）を呼び出す場合、ステートメントでインスタンスデータブロック（DB 番号）を指定する必要があります。タイプ"BlockFB"または"BlockFC"の変数を、**CC** 命令と組み合わせて使用することはできません。ステートメントのアドレスに **CC** 命令が含まれる呼び出しにはデータブロックを割り当てることはできないため、この命令は、ブロックパラメータおよび静的なローカルデータがないブロックでしか使用できません。

プログラムエディタは、ラダーロジックプログラム言語からステートメントリストプログラム言語への変換中に、使用しているネットワークに応じて **UC** 命令または **CC** 命令を生成します。プログラムでのエラーを避けるには、**CALL** 命令を使用するとよいでしょう。

10.13 UC 無条件呼び出し

フォーマット

UC <logic block identifier>

説明

UC <logic block identifier>（無条件ブロック呼び出し）は、FC タイプまたは SFC タイプのロジックブロックを呼び出します。UC は CALL 命令と似ていますが、呼び出されたブロックでパラメータを渡すことができないという相違点があります。この命令は、リターンアドレス（セクタおよび相対アドレス）、2 つの現在のデータブロックのセクタ、MA ビットを B（ブロック）スタックに保存し、MCR の依存性を無効にし、呼び出されるブロックのローカルデータ領域を作成し、呼び出されたコードの実行を開始します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	0	0	1	-	0

例 1

STL	説明
UC FC6	//ファンクション FC6 を呼び出す（パラメータなし）。

例 2

STL	説明
UC SFC43	//システムファンクション SFC43 を呼び出す（パラメータなし）。

注

CALL 命令で FB または FC を呼び出す場合、命令には必ずインスタンスデータブロック（DB 番号）を指定する必要があります。タイプ"BlockFB"または"BlockFC"を、UC 命令と組み合わせて使用することはできません。ステートメントのアドレスに UC 命令が含まれる呼び出しにはデータブロックを割り当てることができないため、この命令は、ブロックパラメータおよび静的なローカルデータがないブロックでしか使用できません。

プログラムエディタは、ラダーロジックプログラム言語からステートメントリストプログラム言語への変換中に、使用しているネットワークに応じて UC 命令または CC 命令を生成します。プログラムでのエラーを避けるには、CALL 命令を使用するとよいでしょう。

10.14 MCR (マスタコントロールリレー)

MCR ファンクションの使用方法に関する重要注記



警告

人身障害や機器の損傷を防ぐため、内蔵マスタコントロールリレー装置の代わりに MCR を使って緊急停止機能を実行するようなことはおやめください。

説明

マスタコントロールリレー (MCR) は、リレーラダーロジックのマスタスイッチで、パワーフローをオンおよびオフにします。次のビットロジックでトリガされた命令および転送命令は、MCR に依存します。

- = <bit>
- S <bit>
- R <bit>
- T <byte>, T <word>, T <double word>

T 命令は、バイト、ワイド、ダブルワードで使用され、MCR が 0 の場合にメモリに 0 を書き込みます。S および R 命令は、既存の値を変更せずに残します。命令=は、アドレスビットに"0"を書き込みます。

MCR に依存する命令、および MCR の信号状態に対するその応答

MCR の信号状態	= <bit>	S <bit>, R <bit>	T <byte>, T <word> T <double word>
0 ("オフ")	0 が書き込まれる。 (電圧が失われると、オフ状態のリレーと同様に機能する)	書き込みは行われない。 (電圧が失われると、現在の状態のリレーと同様に機能する)	0 が書き込まれる。 (電圧が失われると、値 0 を生成するコンポーネントと同様に機能する)
1 ("オン")	処理を正常に実行	処理を正常に実行	処理を正常に実行

MCR (- MCR 領域を開始、) MCR - MCR 領域を終了

MCR は、幅 1 ビット、深さ 8 ビットのスタックで制御されます。MCR は、8 つのエントリすべてが 1 である間、通電されます。MCR (命令は、RLO ビットを MCR スタックにコピーします。) MCR 命令は、スタックから最後のエントリを削除し、空いた位置を 1 に設定します。MCR (と) MCR は、必ずペアで使用します。8 個を超える MCR (命令が連続している場合、あるいは、MCR スタックが空のときに) MCR 命令を実行しようとした場合、MCRF エラーメッセージが表示されます。

MCRA - MCR の開始、MCRD - MCR の終了

MCRA と MCRD は、必ずペアで使用します。MCRA と MCRD 間で設定されている命令は、MCR ビットの状態に依存します。MCRA から MCRD の範囲外で設定されている命令は、MCR ビットの状態からは影響を受けません。

呼び出されたブロックで MCRA 命令を使用して、ブロック内でファンクション（FC）およびファンクションブロック（FB）の MCR の依存性を設定する必要があります。

10.15 MCR ファンクションの使用方法に関する重要注記



MCRA によってマスタコントロールリレーが有効化されたブロックに関する注意

- MCR が無効になると、MCR (と) MCR 間のプログラムセグメント内の全割り付け (T, =) により値 0 が書き込まれます。
- MCR (命令の前に RLO が 0 になっている場合には、MCR は無効になります。



危険:

PLC が STOP になったり、未定義のランタイム特性が生成されます!

コンパイラは、VAR_TEMP で定義されているテンポラリ変数のローカルデータへ書き込みアクセスして、アドレスの計算を行います。すなわち、次のコマンドシーケンスを使用すると、PLC が STOP に設定されたり、ランタイム特性が未定義になったりします。

仮パラメータアクセス

- タイプ STRUCT、UDT、ARRAY、STRING の複合 FC パラメータのコンポーネントにアクセス
- マルチプルインスタンス能力を持つブロック (バージョン 2 ブロック) の IN_OUT 領域からの、STRUCT、UDT、ARRAY、STRING タイプの複合 FB パラメータで構成されるコンポーネントへのアクセス
- アドレスが 8180.0 を超える場合の、マルチプルインスタンス能力を持つファンクションブロック (バージョン 2 ブロック) のパラメータへのアクセス
- マルチプルインスタンス能力を持つファンクションブロック (バージョン 2 ブロック) の DB0 を開く BLOCK_DB パラメータへのアクセス さらにデータアクセスが実行されると、CPU は STOP になります。T 0、C 0、FC0、または FB0 も、必ず TIMER、COUNTER、BLOCK_FC、および BLOCK_FB に使用されます。

パラメータの引き渡し

- パラメータの引き渡しに使用される呼び出し

LAD/FBD

- RLO = 0 で開始する Ladder または FBD の T ブランチおよび中間出力

対策

上記のコマンドを MCR から独立させます。

1. 該当するステートメントまたはネットワークの前に MCRD 命令を使用して、マスタコントロールリレーを無効にします。
2. 該当するステートメントまたはネットワークの後に MCRA 命令を使用して、マスタコントロールリレーを再度有効にします。

10.16 MCR (MCR オン

MCR ファンクションの使用方法に関する重要注記

フォーマット

MCR (

説明

MCR ((MCR 領域を開く) は、RLO を MCR スタックに保存し、MCR 領域を開きます。MCR 領域は、**MCR 命令** (およびこれに対応する) **MCR** の間にある命令です。**MCR (** (命令は必ず) **MCR** と対で使用します。

RLO が 1 の場合、MCR は"オン"になります。通常、この MCR ゾーン内の MCR 依存命令が実行されます。

If RLO が 0 のとき、MCR は"オフ"になります。

MCR ゾーン内の MCR に依存する命令は、下記の表に従って実行します。

MCR ビット状態に依存する命令

MCR の信号状態	= <bit>	S <bit>, R <bit>	T <byte>, T <word> T <double word>
0 ("オフ")	0 が書き込まれる。 (電圧が失われると、オフ状態のリレーと同様に機能する)	書き込みは行われない。 (電圧が失われると、現在の状態のリレーと同様に機能する)	0 が書き込まれる。 (電圧が失われると、値 0 を生成するコンポーネントと同様に機能する)
1 ("オン")	処理を正常に実行	処理を正常に実行	処理を正常に実行

MCR (命令および) MCR 命令はネスト構造にできます。命令を 8 個までネストできます。スタックエントリの最大数は 8 です。スタックがいっぱいの状態で MCR (を実行すると、MCR スタック障害 (MCRF) が発生します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

10.16 MCR (MCR オン

例

STL	説明
MCR A I 1.0	//MCR 領域を有効にする。
MCR (	//RLO を MCR スタックに保存し、MCR 領域を開く。RLO が 1 のとき (I 1.0 ="1") 、 //MCR は"オン"になり、RLO が 0 のとき (I 1.0 ="0") 、MCR は"オフ"になる。
A I 4.0	
= Q 8.0	//MCR が"オフ"の場合、I 4.0 の状態に関係なく Q 8.0 は"0"に設定される。
L MW20	
T QW10	//MCR が"オフ"の場合、QW10 に"0"が転送される。
) MCR	//MCR 領域を終了する。
MCRD	//MCR 領域を無効にする。
A I 1.1	
= Q 8.1	//これらの命令は、MCR 領域外にあるため、MCR ビットに依存しない。

10.17) MCR MCR 終了

MCR ファンクションの使用方法に関する重要注記

フォーマット

) MCR

説明

) MCR (MCR 領域の終了) は、MCR スタックのエントリを削除し、MCR 領域を無効にします。前の MCR スタック位置は解放され、1 に設定されます。命令 MCR (は、必ず命令) MCR と組み合わせて使用します。) MCR 命令を空のスタックを使って実行すると、MCR スタック障害 (MCRF) が発生します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	1	-	0

例

STL	説明
MCRA	//MCR 領域を有効にする。
A I 1.0	
MCR (	//RLO を MCR スタックに保存し、MCR 領域を開く。RLO が 1 のとき (I 1.0 ="1") 、
	//MCR は"オン"になり、RLO が 0 のとき (I 1.0 ="0") 、MCR は"オフ"になる。
A I 4.0	
= Q 8.0	//MCR が"オフ"の場合、I 4.0 の状態に関係なく Q 8.0 は"0"に設定される。
L MW20	
T QW10	//MCR が"オフ"の場合、QW10 に"0"が転送される。
) MCR	//MCR 領域を終了する。
MCRD	//MCR 領域を無効にする。
A I 1.1	
= Q 8.1	//これらの命令は、MCR 領域外にあるため、MCR ビットに依存しない。

10.18 MCRA MCR 領域の有効化

MCR ファンクションの使用方法に関する重要注記

フォーマット

MCRA

説明

MCRA（マスタコントロールリレー開始）は、後続する命令に対し MCR の依存性を起動します。MCRA は、必ず MCRD（マスタコントロールリレー終了）と対で使用します。MCRA と MCRD 間で設定済みの命令は、MCR ビットの信号状態に依存します。

命令は、ステータスワードビットに関係なく実行されます。ステータスビットも影響されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
MCRA	//MCR 領域を有効にする。
A I 1.0	
MCR (	//RLO を MCR スタックに保存し、MCR 領域を開く。RLO が 1 のとき (I 1.0 ="1") 、 //MCR は"オン"になり、RLO が 0 のとき (I 1.0 ="0") 、MCR は"オフ"になる。
A I 4.0	
= Q 8.0	//MCR が"オフ"の場合、I 4.0 の状態に関係なく Q 8.0 は"0"に設定される。
L MW20	
T QW10	//MCR が"オフ"の場合、QW10 に"0"が転送される。
) MCR	//MCR 領域を終了する。
MCRD	//MCR 領域を無効にする。
A I 1.1	
= Q 8.1	//これらの命令は、MCR 領域外にあるため、MCR ビットに依存しない。

10.19 MCRD MCR 領域の無効化

MCR ファンクションの使用方法に関する重要注記

フォーマット

MCRD

説明

MCRD（マスタコントロールリレー終了）は、後続する命令に対し MCR の依存性を無効にします。命令 MCRA（マスタコントロールリレー開始）は、必ず MCRD（マスタコントロールリレー終了）と対で使用します。MCRA と MCRD 間で設定済みの命令は、MCR ビットの信号状態に依存します。

命令は、ステータスワードビットに関係なく実行されます。ステータスビットも影響されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
MCRA	//MCR 領域を有効にする。
A I 1.0	
MCR (	//RLO を MCR スタックに保存し、MCR 領域を開く。RLO が 1 のとき (I 1.0 ="1") 、 //MCR は"オン"になり、RLO が 0 のとき (I 1.0 ="0") 、MCR は"オフ"になる。
A I 4.0	
= Q 8.0	//MCR が"オフ"の場合、I 4.0 の状態に関係なく Q 8.0 は"0"に設定される。
L MW20	
T QW10	//MCR が"オフ"の場合、QW10 に"0"が転送される。
) MCR	//MCR 領域を終了する。
MCRD	//MCR 領域を無効にする。
A I 1.1	
= Q 8.1	//これらの命令は、MCR 領域外にあるため、MCR ビットに依存しない。

11 シフト命令および回転命令

11.1 シフト命令

11.1.1 シフト命令の概要

説明

シフト命令を使用すれば、アキュムレータ 1 の下位ワードの内容またはアキュムレータ全体の内容をビット単位で左右に移動できます（「CPU レジスタ」も参照してください）。n ビットだけ左方向にシフトすると、アキュムレータの内容に 2^n が乗算されます。n ビットだけ右方向にシフトすると、アキュムレータの内容が 2^n で除算されます。たとえば、10 進数の 3 に相当する 2 進数を 3 ビット左へシフトすると、10 進数の 24 に相当する 2 進数が得られます。また、10 進数の 16 に相当する 2 進数を 2 ビット右へシフトすると、10 進数の 4 に相当する 2 進数が得られます。

シフト命令の後に数字を付けて、または、アキュムレータ 2 の下位ワードの下位バイトの値で、シフトさせるビット数を指定します。シフト命令によって空になったビットの桁には、0 または符号ビットの信号状態（0＝正、1＝負）が入ります。最後にシフトされたビットは、ステータスワードの CC 1 ビットへロードされます。ステータスワードの CC 0 および OV ビットは 0 にリセットされます。ジャンプ命令を使用して、CC 1 ビットを評価することができます。シフト演算は、無条件に実行されます。これらは、論理演算結果に影響を与えません。

以下のシフト命令が使用可能です。

- SSI 符号付きシフト整数（16 ビット）
- 符号付きシフトダブル整数（32 ビット）
- SLW 左シフトワード（16 ビット）
- SRW 右シフトワード（16 ビット）
- SLD 左シフトダブルワード（32 ビット）
- SRD 右シフトダブルワード（32 ビット）

11.1 シフト命令

11.1.2 SSI 符号付きシフト整数（16 ビット）

フォーマット

SSI
SSI <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～15。

説明

SSI（符号付き倍長整数の右シフト）は、ACCU 1 の内容だけを右方向に 1 ビットずつシフトします。シフト命令によって空になるビット桁には符号ビット（ビット 15）の信号状態が挿入されます。最後にシフトされたビットは、ステータスワードのビット CC 1 へロードされます。シフトするビットの桁の数値は、アドレス<number>または ACCU 2-L-L で指定します。

SSI <number>: シフト桁数は、アドレス<number>で指定されます。許容数値範囲は 0～15 です。<number>が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number>が 0 の場合、シフト命令は **NOP** 操作と見なされます。

SSI: シフト桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。16 より大きなシフト数では、常に同じ結果になります。ACCU 1 = 16#0000、CC 1 = 0 または、ACCU 1 = 16#FFFF、CC 1 = 1 です。シフト数が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。シフト数が 0 の場合、シフト命令は **NOP** 操作と見なされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
SSI 6 命令実行前の値	0101	1111	0110	0100	1001	1101	0011	1011
SSI 6 命令実行後の値	0101	1111	0110	0100	1111	1110	0111	0100

例 1

STL	説明
L MW4	//値を ACCU 1 にロードする。
SRW 6	//ACCU 1 の符号付きビットを右方向に 6 桁シフトする。
T MW8	//結果を MW8 へ転送する。

例 2

STL	説明
L +3	//数値+3 を ACCU 1 にロードする。
L MW20	//ACCU 1 の値を ACCU 2 にロードする。MW20 の値を ACCU 1 にロードする
SRW	//ACCU 2-L-L の値がシフト桁数になる。=>ACCU 1-L の符号付きビットを右方向に 3 桁シフトする。 //空の桁には符号ビットの状態を充てんする。
JP NEXT	//最後にシフトアウトしたビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.1 シフト命令

11.1.3 符号付きシフトダブル整数（32 ビット）

フォーマット

SSD
SSD <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～32。

説明

SSD（符号付き右シフトダブル整数）は、ACCU 1 の内容全体を右へ 1 ビットずつ移動します。シフト命令によって空になるビット位置には符号ビットの信号状態が挿入されます。最後にシフトされたビットは、ステータスワードのビット CC 1 へロードされます。シフトするビットの桁の数値は、アドレス<number>または ACCU 2-L-L で指定します。

SSD <number>: シフト桁数は、アドレス<number>で指定されます。許容数値範囲は 0～32 です。<number>が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number>が 0 の場合、シフト命令は **NOP** 操作と見なされます。

SSD: シフト桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。32 より大きなシフト数では、常に同じ結果になります。ACCU 1 = 32#00000000、CC 1 = 0 または、ACCU 1 = 32#FFFFFFF、CC 1 = 1 です。シフト数が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。シフト数が 0 の場合、シフト命令は **NOP** 操作と見なされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
SSD 7 命令実行前の値	1000	1111	0110	0100	0101	1101	0011	1011
SSD 7 命令実行後の値	1111	1111	0001	1110	1100	1000	1011	1010

例 1

STL	説明	
L	MD4	//値を ACCU 1 にロードする。
SSD	7	//符号に応じて、ACCU 1 のビットを右方向に 7 桁シフトする。
T	MD8	//結果を MD8 に転送する。

例 2

STL	説明	
L	+3	//数値+3 を ACCU 1 にロードする。
L	MD20	//ACCU 1 の値を ACCU 2 にロードする。MD20 の値を ACCU 1 にロードする
SSD		//ACCU 2-L-L の値がシフト桁数になる。=>ACCU 1 の符号付きビットを右方向に 3 桁シフトする。 //空の桁には符号ビットの状態を充てんする。
JP	NEXT	//最後にシフトアウトしたビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.1 シフト命令

11.1.4 SLW 左シフトワード (16 ビット)

フォーマット

SLW
SLW <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～15。

説明

SLW (ワード左シフト) は、ACCU 1-L の内容だけを左方向に 1 ビットずつシフトします。シフト命令によって空になるビット位置には 0 が挿入されます。最後にシフトされたビットは、ステータスワードのビット CC 1 へロードされます。シフトするビットの桁の数値は、アドレス<number>または ACCU 2-L-L で指定します。

SLW <number>: シフト桁数は、アドレス<number>で指定されます。許容数値範囲は 0～15 です。<number>が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number>が 0 の場合、シフト命令は **NOP** 操作と見なされます。

SLW: シフト桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。16 より大きなシフト数では、常に同じ結果になります。ACCU 1 = 0、CC 1 = 0、CC 0 = 0、OV = 0 です。0 < シフト数 ≤ 16 の場合、ステータスワードのビット 0 および OV は 0 にリセットされます。シフト数が 0 の場合、シフト命令は **NOP** 操作と見なされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
SLW 5 命令実行前の値	0101	1111	0110	0100	0101	1101	0011	1011
SLW 5 命令実行後の値	0101	1111	0110	0100	1010	0111	0110	0000

例 1

STL	説明	
L	MW4	//値を ACCU 1 にロードする。
SLW	5	//ACCU 1 のビットを左方向に 5 桁シフトする。
T	MW8	//結果を MW8 へ転送する。

例 2

STL	説明	
L	+3	//数値+3 を ACCU 1 にロードする。
L	MW20	//ACCU 1 の値を ACCU 2 にロードする。MW20 の値を ACCU 1 にロードする
SLW		//ACCU 2-L-L の値がシフト桁数になる。=>ACCU 1-L 内のビットを左方向に 3 桁シフトする。
JP	NEXT	//最後にシフトアウトしたビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.1 シフト命令

11.1.5 SRW 右シフトワード (16 ビット)

フォーマット

SRW
SRW <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～15。

説明

SRW (ワード右シフト) は、ACCU 1-L の内容だけを右方向に 1 ビットずつシフトします。シフト命令によって空になるビット位置には 0 が挿入されます。最後にシフトされたビットは、ステータスビット CC 1 へロードされます。シフトするビットの桁の数値は、アドレス<number>または ACCU 2-L-L で指定します。

SRW <number>: シフト桁数は、アドレス<number>で指定されます。許容数値範囲は 0～15 です。<number>が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number>が 0 の場合、シフト命令は **NOP** 操作と見なされます。

SRW: シフト桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。16 より大きなシフト数では、常に同じ結果になります。ACCU 1 = 0、CC 1 = 0、CC 0 = 0、OV = 0 です。0 < シフト数 ≤ 16 の場合、ステータスワードのビット 0 および OV は 0 にリセットされます。シフト数が 0 の場合、シフト命令は **NOP** 操作と見なされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
SRW 6 命令実行前の値	0101	1111	0110	0100	0101	1101	0011	1011
SRW 6 命令実行後の値	0101	1111	0110	0100	0000	0001	0111	0100

例 1

STL	説明	
L	MW4	//値を ACCU 1 にロードする。
SRW	6	//ACCU 1-L のビットを右方向に 6 桁シフトする。
T	MW8	//結果を MW8 へ転送する。

例 2

STL	説明	
L	+3	//数値+3 を ACCU 1 にロードする。
L	MW20	//ACCU 1 の値を ACCU 2 にロードする。MW20 の値を ACCU 1 にロードする
SRW		//ACCU 2-L-L の値がシフト桁数になる。=>ACCU 1-L 内のビットを右方向に 3 桁シフトする。
SPP	NEXT	//最後にシフトアウトしたビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.1 シフト命令

11.1.6 SLD 左シフトダブルワード (32 ビット)

フォーマット

SLD
SLD <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～32。

説明

SLD (左シフトダブルワード) は、ACCU 1 の内容全体を左へ 1 ビットずつ移動します。シフト命令によって空になるビット位置には 0 が挿入されます。最後にシフトされたビットは、ステータスワードのビット CC 1 へロードされます。シフトするビットの桁の数値は、アドレス<number>または ACCU 2-L-L で指定します。

SLD <number>: シフト桁数は、アドレス<number>で指定されます。許容数値範囲は 0～32 です。<number>が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number>が 0 の場合、シフト命令は **NOP** 操作と見なされます。

SLD: シフト桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。32 より大きなシフト数では、常に同じ結果になります。ACCU 1=0、CC 1=0、CC 0=0、OV=0 です。0<シフト数 ≤ 32 の場合、ステータスワードのビット 0 および OV は 0 にリセットされます。シフト数が 0 の場合、シフト命令は **NOP** 操作と見なされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	X	X	X	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
SLD 5 命令実行前の値	0101	1111	0110	0100	0101	1101	0011	1011
SLD 5 命令実行後の値	1110	1100	1000	1011	1010	0111	0110	0000

例 1

STL	説明	
L	MD4	//値を ACCU 1 にロードする。
SLD	5	//ACCU 1 のビットを左方向に 5 桁シフトする。
T	MD8	//結果を MD8 に転送する。

例 2

STL	説明	
L	+3	//数値+3 を ACCU 1 にロードする。
L	MD20	//ACCU 1 の値を ACCU 2 にロードする。MD20 の値を ACCU 1 にロードする
SLD		//ACCU 2-L-L の値がシフト桁数になる。=>ACCU 1 内のビットを左方向に 3 桁シフトする。
JP	NEXT	//最後にシフトアウトしたビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.1 シフト命令

11.1.7 SRD 右シフトダブルワード (32 ビット)

フォーマット

SRD
SRD <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～32。

説明

SRD (右シフトダブルワード) は、ACCU 1 の内容全体を右へ 1 ビットずつ移動します。シフト命令によって空になるビット位置には 0 が挿入されます。最後にシフトされたビットは、ステータスワードのビット CC 1 へロードされます。シフトするビットの桁の数値は、アドレス<number>または ACCU 2-L-L で指定します。

SRD <number>: シフト桁数は、アドレス<number>で指定されます。許容数値範囲は 0～32 です。<number>が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number>が 0 の場合、シフト命令は **NOP** 操作と見なされます。

SRD: シフト桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。32 より大きなシフト数では、常に同じ結果になります。ACCU 1=0、CC 1=0、CC 0=0、OV=0 です。0<シフト数 ≤ 32 の場合、ステータスワードのビット 0 および OV は 0 にリセットされます。シフト数が 0 の場合、シフト命令は **NOP** 操作と見なされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
SRD 7 命令実行前の値	0101	1111	0110	0100	0101	1101	0011	1011
SRD 7 命令実行後の値	0000	0000	1011	1110	1100	1000	1011	1010

例 1

STL	説明	
L	MD4	//値を ACCU 1 にロードする。
SRD	7	//ACCU 1 のビットを右方向に 7 桁シフトする。
T	MD8	//結果を MD8 に転送する。

例 2

STL	説明	
L	+3	//数値+3 を ACCU 1 にロードする。
L	MD20	//ACCU 1 の値を ACCU 2 にロードする。MD20 の値を ACCU 1 にロードする
SRD		//ACCU 2-L-L の値がシフト桁数になる。=>ACCU 1 内のビットを右方向に 3 桁シフトする。
JP	NEXT	//最後にシフトアウトしたビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.2 回転命令

11.2.1 回転命令の概要

説明

循環命令を使用すれば、アキュムレータ 1 の全内容をビット単位で左右に循環できます（「CPU レジスタ」も参照してください）。回転命令は、セクション 14.1 で説明するシフト機能と同様のファンクションをトリガします。ただし、回転によって空いたビットの桁には、アキュムレータから押し出されたビットの信号状態が入ります。

循環命令の後に数字を付けて、またはアキュムレータ 2 の下位ワードの下位バイトの値で、循環させるビットの桁数を指定します。命令によっては、ステータスワードの CC 1 ビットを使用して循環が実行されます。ステータスワードの CC 0 ビットは、0 にリセットされます。

循環命令には以下のものがあります。

- RLD 左循環ダブルワード（32 ビット）
- RRD 右循環ダブルワード（32 ビット）
- RLDA CC 1 により ACCU 1 を左へ循環（32 ビット）
- RRDA CC 1 により ACCU 1 を右へ循環（32 ビット）

11.2.2 RLD 左循環ダブルワード（32 ビット）

フォーマット

RLD
RLD <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～32。

説明

RLD（左循環ダブルワード）は、ACCU1 の内容全体を左へ 1 ビットずつ移動します。回転命令によって空いたビットの桁には、ACCU 1 から押し出されたビットの信号状態が格納されます。最後に回転されたビットは、ステータスビット CC 1 へロードされます。回転するビットの桁の数値は、アドレス <number> または ACCU 2-L-L で指定します。

RLD <number>: 循環桁数は、アドレス <number> で指定されます。許容数値範囲は 0～32 です。<number> が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number> が 0 の場合、循環命令は **NOP** 操作とみなされます。

RLD: 循環桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。ACCU 2-L-L の内容が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。移動の数が 0 の場合、循環命令は **NOP** 操作と見なされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31 ...	..	..	... 16	15 ...	..	..	... 0
RLD 4 命令実行前の値	0101	1111	0110	0100	0101	1101	0011	1011
RLD 4 命令実行後の値	1111	0110	0100	0101	1101	0011	1011	0101

11.2 回転命令

例 1

STL	説明	
L	MD2	//値を ACCU 1 にロードする。
RLD	4	//ACCU 1 のビットを左へ 4 桁循環する。
T	MD8	//結果を MD8 に転送する。

例 2

STL	説明	
L	+3	//数値+3 を ACCU 1 にロードする。
L	MD20	//ACCU 1 の値を ACCU 2 にロードする。MD20 の値を ACCU 1 にロードする
RLD		//ACCU 2-L-L の値が循環桁数になる。 //=>ACCU 1 内のビットを左方向に 3 桁循環する。
JP	NEXT	//最後に循環したビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.2.3 RRD 右循環ダブルワード (32 ビット)

フォーマット

RRD
RRD <number>

アドレス	データタイプ	説明
<number>	整数、符号なし	移動するビット位置の数。有効な数は 0～32。

説明

RRD (右循環ダブルワード) は、ACCU 1 の内容全体を右へ 1 ビットずつ移動します。回転命令によって空いたビットの桁には、ACCU 1 から押し出されたビットの信号状態が格納されます。最後に回転されたビットは、ステータスビット CC 1 へロードされます。回転するビットの桁の数値は、アドレス <number> または ACCU 2-L-L で指定します。

RRD <number>: 循環桁数は、アドレス <number> で指定されます。許容数値範囲は 0～32 です。<number> が 0 より大きい場合、ステータスワードビット CC 0 および OV は 0 にリセットされます。<number> 0 の場合、循環命令は NOP 操作と見なされます。

RRD: 循環桁数は、ACCU 2-L-L 内の値で指定されます。可能な数値の範囲は 0～255 です。ACCU 2-L-L の内容が 0 より大きい場合、ステータスワードビットは 0 にリセットされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	x	x	-	-	-	-	-

例

目次	ACCU1-H				ACCU1-L			
ビット	31...	...	...	...16	15...	...	...	...0
RRD 4 命令実行前の値	0101	1111	0110	0100	0101	1101	0011	1011
RRD 4 命令実行後の値	1011	0101	1111	0110	0100	0101	1101	0011

11.2 回転命令

例 1

STL	説明	
L	MD2	//値を ACCU 1 にロードする。
RRD	4	//ACCU 1 のビットを右方向に 4 桁循環する。
T	MD8	//結果を MD8 に転送する。

例 2

STL	説明	
L	+3	//数値+3 を ACCU 1 にロードする。
L	MD20	//ACCU 1 の値を ACCU 2 にロードする。MD20 の値を ACCU 1 にロードする
RRD		//ACCU 2-L-L の値が循環桁数になる。 //=>ACCU 1 内のビットを右方向に 3 桁循環する。
JP	NEXT	//最後に循環したビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。

11.2.4 RLDA CC 1 により ACCU 1 を左へ循環 (32 ビット)

フォーマット

RLDA

説明

RLDA (CC 1 による左循環ダブルワード) は、CC 1 によって、ACCU 1 の内容全体を左へ 1 ビットずつ移動します。ステータスワードビット CC 0 と OV は 0 にリセットされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	IFC
書き込みの内容:	-	x	0	0	-	-	-	-	-

例

目次	CC 1	ACCU1-H				ACCU1-L			
ビット		31 ...	..	..	... 16	15 ...	..	..	... 0
RLDA 命令実行前の値	X	0101	1111	0110	0100	0101	1101	0011	1011
RLDA 命令実行後の値	0	1011	1110	1100	1000	1011	1010	0111	011X
(X = 0 または 1、CC 1 の前回の信号状態)									

STL	説明	
L MD2	//MD2 の値を ACCU 1 にロードする	
RLDA	//CC 1 により、ACCU 1 のビットを左方向に 1 桁循環する。	
JP NEXT	//最後に循環したビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。	

11.2 回転命令

11.2.5 RRDA CC 1 により ACCU 1 を右へ循環 (32 ビット)

フォーマット

RRDA

説明

RRDA (CC 1 による右循環ダブルワード) は、ACCU 1 の内容全体を右へ 1 ビットずつ移動します。ステータスワードビット CC 0 と OV は 0 にリセットされます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	0	0	-	-	-	-	-

例

目次	CC 1	ACCU1-H				ACCU1-L			
ビット		31...	..	..	...16	15...	..	..	...0
RRDA 命令実行前の値	X	0101	1111	0110	0100	0101	1101	0011	1011
RRDA 命令実行後の値	1	X010	1111	1011	0010	0010	1110	1001	1101
(X = 0 または 1、CC 1 の前回の信号状態)									

STL	説明	
L MD2	//MD2 の値を ACCU 1 にロードする	
RRDA	//CC 1 により、ACCU 1 のビットを右方向に 1 桁循環する。	
JP NEXT	//最後に循環したビット (CC 1) が 1 の場合、NEXT ジャンプラベルへジャンプする。	

12 タイマ命令

12.1 タイマ命令の概要

説明

適切な時間の設定と選択については、「メモリ内のタイマの位置とタイマのコンポーネント」を参照してください。

タイマ命令には以下のものがあります。

- FR タイマの有効化（フリー）
- L 現在のタイマ値を ACCU 1 に整数でロード
- LC 現在のタイマ値を BCD として ACCU 1 にロード
- R リセットタイマ
- SD オンディレイタイマ
- SE 拡張パルスタイマ
- SF オフディレイタイマ
- SP パルスタイマ
- SS 保持型オンディレイタイマ

12.2 メモリ内のタイマの場所およびタイマのコンポーネント

メモリ内の領域

タイマは、CPU のメモリ内でタイマ用に確保された領域を使用します。このメモリ領域では、タイマアドレスごとに 16 ビットワードが 1 つ確保されます。ラダーロジック命令セットでは、256 個のタイマがサポートされます。使用可能なタイマワードの数を設定する場合には、CPU の技術情報を参照してください。

以下の機能を使って、タイマのメモリ領域へアクセスできます。

- タイマ命令
- クロックタイミングによるタイマワードの更新。この機能は、CPU が RUN モードになっているときに使用可能で、タイマ値が 0 になるまで、指定されたタイムベースで単位時間ずつ設定値を減少します。

時間値

タイマワードのビット 0~9 には、時間値がバイナリコードで格納されます。時間値では、多数の単位が指定されます。時間を更新すると、タイムベースで指定された間隔で時間値が 1 単位ずつ減っていきます。タイマ値は、0 になるまで減少し続けます。タイマ値は、2 進数表記、16 進数表記、または 2 進化 10 進数 (BCD) 表記で、アキュムレータ 1 の下位ワードへロードできます。

次のいずれかのフォーマットを使用して、時間値を事前にロードすることができます。

- W#16#txyz
 - ここで、t=タイムベース（つまり、時間間隔または分解能）
 - xyz = 2 進化 10 進フォーマットの時間値
- S5T#aH_bM_cS_dMS
 - ここで、H = 時間、M = 分、S = 秒、および MS = ミリ秒、a、b、c、d はユーザー変数です。
 - タイムベースは自動的に選択され、タイマ値は選択されたタイムベースごとに一単位ずつ減少します。

入力できる最大タイマ値は、9,990 秒または 2H_46M_30S です。

タイムベース

タイマワードのビット 12 とビット 13 には、タイムベースがバイナリコードで格納されます。タイムベースは、時間値が一単位ずつ減少する間隔を定義します。最小タイムベースは 10 ms で、最大タイムベースは 10 s です。

タイムベース	タイムベースの 2 進コード
10 ms	00
100 ms	01
1 秒	10
10 秒	11

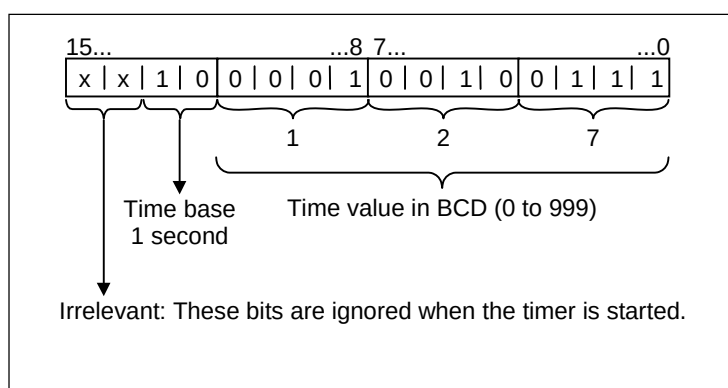
2h46m30s を超える値は無効です。分解能が有効範囲よりも大きい場合（たとえば、2h10ms）、有効範囲になるように、その値に対して切り捨てが実行されます。S5TIME の一般形式では、値の範囲と分解能について次のような制限があります。

分解能	範囲
0.01 秒	10MS ~ 9S_990MS
0.1 秒	100MS~1M_39S_900MS
1 秒	1S~16M_39S
10 秒	10S~2H_46M_30S

ACCU 1 のビット構成

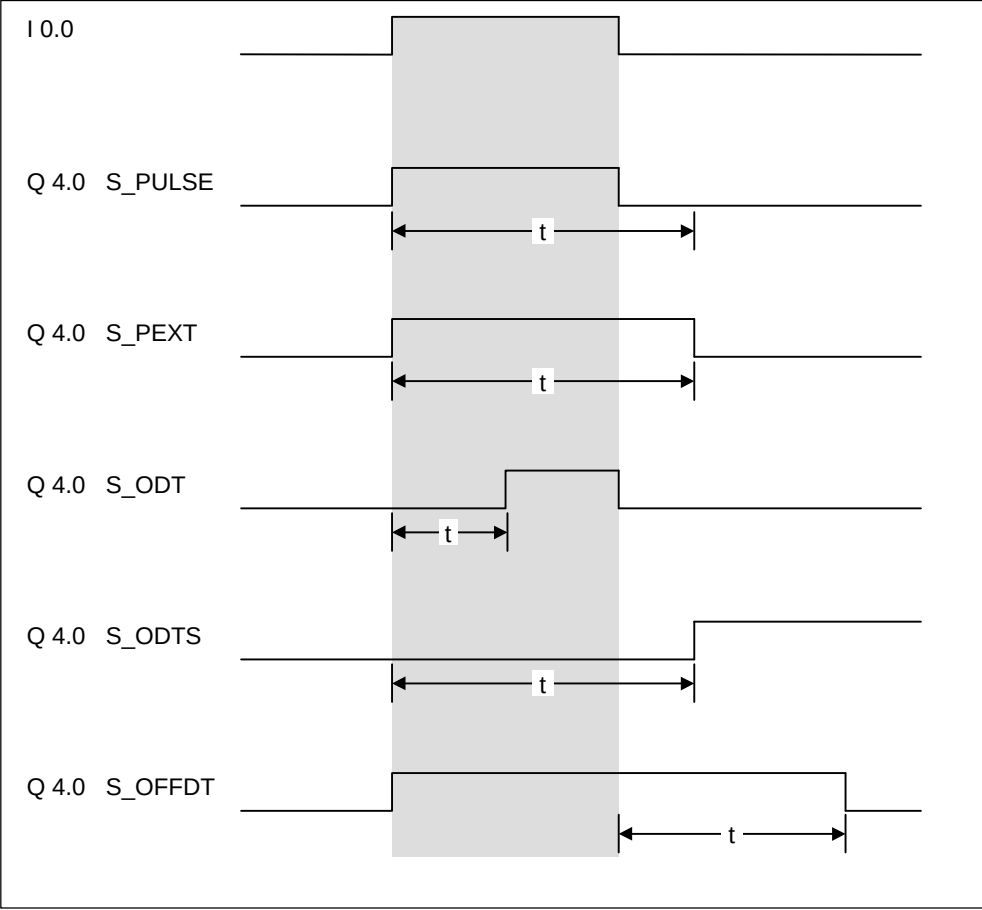
タイマを起動すると、ACCU1 の内容が時間値として使用されます。ACCU1-L のビット 0~11 には、2 進化 10 進数フォーマットで時間値が保持されます。2 進化 10 進数フォーマットつまり BCD フォーマットでは、4 ビットセットごとに、1 つの 10 進値に対応する 2 進数が指定されます。ビット 12 とビット 13 には、タイムベースがバイナリコードで格納されます。

以下の図に、時間値 127 とタイムベース 1 秒がロードされた ACCU1-L の内容を示します。



適切なタイマの選択

この概要を参考にすれば、タイミングジョブに適切なタイマを選択できます。



タイマ	説明
S_PULSE パルスタイマ	出力信号が 1 となる最大時間は、プログラムされたタイマ値 t と同じです。入力信号が 0 に変わる場合は、出力信号が短い間だけ 1 になります。
S_PEXT 拡張パルスタイマ	入力信号が 1 である時間に関係なくプログラムされた時間 t の間、出力信号は 1 です。
S_ODT オンディレイタイマ	プログラムされた時間 t が経過し、入力信号が 1 である場合にだけ、出力信号が 1 に変わります。
S_ODTS 拡張オンディレイタイマ	プログラムされた時間 t が経過すると、入力信号が 0 のままかどうかに関係なく、出力信号が 1 になります。
S_OFFDT オフディレイタイマ	入力信号が 1 に変わる場合、またはタイマが作動している間、出力信号は 1 になります。入力信号が 1 から 0 に変わると、このタイマが起動します。

12.3 FR タイマの有効化（フリー）

フォーマット

FR <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲は CPU により異なる。

命令の説明

RLO が"0"から"1"になると、**FR <timer>**により、指定されたタイマの起動に使用するエッジ検出フラグがクリアされます。有効化命令（FR）の前に RLO ビットが 0 から 1 になると、タイマは使用可能になります。

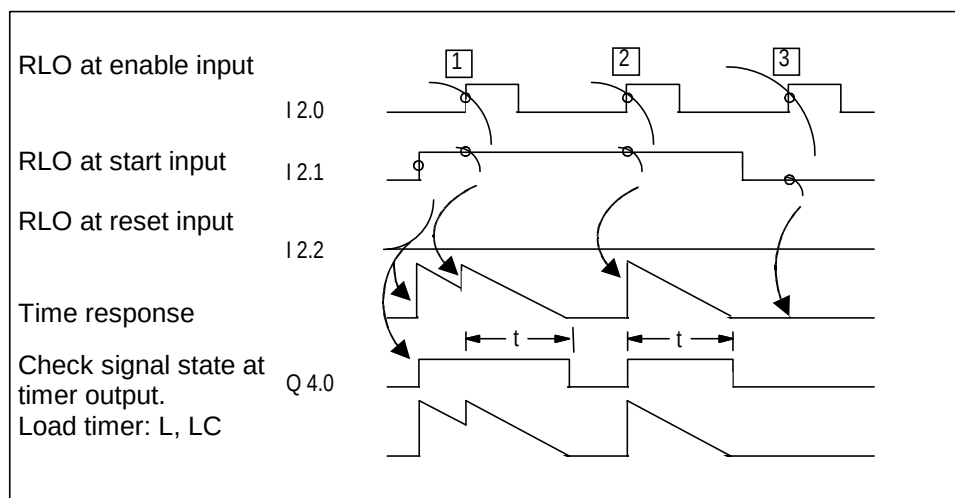
タイマを起動する場合や、通常のタイマ命令に対しては、タイマの有効化は必要ありません。有効化は、実行中のタイマを再トリガする場合、つまり、タイマを再起動する場合に使用します。再起動は、RLO が 1 の状態で起動命令の処理を継続する場合に可能です。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.0	
FR T1	//タイマ T1 を有効にする。
A I 2.1	
L S5T#10s	//ACCU 1 に 10 秒をプリセットする。
SI T1	//タイマ T1 をパルスタイマとして起動する。
A I 2.2	
R T1	//タイマ T1 をリセットする。
A T1	//タイマ T1 の信号状態をチェックする。
= Q 4.0	
L T1	//タイマ T1 の現在の時間値を 2 進数形式でロードする。
T MW10	



t = programmed time interval

- (1) タイマの作動中、有効化入力時に RLO が 0 から 1 に変わると、タイマは完全に再起動します。再起動には、プログラムされた時間が現在値として使用されます。有効化入力時に RLO が 1 から 0 に変わっても、何の影響も及ぼしません。
- (2) タイマが作動していない間、有効化入力時に RLO が 0 から 1 へ変わり、起動入力時に RLO がまだ 1 である場合、タイマは、時間がプログラムされたパルスタイマとしても起動します。
- (3) 起動入力時に RLO がまだ 0 になっている間に有効化入力時に RLO が 0 から 1 に変わっても、何の影響も及ぼしません。

12.4 L 現在のタイマ値を ACCU 1 に整数でロード

フォーマット

L <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲は CPU により異なる。

命令の説明

L <timer>は、指定されたタイマワードの現在のタイマ値（タイムベースは除く）を 2 進整数として ACCU 1-L にロードします。これは、ACCU 1 の内容が ACCU 2 に保存された後に実行されます。

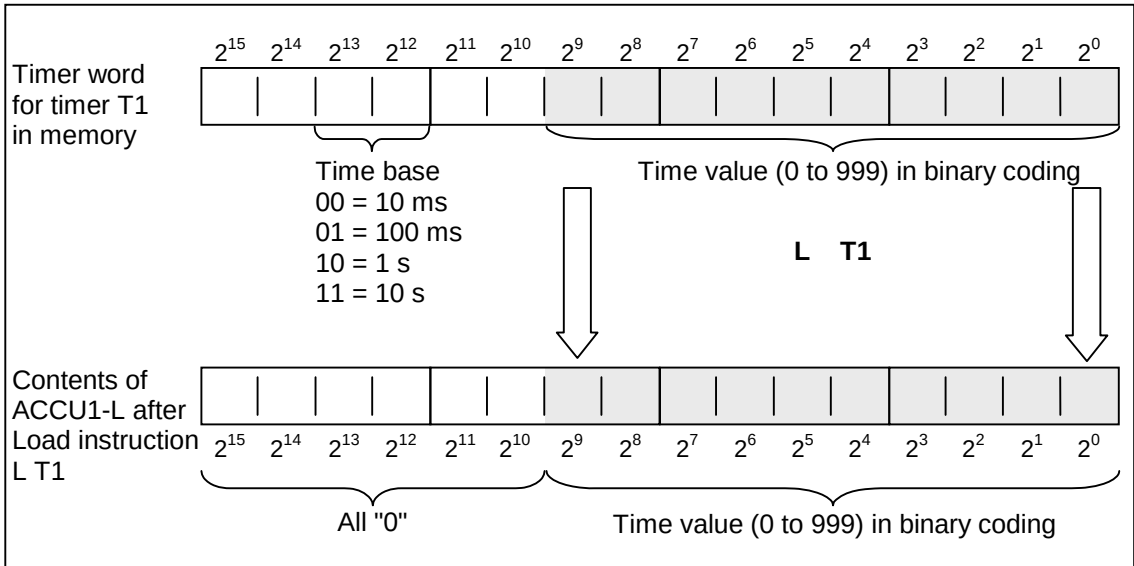
ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

12.4 L 現在のタイマ値を ACCU 1 に整数でロード

例

STL	説明
L T1	//ACCU 1-L に、タイマ T1 の現在のタイマ値を 2 進数形式でロードします。



注記

L <timer>を使用すると、現在のタイマ値の 2 進数だけが ACCU1-L にロードされ、これはタイムベースではありません。ロードされる時間は、初期値から、タイマの起動以降の満了時間を引いたものになります。

12.5 LC 現在のタイマ値をBCDとしてACCU 1にロード

フォーマット

LC <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲はCPUにより異なる。

命令の説明

LC <timer> は、指定されたタイマワードの現在のタイマ値とタイムベースを2進10進数（BCD）でACCU 1にロードします。これは、ACCU 1の内容がACCU 2に保存された後に実行されます。

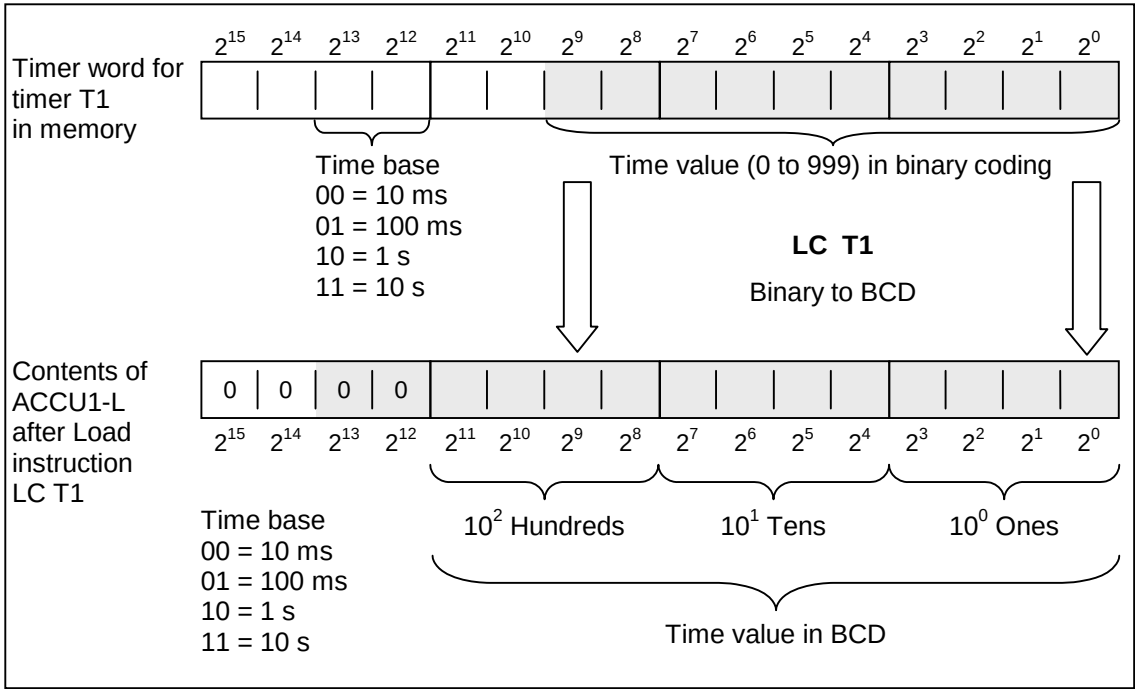
ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

12.5 LC 現在のタイマ値をBCD としてACCU 1 にロード

例

STL		説明
LC	T1	//ACCU 1-L に、タイマ T1 のタイムベースと現在のタイマ値を //2 進化 10 進数 (BCD) フォーマットでロードする。



12.6 R リセットタイマ

フォーマット

R<timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲は CPU により異なる。

命令の説明

R <timer>は、RLO が 0 から 1 になると、現在のタイミングファンクションを停止し、指定されたタイマワードのタイマ値とタイムベースをクリアします。

ステータスワード

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
書き込みの内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.1	
R T1	//入力 I 2.1 の信号状態をチェックする。RLO が 0 から 1 になると、 //タイマ T1 をリセットする。

12.7 SP パルスタイマ

フォーマット

SP <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲はCPUにより異なる。

命令の説明

SP <timer> は、RLO が"0"から"1"になると、指定されたタイマを起動します。プログラムされた時間間隔は、RLO が 1 である間、経過します。設定された時間に達する前に RLO が"0"になると、タイマは停止します。このタイマ起動コマンドは、時間値とタイムベースを ACCU 1-L に BCD で保存します。

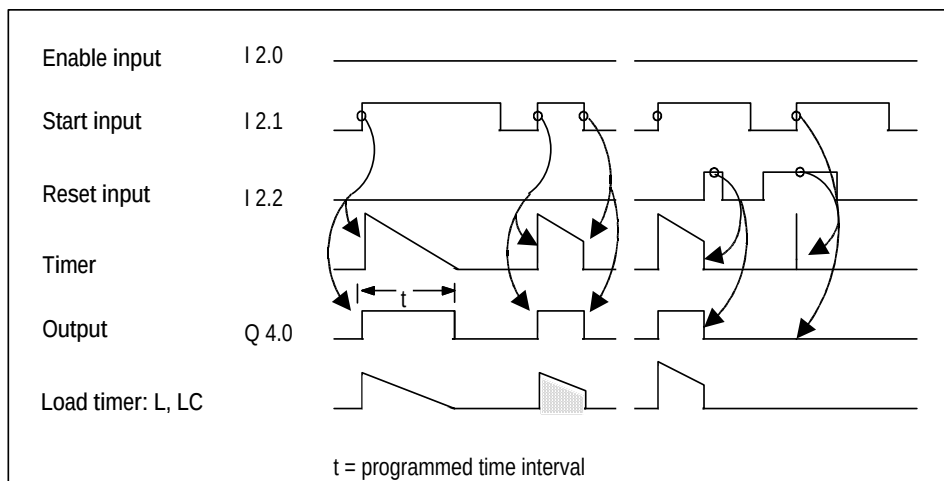
「メモリ内のタイマの位置とタイマのコンポーネント」も参照してください。

ステータスワード

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
書き込み の内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.0	
FR T1	//タイマ T1 を有効にする。
A I 2.1	
L S5T#10s	//ACCU 1 に 10 秒をプリセットする。
SP T1	//タイマ T1 をパルスタイマとして起動する。
A I 2.2	
R T1	//タイマ T1 をリセットする。
A T1	//タイマ T1 の信号状態をチェックする。
= Q 4.0	
L T1	//タイマ T1 の現在の時間値をバイナリ形式でロードする。
T MW10	
LC T1	//タイマ T1 の現在の時間値を BCD 形式でロードする。
T MW12	



12.8 SE 拡張パルスタイマ

フォーマット

SE <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲はCPUにより異なる。

命令の説明

SE <timer> は、RLO が"0"から"1"になると、指定されたタイマを起動します。RLO が"0"になっても、設定された時間間隔までカウントします。設定された時間に達する前にRLOが "0"から "1"になると、設定された時間間隔のカウントを開始します。このタイマ起動コマンドは、時間値とタイムベースを ACCU 1-L に BCD で保存します。

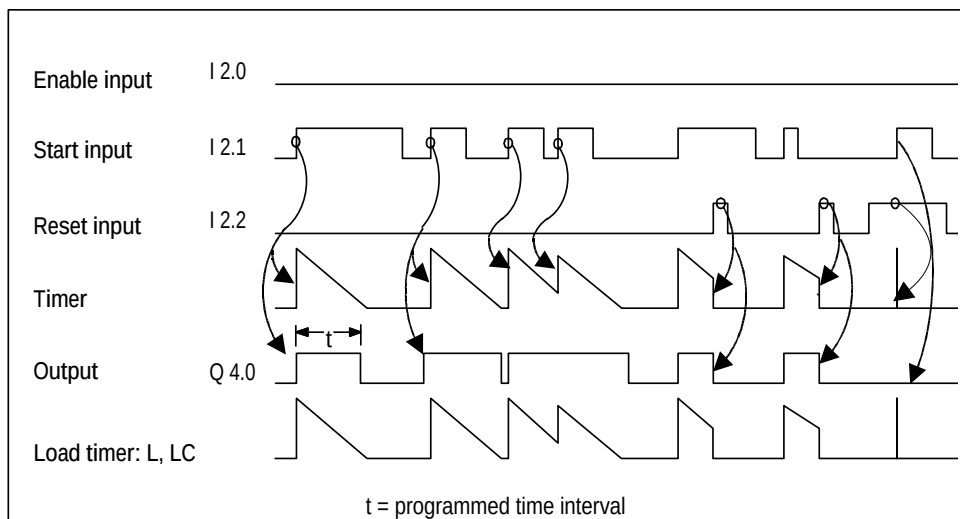
「メモリ内のタイマの位置とタイマのコンポーネント」も参照してください。

ステータスワード

	BIE	A1	A0	OV	OS	OR	STA	VKE	IER
書き込み の内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.0	
FR T1	//タイマ T1 を有効にする。
A I 2.1	
L S5T#10s	//ACCU 1 に 10 秒をプリセットする。
SE T1	//タイマ T1 を拡張パルスタイマとして起動する。
A I 2.2	
R T1	//タイマ T1 をリセットする。
A T1	//タイマ T1 の信号状態をチェックする。
= Q 4.0	
L T1	//タイマ T1 の現在のタイマ値をバイナリ形式でロードする。
T MW10	
LC T1	//タイマ T1 の現在のタイマ値を BCD 形式でロードする。
T MW12	



12.9 SD オンディレイタイマ

フォーマット

SD <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲はCPUにより異なる。

命令の説明

SD <timer>は、RLO が"0"から"1"になると、指定されたタイマを起動します。プログラムされた時間間隔は、RLO が 1 である間、経過します。設定された時間間隔に達する前に RLO が"0"になると、時間のカウントは停止します。このタイマ起動命令では、時間値とタイムベースを BCD で ACCU 1-L に保存します。

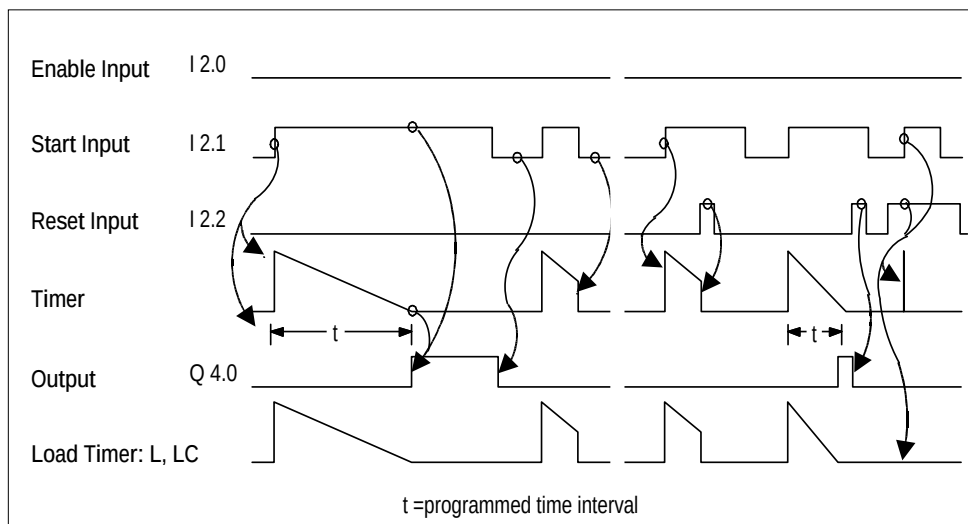
「メモリ内のタイマの位置とタイマのコンポーネント」も参照してください。

ステータスワード

	BIE	A1	A0	OV	OS	OR	STA	VKE	IER
書き込み の内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.0	
FR T1	//タイマ T1 を有効にする。
A I 2.1	
L S5T#10s	//ACCU 1 に 10 秒をプリセットする。
SD T1	//タイマ T をオンディレイタイマとして起動する。
A I 2.2	
R T1	//タイマ T1 をリセットする。
A T1	//タイマ T1 の信号状態をチェックする。
= Q 4.0	
L T1	//タイマ T1 の現在のタイマ値をバイナリ形式でロードする。
T MW10	
LC T1	//タイマ T1 の現在のタイマ値を BCD 形式でロードする。
T MW12	



12.10 SS 保持型オンディレイタイマ

フォーマット

SS <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲は CPU により異なる。

命令の説明

SS <timer>（タイマを拡張オンタイマとして起動）は、RLO が"1"から "0"になると、指定されたタイマを起動します。RLO が"0"になったとしても、設定された時間間隔までカウントします。設定した時間に達する前に RLO が"0"から"1"になると、設定した時間間隔が再トリガ（再起動）します。このタイマ起動コマンドは、時間値とタイムベースを ACCU 1-L に BCD で保存します。

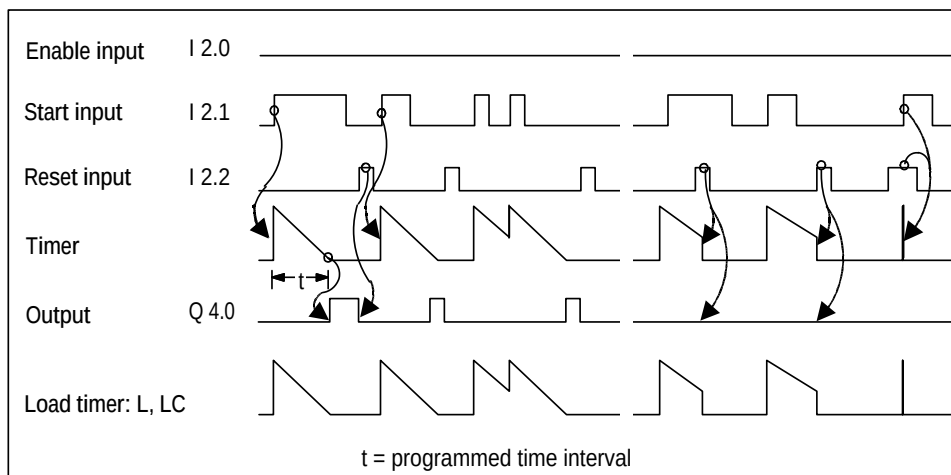
「メモリ内のタイマの位置とタイマのコンポーネント」も参照してください。

ステータスワード

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
書き込みの内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.0	
FR T1	//タイマ T1 を有効にする。
A I 2.1	
L S5T#10s	//ACCU 1 に 10 秒をプリセットする。
SS T1	//タイマ T1 を拡張オンディレイとして起動する。
A I 2.2	
R T1	//タイマ T1 をリセットする。
A T1	//タイマ T1 の信号状態をチェックする。
= Q 4.0	
L T1	//タイマ T1 の現在の時間値をバイナリ形式でロードする。
T MW10	
LC T1	//タイマ T1 の現在の時間値を BCD 形式でロードする。
T MW12	



12.11 SF オフディレイタイマ

フォーマット

SF <timer>

アドレス	データタイプ	メモリ領域	説明
<timer>	TIMER	T	タイマ番号。番号の範囲はCPUにより異なる。

命令の説明

SF <timer> は、RLO が"1"から "0"になると指定されたタイマを起動します。プログラムされた時間間隔は、RLO が 0 である間、経過します。設定された時間に達する前に RLO が"1"になると、時間は停止します。このタイマ起動コマンドは、時間値とタイムベースを ACCU 1-L に BCD で保存します。

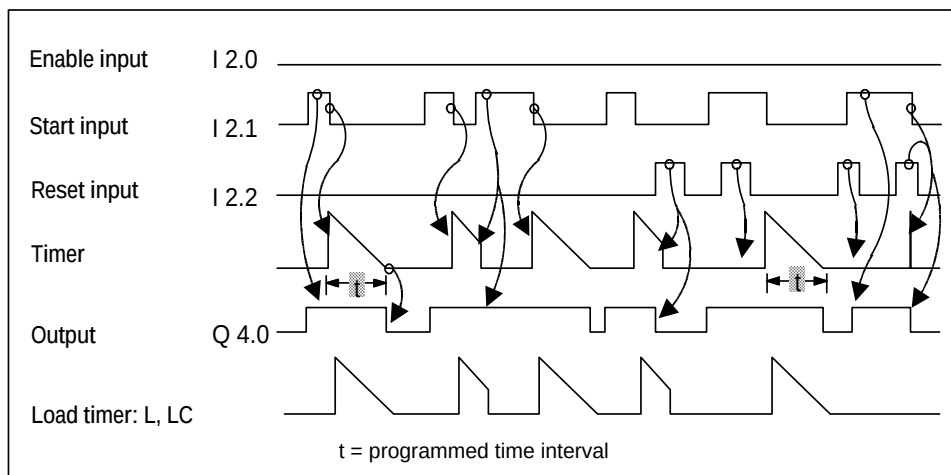
「メモリ内のタイマの位置とタイマのコンポーネント」も参照してください。

ステータスワード

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
書き込み の内容:	-	-	-	-	-	0	-	-	0

例

STL	説明
A I 2.0	
FR T1	//タイマ T1 を有効にする。
A I 2.1	
L S5T#10s	//ACCU 1 に 10 秒をプリセットする。
SF T1	//タイマ T1 をオフディレイタイマとして起動する。
A I 2.2	
R T1	//タイマ T1 をリセットする。
A T1	//タイマ T1 の信号状態をチェックする。
= Q 4.0	
L T1	//タイマ T1 の現在のタイマ値をバイナリ形式でロードする。
T MW10	
LC T1	//タイマ T1 の現在のタイマ値を BCD 形式でロードする。
T MW12	



13 ワード論理命令

13.1 ワード論理命令の概要

説明

ワード論理演算命令により、ワード（16 ビット）やダブルワード（32 ビット）のペアをブールロジックに従ってビットごとに比較します。ワードやダブルワードはそれぞれ、アキュムレータに入っていない必要があります。

ワードについて、アキュムレータ 2 の下位ワードの内容を、アキュムレータ 1 の下位ワードの内容と結合します。結合の結果は、アキュムレータ 1 の下位ワードに、前の内容を上書きして保存されます。

ダブルワードについて、アキュムレータ 2 の内容を、アキュムレータ 1 の内容と結合します。結合の結果は、アキュムレータ 1 に、前の内容を上書きして保存されます。

結果が 0 でない場合、ステータスワードのビット CC 1 は"1"に設定されます。結果が 0 の場合、ステータスワードのビット CC 1 は"0"に設定されます。

ワード論理演算には、以下の命令を使用できます。

- AW AND ワード（16 ビット）
- OW OR ワード（16 ビット）
- XOW 排他的 OR ワード（16 ビット）
- AD AND ダブルワード（32 ビット）
- OD OR ダブルワード（32 ビット）
- XOD 排他的 OR ダブルワード（32 ビット）

13.2 AW AND ワード (16 ビット)

フォーマット

AW
AW <constant>

アドレス	データタイプ	説明
<constant>	WORD、 16 ビット定数	AND により、ACCU 1-L と結合されるビットパターン

命令の説明

AW (AND ワード) は、ブール論理演算 AND に従って、ACCU 1-L の内容を ACCU 2-L または 16 ビット定数とビットごとに結合します。リザルトダブルワードのビットは、論理演算に組み込まれている両方のダブルワードの対応するビットが"1"の場合に"1"になります。この結果は、ACCU 1-L に保存されます。ACCU 1-H および ACCU 2 (ACCU が 4 つの CPU の場合は ACCU 3 と ACCU 4 も) は変更されません。ステータスビット CC 1 は、演算結果として設定されます (結果が 0 以外の場合、CC 1 は 1 になる)。ステータスワードビット CC 0 と OV は 0 にリセットされます。

AW: ACCU 1-L を ACCU 2-L と結合します。

AW <constant>: ACCU 1-L を 16 ビット定数と結合します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	0	0	-	-	-	-	-

例

ビット	15 ...	..	..	... 0
AW 命令実行前の ACCU 1-L	0101	1001	0011	1011
ACCU 2-L または 16 ビット定数	1111	0110	1011	0101
AW 命令実行後の結果 (ACCU 1-L)	0101	0000	0011	0001

例 1

STL	説明
L IW20	//IW20 の内容を ACCU 1-L にロードする。
L IW22	//ACCU 1 の値を ACCU 2 にロードする。//IW22 の内容を ACCU 1-L にロードする。
AW	//AND により、ACCU 1-L のビットを ACCU 2-L のビットと結合する。結果を ACCU 1-L に保存する。
T MW 8	//結果を MW8 へ転送する。

例 2

STL	説明
L IW20	//IW20 の内容を ACCU 1-L にロードする。
AW W#16#0FFF	//AND により、ACCU 1-L のビットを 16 ビット定数のビットパターン // (0000_1111_1111_1111) と結合する。結果を ACCU 1-L に保存する。
JP NEXT	//結果が 0 以外の場合 (CC 1 = 1)、NEXT ジャンプラベルへジャンプする。

13.3 OW OR ワード (16 ビット)

フォーマット

OW
OW <constant>

アドレス	データタイプ	説明
<constant>	WORD、 16 ビット定数	OR により、ACCU 1-L と結合されるビットパターン

命令の説明

OW (OR ワード) は、ブール論理演算 OR に従って、ACCU 1-L の内容を ACCU 2-L または 16 ビット定数とビットごとに結合します。論理演算に組み込まれている両方のダブルワードの対応するビットの少なくとも 1 つが"1"の場合、リザルトワードのビットは"1"になります。この結果は、ACCU 1-L に保存されます。ACCU 1-H および ACCU 2 (ACCU が 4 つの CPU の場合は ACCU 3 と ACCU 4 も) は変更されません。この命令は、RLO に関係なく実行され、これにより RLO が変更されることもありません。ステータスビット CC 1 は、演算結果として設定されます (結果が 0 以外の場合、CC 1 は 1 になる)。ステータスワードビット CC 0 と OV は 0 にリセットされます。

OW: ACCU 1-L を ACCU 2-L と結合します。

OW <constant>: ACCU 1-L を 16 ビット定数と結合します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	x	0	0	-	-	-	-	-

例

ビット	15 ...	..	..	... 0
OW 命令実行前の ACCU 1-L	0101	0101	0011	1011
ACCU 2-L または 16 ビット定数	1111	0110	1011	0101
OW 命令実行後の結果 (ACCU 1-L)	1111	0111	1011	1111

例 1

STL	説明
L IW20	//IW20 の内容を ACCU 1-L にロードする。
L IW22	//ACCU 1 の値を ACCU 2 にロードする。IW22 の内容を ACCU 1-L にロードする。
OW	//OR により、ACCU 1-L のビットを ACCU 2-L のビットと結合する。結果を ACCU 1-L に保存する。
T MW8	//結果を MW8 へ転送する。

例 2

STL	説明
L IW20	//IW 20 の内容を ACCU 1-L にロードする。
OW W#16#0FFF	//OR により、ACCU 1-L のビットを 16 ビット定数のビットパターン // (0000 1111 1111 1111) と結合する。結果を ACCU 1-L に保存する。
JP NEXT	//結果が 0 以外の場合 (CC 1 = 1)、NEXT ジャンプラベルへジャンプする。

13.4 XOW 排他的 OR ワード (16 ビット)

フォーマット

XOW
XOW <constant>

アドレス	データタイプ	説明
<constant>	WORD、 16 ビット定数	XOR (排他的 OR) により、ACCU 1-L と結合される ビットパターン

命令の説明

XOW (XOR ワード) は、ブール論理演算 XOR に従って、ACCU 1-L の内容を ACCU 2-L または 16 ビット定数とビットごとに結合します。論理演算に組み込まれている両方のダブルワードの対応するビットの少なくとも 1 つが "1" の場合、リザルトワードのビットは "1" になります。この結果は、ACCU 1-L に保存されます。ACCU 1-H および ACCU 2 は変更されません。ステータスビット CC 1 は、演算結果として設定されます (結果が 0 以外の場合、CC 1 は 1 になる)。ステータスワードビット CC 0 と OV は 0 にリセットされます。

排他的 OR ファンクションを複数回使用できます。チェックされたアドレスの不良数が "1" の場合、論理演算の結果が "1" になります。

XOW: ACCU 1-L を ACCU 2-L と結合します。

XOW <constant>: ACCU 1-L を 16 ビット定数と結合します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	x	0	0	-	-	-	-	-

例

ビット	15 ...	..	..	... 0
XOW 命令実行前の ACCU 1	0101	0101	0011	1011
ACCU 2-L または 16 ビット定数	1111	0110	1011	0101
XOW 命令実行後の結果 (ACCU 1)	1010	0011	1000	1110

例 1

STL	説明
L IW20	//IW20 の内容を ACCU 1-L にロードする。
L IW22	//ACCU 1 の値を ACCU 2 にロードする。ID24 の内容を ACCU 1-L にロードする。
XOW	//XOR により、ACCU 1-L のビットを ACCU 2-L のビットと結合する。結果を ACCU 1-L に保存する。
T MW8	//結果を MW8 へ転送する。

例 2

STL	説明
L IW20	//IW20 の内容を ACCU 1-L にロードする。
XOW 16#0FFF	//XOR により、ACCU 1-L のビットを 16 ビット定数のビットパターン (0000_1111_1111_1111) と結合する。結果を ACCU 1-L に保存する。
JP NEXT	//結果が 0 以外の場合 (CC 1=1)、NEXT ジャンプラベルへジャンプする。

13.5 AD AND ダブルワード (32 ビット)

フォーマット

AD
AD <constant>

アドレス	データタイプ	説明
<constant>	DWORD、 32 ビットの定数	AND により ACCU 1 と結合されるビットパターン

命令の説明

AD (AND ダブルワード) は、ブール論理演算 AND に従って、ACCU 1 の内容を ACCU 2 または 32 ビット定数と 1 ビットずつ結合します。リザルトダブルワードのビットは、論理演算に組み込まれている両方のダブルワードの対応するビットが"1"の場合に"1"になります。この結果は、ACCU 1-L に保存されます。ACCU 2 (ACCU が 4 つの CPU の場合は ACCU 3 と ACCU 4 も) は変更されません。ステータスビット CC 1 は、演算結果として設定されます (結果が 0 以外の場合、CC 1 は 1 になる)。ステータスワードビット CC 0 と OV は 0 にリセットされます。

AD: ACCU 1 を ACCU 2 と結合します。

AD <constant>: ACCU 1-L を 32 ビット定数と結合します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	x	0	0	-	-	-	-	-

例

ビット	31 ..	..	..	..	..	..	..	... 0
UD 命令実行前の ACCU 1	0101	0000	1111	1100	1000	1001	0011	1011
ACCU 2 または 32 ビット定数	1111	0011	1000	0101	0111	0110	1011	0101
UD 命令実行後の結果 (ACCU 1)	0101	0000	1000	0100	0000	0000	0011	0001

例 1

STL	説明
L ID20	//ID20 の内容を ACCU 1 にロードする。
L ID24	//ACCU 1 の値を ACCU 2 にロードする。ID24 の内容を ACCU 1 にロードする。
AD	//AND により、ACCU 1 のビットと ACCU 2 のビットを結合する。結果を ACCU 1 に保存する。
T MD8	//結果を MD8 に転送する。

例 2

STL	説明
L ID 20	//ID20 の内容を ACCU 1 にロードする。
AD DW#16#0FFF_EF21	//AND により、ACCU 1 のビットを 32 ビット定数のビットパターン // (0000_1111_1111_1111_1110_1111_0010_0001) と結合する。 //その結果を ACCU 1 に保存する。
JP NEXT	//結果が 0 以外の場合 (CC 1 = 1)、NEXT ジャンプラベルへジャンプする。

13.6 OD OR ダブルワード (32 ビット)

フォーマット

OD
OD <constant>

アドレス	データタイプ	説明
<constant>	DWORD、 32 ビットの定数	OR により、ACCU 1 と結合されるビットパターン

命令の説明

OD (OR ダブルワード) は、ブール論理演算 OR に従って、ACCU 1 の内容を ACCU 2 または 32 ビット定数と 1 ビットずつ結合します。論理演算に組み込まれている両方のダブルワードの対応するビットの少なくとも 1 つが"1"の場合、リザルトダブルワードのビットは"1"になります。この結果は、ACCU 1-L に保存されます。ACCU 2 (ACCU が 4 つの CPU の場合は ACCU 3 と ACCU 4 も) は変更されません。ステータスビット CC 1 は、演算結果のファンクションとして設定されます (結果が 0 以外の場合、CC 1 は 1 になる)。ステータスワードビット CC 0 と OV は 0 にリセットされます。

OD: ACCU 1 を ACCU 2 と結合します。

OD <constant>: ACCU 1-L を 32 ビット定数と結合します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	0	0	-	-	-	-	-

例

ビット	31 ..	..	..	..	..	..	..	... 0
OD 命令実行前の ACCU 1	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 または 32 ビット定数	1111	0011	1000	0101	0111	0110	1011	0101
OD 命令実行後の結果 (ACCU 1)	1111	0011	1111	1101	1111	0111	1011	1111

例 1

STL	説明
L ID20	//ID20 の内容を ACCU 1 にロードする。
L ID24	//ACCU 1 の値を ACCU 2 にロードする。ID24 の内容を ACCU 1 にロードする。
OD	//OR により、ACCU 1 のビットを ACCU 2 ビットと結合する。結果を ACCU 1 に保存する。
T MD8	//結果を MD8 に転送する。

例 2

STL	説明
L ID20	//ID20 の内容を ACCU 1 にロードする。
OD DW#16#0FFF_EF21	//OR により、ACCU 1 のビットを 32 ビット定数のビットパターン // (0000_1111_1111_1111_1110_1111_0010_0001) と結合する。 //結果を ACCU 1 に保存する。
JP NEXT	//結果が 0 以外の場合 (CC 1 = 1)、NEXT ジャンプラベルへジャンプする。

13.7 XOD 排他的 OR ダブルワード (32 ビット)

フォーマット

XOD
XOD <constant>

アドレス	データタイプ	説明
<constant>	DWORD、 32 ビットの定数	XOR (排他的 OR) により、ACCU 1 と結合されるビットパターン

命令の説明

XOD (XOR ダブルワード) は、ブール論理演算 XOR (排他的 OR) に従って、ACCU 1 の内容を ACCU 2 または 32 ビット定数と 1 ビットずつ結合します。論理演算に組み込まれている両方のダブルワードの対応するビットの少なくとも 1 つが"1"の場合、リザルトダブルワードのビットは"1"になります。この結果は、ACCU 1-L に保存されます。ACCU 2 は変更されません。ステータスビット CC 1 は、演算結果として設定されます (結果が 0 以外の場合、CC 1 は 1 になる)。ステータスワードビット CC 0 と OV は 0 にリセットされます。

排他的 OR ファンクションを複数回使用できます。チェックされたアドレスの不良数が"1"の場合、論理演算の結果が"1"になります。

XOD: ACCU 1 を ACCU 2 と結合します。

XOD <constant>: ACCU 1-L を 32 ビット定数と結合します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	x	0	0	-	-	-	-	-

例

ビット	31 ..	..	..	..	..	..	..	... 0
XOD 命令実行前の ACCU 1	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 または 32 ビット定数	1111	0011	1000	0101	0111	0110	1011	0101
XOD 命令実行後の結果 (ACCU 1)	1010	0011	0111	1001	1111	0011	1000	1110

例 1

STL	説明
L ID20	//ID20 の内容を ACCU 1 にロードする。
L ID24	//ACCU 1 の値を ACCU 2 にロードする。ID24 の内容を ACCU 1 にロードする。
XOD	//XOR により、ACCU 1 のビットを ACCU 2 のビットと結合する。結果を ACCU 1 に保存する。
T MD8	//結果を MD8 に転送する。

例 2

STL	説明
L ID20	//ID20 の内容を ACCU 1 にロードする。
XO DW#16#0FFF_EF21	//XOR により、ACCU 1 のビットを 32 ビット定数のビットパターン
D	// (0000_1111_1111_1111_1111_1110_0010_0001) と結合する。 //結果を ACCU 1 に保存する。
JP NEXT	//結果が 0 以外の場合 (CC 1 = 1)、NEXT ジャンプラベルへジャンプする。

13.7 XOD 排他的OR ダブルワード (32 ビット)

14 アキュムレータ命令

14.1 アキュムレータの演算とアドレスレジスタ命令の概要

説明

以下の命令は、アキュムレータの内容を処理するために使用することができます。

- TAK ACCU 1 と ACCU 2 の切り替え
- PUSH ACCU が 2 つの CPU
- PUSH ACCU が 4 つの CPU
- POP CPU が 2 つの CPU
- POP ACCU が 4 つの CPU

- ENT ACCU スタックに入る
- LEAVE ACCU スタックから出る
- INC ACCU 1-L-L に加算
- DEC ACCU 1-L-L から減算

- +AR1 ACCU 1 をアドレスレジスタ 1 に追加
- +AR2 ACCU 1 をアドレスレジスタ 2 に追加

- BLD プログラム表示命令 (Null)
- NOP 0 Null 命令
- NOP 1 Null 命令

14.2 TAK ACCU 1 と ACCU 2 の切り替え

フォーマット

TAK

説明

TAK (ACCU 1 を ACCU 2 と入れ替え) は、ACCU 1 の内容を ACCU 2 の内容と入れ替えます。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。ACCU が 4 つの CPU の場合、ACCU 3 および ACCU 4 の内容は変更されません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例: 大きい値から小さい値を減算

STL	説明		
L MW10	//MW10 の内容を ACCU 1-L にロードする。		
L MW12	//ACCU 1-L の内容を ACCU 2-L にロードする。MW12 の内容を ACCU 1-L にロードする。		
>I	//ACCU 2-L (MD10) が ACCU 1-L (ID14) より大きいかどうかチェックする。		
SPB NEXT	//ACCU 2 (MW10) が ACCU 1 (MW12) よりも大きい場合、NEXT ジャンプラベルへジャンプする。		
TAK	//ACCU 1 の内容と ACCU 2 の内容を交換する。		
NEXT: -I	//ACCU 1-L の内容から ACCU 2-L の内容を減算する。		
T MW14	//結果 (= 大きい値から小さい値を減算) を MW14 へ転送する。		

目次	ACCU 1	ACCU 2
TAK 命令実行前の値	<MW12>	<MW10>
TAK 命令実行後の値	<MW10>	<MW12>

14.3 POP ACCU が2 つの CPU

フォーマット

POP

説明

POP (ACCU が2 つの CPU) は、ACCU 2 の内容全体を ACCU 1 にコピーします。ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
T MD10	//ACCU 1 の内容 (= A) を MD10 に転送する。
POP	//ACCU 2 の内容全体を ACCU 1 にコピーする。
T MD14	//ACCU 1 の内容 (= B) を MD14 に転送する。

目次	ACCU 1	ACCU 2
POP 命令実行前の値	A	B
POP 命令実行後の値	B	B

14.4 POP ACCU が4 つの CPU

フォーマット

POP

説明

POP (ACCU が4 つの CPU) は、ACCU 2 の内容全体を ACCU 1 に、ACCU 3 の内容全体を ACCU 2 に、ACCU 4 の内容全体を ACCU 3 にコピーします。ACCU 4 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
T MD10	//ACCU 1 の内容 (= A) を MD10 に転送する。
POP	//ACCU 2 の内容全体を ACCU 1 にコピーする。
T MD14	//ACCU 1 の内容 (= B) を MD14 に転送する。

目次	ACCU 1	ACCU 2	ACCU 3	ACCU 4
POP 命令実行前の値	A	B	C	D
POP 命令実行後の値	B	C	D	D

14.5 PUSH ACCU が2 つの CPU

フォーマット

PUSH

説明

PUSH (ACCU 1 から ACCU 2) は、ACCU 1 の内容全体を ACCU 2 にコピーします。ACCU 1 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MW10	//MW10 の内容を ACCU 1 にロードする。
PUSH	//ACCU 1 の内容全体を ACCU 2 にコピーする。

目次	ACCU 1	ACCU 2
PUSH 命令実行前の値	<MW10>	<X>
PUSH 命令実行後の値	<MW10>	<MW10>

14.6 PUSH ACCU が4 つの CPU

フォーマット

PUSH

説明

PUSH (ACCU が4 つの CPU) は、ACCU 3 の内容全体を ACCU 4 に、ACCU 2 の内容全体を ACCU 3 に、ACCU 1 の内容全体を ACCU 2 にコピーします。ACCU 1 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MW10	//MW10 の内容を ACCU 1 にロードする。
PUSH	//ACCU 1 の内容全体を ACCU 2 に、ACCU 2 の内容を ACCU 3 に、 //ACCU 3 の内容を ACCU 4 にコピーする。

目次	ACCU 1	ACCU 2	ACCU 3	ACCU 4
PUSH 命令実行前の値	A	B	C	D
PUSH 命令実行後の値	A	A	B	C

14.7 ENT ACCU スタックに入る

フォーマット

ENT

説明

ENT（アキュムレータスタック入力）は、ACCU 3 の内容を ACCU 4 に、ACCU 2 の内容を ACCU 3 にコピーします。ロード命令の入力側に直接 ENT 命令をプログラムすると、ACCU 3 に中間結果を保存できます。

例

STL	説明
L DBD0	//データダブルワード DBD0 の値を ACCU 1 にロードする。 //（この値は浮動小数点数形式にする必要がある）。
L DBD4	//ACCU 1 の値を ACCU 2 にコピーする。 //データダブルワード DBD4 の値を ACCU 1 にロードする。（この値は浮動小数点数形式にする必要がある）。
+R	//ACCU 1 と ACCU 2 の各内容を浮動小数点数（32 ビット、IEEE 754） //として加算して、その結果を ACCU 1 に保存する。
L DBD8	//ACCU 1 の値を ACCU 2 にコピーし、データダブルワード DBD8 の値を // ACCU 1 にロードする。
ENT	//ACCU 3 の内容を ACCU 4 にコピーする。ACCU 2 の内容（中間結果）を ACCU 3 に //コピーする。
L DBD12	//データダブルワード DBD12 の値を ACCU 1 にロードする。
-R	//ACCU 2 から ACCU 1 の内容を減算し、その結果を //ACCU 1 に保存する。ACCU 3 の内容を ACCU 2 にコピーする。 //ACCU 4 の内容を ACCU 3 にコピーする。
/R	//ACCU 2 の内容（DBD0+DBD4）を ACCU 1 の内容（DBD8-DBD12）で除算する。その結果を ACCU 1 //に保存する。
T DBD16	//結果（ACCU 1）をデータダブルワード DBD16 に転送する。

14.8 LEAVE ACCU スタックから出る

フォーマット

LEAVE

説明

LEAVE（アキュムレータスタックから出す）は、ACCU 3 の内容を ACCU 2 に、ACCU 4 の内容を ACCU 3 にコピーします。移動または回転命令の入力側に直接 LEAVE 命令をプログラムし、アキュムレータを結合すると、LEAVE 命令は算術命令のように機能します。ACCU 1 および ACCU 4 の内容は変更されません。

14.9 INC ACCU 1-L-L に加算

フォーマット

INC <8-bit integer>

パラメータ	データタイプ	説明
<8-bit integer>	8 ビット整数	ACCU 1-L-L に加算される定数（値の範囲は 0～255）。

説明

INC <8-bit integer>（ACCU 1-L-L に加算）は、ACCU 1-L-L の内容に 8 ビット整数を加算し、その結果を ACCU 1-L-L に保存します。ACCU 1-L-H、ACCU 1-H、および ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

注

これらの命令は 16 ビットまたは 32 ビット演算には対応していません。アキュムレータ 1 の下位ワードの下位バイトから、アキュムレータ 1 の下位ワードの上位バイトへの繰り上がりが行われないからです。16 ビットまたは 32 ビット演算には、+I または +D 命令をそれぞれ使用します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MB22	//MB22 の値をロードする。
INC 1	// "ACCU 1-L-L に 1 を加算" 命令を実行し、その結果を ACCU 1-L-L に保存する。
T MB22	// ACCU 1-L-L の内容（結果）を MB22 へ転送する。

14.10 DEC ACCU 1-L-L からの減算

フォーマット

DEC <8-bit integer>

アドレス	データタイプ	説明
<8-bit integer>	8 ビット整数定数	ACCU 1-L-L から引かれる定数（値の範囲は 0～255）。

説明

DEC <8-bit integer>（ACCU 1-L-L から減算）は、ACCU 1-L-L の内容から 8 ビット整数を引き、その結果を ACCU 1-L-L に保存します。ACCU 1-L-H、ACCU 1-H、および ACCU 2 は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることもありません。

注

これらの命令は 16 ビットまたは 32 ビット演算には対応していません。アキュムレータ 1 の下位ワードの下位バイトから、アキュムレータ 1 の下位ワードの上位バイトへの繰り上がりが行われないからです。16 ビットまたは 32 ビット演算には、+I または +D 命令をそれぞれ使用します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例

STL	説明
L MB250	//MB250 の値をロードする。
DEC 1	// "ACCU 1-L-L から 1 を減算" 命令を実行し、その結果を ACCU 1-L-L に保存する。
T MB250	// ACCU 1-L-L の内容（結果）を MB250 へ転送する。

14.11 +AR1 ACCU 1 をアドレスレジスタ 1 に追加

フォーマット

+AR1
+AR1 <P#Byte.Bit>

パラメータ	データタイプ	説明
<P#Byte.Bit>	ポインタ定数	AR1 に追加されるアドレス

説明

+AR1（AR1 に加算）は、命令または ACCU 1-L で指定したオフセットを AR の内容に加算します。整数（16 ビット）は、まず、適切な符号付きの 24 ビット整数に拡張された後、AR1 の最下位 24 ビット（AR1 の相対アドレス部）に加算されます。AR1 の ID 領域（ビット 24、25、および 26）は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

+AR1: AR1 の内容に加算される整数（16 ビット）は、ACCU 1-L の値で指定されます。有効な値の範囲は-32768～+32767 です。

+AR1 <P#Byte.Bit>: 加算するオフセットは、<P#Byte.Bit>アドレスで指定されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例 1

STL	説明
L +300	//値を ACCU 1-L にロードする。
+AR1	//ACCU 1-L（整数、16 ビット）を AR1 に加算する。

例 2

STL	説明
+AR1 P#300.0	//オフセット 300.0 を AR1 に加算する。

14.12 +AR2 ACCU 1 をアドレスレジスタ 2 に追加

フォーマット

+AR2
+AR2 <P#Byte.Bit>

パラメータ	データタイプ	説明
<P#Byte.Bitr>	ポインタ定数	AR2 に追加されるアドレス

説明

+AR2 (AR2 に追加) は、命令または ACCU 1-L で指定されたオフセットを AR の内容に加算します。整数 (16 ビット) はまず、正しい符号付きの 2 ビット整数になり、AR2 の最下位 24 ビット (AR2 の相対アドレス部) に追加されます。AR2 の ID 領域 (ビット 24、25、および 26) は変更されません。この命令は、ステータスビットに関係なく実行され、これにより、ステータスビットが変更されることはありません。

+AR2: AR2 の内容に追加される整数 (16 ビット) は、ACCU 1-L の値で指定します。有効な値の範囲は -32768 ~ +32767 です。

+AR2 <P#Byte.Bit>: 加算するオフセットは <P#Byte.Bit> アドレスで指定されます。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込みの内容:	-	-	-	-	-	-	-	-	-

例 1

STL	説明
L +300	//値を ACCU 1-L にロードする。
+AR1	//ACCU 1-L (整数、16 ビット) を AR2 に加算する。

例 2

STL	説明
+AR1 P#300.0	//オフセット 30.0 を AR2 に加算する。

14.13 BLD プログラム表示命令 (Null)

フォーマット

BLD <number>

アドレス	説明
<number>	この数値は BLD 命令を指定する（範囲は 0～255）。

説明

BLD <number>（プログラム表示命令、NULL 命令）は、ファンクションを実行しません。ステータスビットも変更しません。この命令は、グラフィック表示用プログラミング装置（PG）に使用されます。STL にラダーまたは FBD プログラムが表示されると、自動的に作成されます。BLD 命令はアドレス <number> で指定しますが、このアドレスはプログラミング装置で生成します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

14.14 NOP 0 Null 命令

フォーマット

NOP 0

説明

NOP 0（アドレス"0"の NOP 命令）は、ファンクションを実行しません。ステータスビットも変更されません。この命令コードには、ゼロが 16 個指定されたビットパターンが指定されています。この命令は、プログラミング装置（PG）でプログラムを表示する場合にのみ使用します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

14.15 NOP 1 Null 命令

フォーマット

NOP 1

説明

NOP 1（アドレス"1"のNOP 命令）は、ファンクションを実行しません。ステータスビットも変更されません。この命令コードには、1 が 16 個指定されたビットパターンが指定されています。この命令は、プログラミング装置（PG）でプログラムを表示する場合にのみ使用します。

ステータスワード

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
書き込み の内容:	-	-	-	-	-	-	-	-	-

A すべての STL 命令の概要

A.1 ドイツ語のプログラム表記法 (SIMATIC) に従ってソートされた STL 命令

ドイツ語の プログラム 表記法	英語の プログラム 表記法	プログラムエレメント カタログ	説明
+	+	整数値演算命令	整数の追加 (16、32 ビット)
=	=	ビットロジック命令	割り付け
)	)	ビットロジック命令	ネストを閉じる
+AR1	+AR1	アキュムレータ	AR1 ACCU 1 をアドレスレジスタ 1 へ加算
+AR2	+AR2	アキュムレータ	AR2 ACCU 1 をアドレスレジスタ 2 へ加算
+D	+D	整数値演算命令	ACCU 1 と ACCU 2 を加算 (32 ビットダブル整数)
-D	-D	整数値演算命令	ACCU 2 から ACCU 1 を倍長整数として減算 (32 ビット)
*D	*D	整数値演算命令	ACCU 1 と ACCU 2 の倍長整数 (32 ビット) での乗算
/D	/D	整数値演算命令	倍長整数 (32 ビット) として、ACCU 2 を ACCU 1 で割る
? D	? D	比較	倍長整数 (32-Bit) の比較 ==, <>, >, <, >=, <=
+I	+I	整数値演算命令	ACCU 1 と ACCU 2 の整数 (16 ビット) での加算
-I	-I	整数値演算命令	ACCU 2 から ACCU 1 を整数として減算 (16 ビット)
*I	*I	整数値演算命令	ACCU 1 と ACCU 2 を整数として乗算 (16 ビット)
/I	/I	整数値演算命令	整数 (16 ビット) として、ACCU 2 を ACCU 1 で割る
? I	? I	比較	整数 (16 ビット) の比較 ==, <>, >, <, >=, <=
+R	+R	浮動小数点命令	ACCU 1 と ACCU 2 の加算 (32 ビットの IEEE 754 浮動小数点数)
-R	-R	浮動小数点命令	ACCU 2 から ACCU 1 を減算 (32 ビットの IEEE 754 浮動小数点数)
*R	*R	浮動小数点命令	ACCU 1 と ACCU 2 の乗算 (32 ビットの IEEE 754 浮動小数点数)
/R	/R	浮動小数点命令	ACCU 2 を ACCU 1 で除算 (32 ビットの IEEE 754 浮動小数点数)
? R	? R	比較	浮動小数点数 (32 ビット) の比較 ==, <>, >, <, >=, <=
ABS	ABS	浮動小数点命令	浮動小数点数の絶対値 (32 ビットの IEEE 754)
ACOS	ACOS	浮動小数点命令	浮動小数点数 (32 ビット) のアークコサインを生成
ASIN	ASIN	浮動小数点命令	浮動小数点数 (32 ビット) のアークサインを生成
ATAN	ATAN	浮動小数点命令	浮動小数点数 (32 ビット) のアークタンジェントを生成
AUF	OPN	DB 呼び出し	データブロックを開く
BE	BE	プログラムコントロール	ブロックの終了
BEA	BEU	プログラムコントロール	ブロックの条件なし終了
BEB	BEC	プログラムコントロール	ブロックの条件付き終了
BLD	BLD	プログラムコントロール	プログラム表示命令 (Null)
BTD	BTD	変換	整数への BCD (32 ビット)
BTI	BTI	変換	整数への BCD (16 ビット)
CALL	CALL	プログラムコントロール	[ブロック呼び出し]
CALL	CALL	プログラムコントロール	複数インスタンスの呼び出し
CALL	CALL	プログラムコントロール	ライブラリからのブロックの呼び出し
CC	CC	プログラムコントロール	条件付き呼び出し

A.1 ドイツ語のプログラム表記法 (SIMATIC) に従ってソートされた STL 命令

ドイツ語の プログラム 表記法	英語の プログラム 表記法	プログラムエレメント カタログ	説明
CLR	CLR	ビットロジック命令	RLO のクリア (= 0)
COS	COS	浮動小数点命令	角度のコサインを生成 (32 ビット浮動小数点数)
DEC	DEC	アキュムレータ	ACCU 1-L-L のディクリメント
DTB	DTB	変換	BCD へのダブル整数 (32 ビット)
DTR	DTR	変換	倍長整数 (32 ビット) から浮動小数点数への変換 (32 ビット、IEEE 754)
ENT	ENT	アキュムレータ	ACCU スタックに入る
EXP	EXP	浮動小数点命令	浮動小数点数の指数値を生成 (32 ビット)
FN	FN	ビットロジック命令	立ち下がリパルス
FP	FP	ビットロジック命令	立ち上がりパルス
FR	FR	カウンタ	イネーブルカウンタ (フリー) (フリー、FR C 0 ~ o C 255)
FR	FR	タイマ	イネーブルタイマ (フリー)
INC	INC	アキュムレータ	ACCU 1-L-L のインクリメント
INVD	INVD	変換	1 の補数ダブル整数 (32 ビット)
INVI	INVI	変換	1 の補数整数 (16 ビット)
ITB	ITB	変換	整数 (16 ビット) から BCD への変換
ITD	ITD	変換	ダブル整数 (32 ビット) への整数 (16 ビット)
L	L	ロード/転送	ロード
L DBLG	L DBLG	ロード/転送	共有 DB の長さを ACCU 1 にロード
L DBNO	L DBNO	ロード/転送	共有 DB の数を ACCU 1 にロード
L DILG	L DILG	ロード/転送	インスタンス DB の長さを ACCU 1 にロード
L DINO	L DINO	ロード/転送	インスタンス DB の数を ACCU 1 にロード
L STW	L STW	ロード/転送	ステータスワードを ACCU 1 にロード
L	L	ロード/転送	カレントタイマ値を整数として ACCU 1 にロード (カレントタイマ値は 0 から 255 まで有効、たとえば L T 32)
L	L	ロード/転送	カレントカウンタ値を ACCU 1 にロード (カレントカウンタ値は 0 から 255 まで有効、たとえば L C 15)
LAR1	LAR1	ロード/転送	ACCU 1 からアドレスレジスタ 1 をロード
LAR1	LAR1	ロード/転送	アドレスレジスタ 1 をダブル整数 (32 ビットポインタ) でロード
LAR1	LAR1	ロード/転送	アドレスレジスタ 2 からアドレスレジスタ 1 をロード
LAR2	LAR2	ロード/転送	ACCU 1 からアドレスレジスタ 2 をロード
LAR2	LAR2	ロード/転送	ダブル整数 (32 ビットポインタ) でアドレスレジスタ 2 をロード
LC	LC	カウンタ	カレントカウンタ値を BCD として ACCU 1 にロード (カレントタイマ値は 0 から 255 まで有効、たとえば LC C 15)
LC	LC	タイマ	カレントタイマ値を BCD として ACCU 1 にロード (カレントカウンタ値は 0 から 255 まで有効、たとえば LC T 32)
LEAVE	LEAVE	アキュムレータ	ACCU スタックから出る
LN	LN	浮動小数点命令	自然対数を浮動小数点数 (32 ビット) で生成
LOOP	LOOP	ジャンプ	ループ
MCR (	MCR (	プログラムコントロール	MCR スタックに RLO を保存、MCR の開始
) MCR	) MCR	プログラムコントロール	MCR の終了

A.1 ドイツ語のプログラム表記法 (SIMATIC) に従ってソートされた STL 命令

ドイツ語の プログラム 表記法	英語の プログラム 表記法	プログラムエレメント カタログ	説明
MCRA	MCRA	プログラムコントロール	MCR 領域の有効化
MCRD	MCRD	プログラムコントロール	MCR 領域の無効化
MOD	MOD	整数値演算命令	除算による余り (32 ビットダブル整数)
NEGD	NEGD	変換	倍長整数 (32 ビット) の 2 の補数
NEGI	NEGI	変換	2 の補数整数 (16 ビット)
NEGR	NEGR	変換	浮動小数点数 (32 ビット、IEEE 754) の否定
NOP 0	NOP 0	アキュムレータ	Null 命令
NOP 1	NOP 1	アキュムレータ	Null 命令
NOT	NOT	ビットロジック命令	RLO の否定
O	O	ビットロジック命令	OR
O (	O (	ビットロジック命令	OR/ネストを開く
OD	OD	ワード論理命令	ダブルワード (32 ビット) の OR
ON	ON	ビットロジック命令	OR NOT
ON (	ON (	ビットロジック命令	OR NOT/ネストを開く
OW	OW	ワード論理命令	OR ワード (16 ビット)
POP	POP	アキュムレータ	ACCU が 2 つの CPU
POP	POP	アキュムレータ	ACCU が 4 つの CPU
PUSH	PUSH	アキュムレータ	ACCU が 2 つの CPU
PUSH	PUSH	アキュムレータ	ACCU が 4 つの CPU
R	R	ビットロジック命令	リセット
R	R	カウンタ	カウンタをリセット (カレントカウンタ値は 0 から 255 まで有効、たとえば R C 15)
R	R	タイマ	タイマをリセット (カレントタイマ値は 0 から 255 まで有効、たとえば R T 32)
RLD	RLD	シフト/ 循環	左回転ダブルワード (32 ビット)
RLDA	RLDA	シフト/ 循環	CC 1 を介して ACCU 1 を左回転 (32 ビット)
RND	RND	変換	丸め
RND+	RND+	変換	上位ダブル整数へ丸め
RND-	RND-	変換	下位ダブル整数へ丸め
RRD	RRD	シフト/ 循環	ダブルワード (32 ビット) の右回転
RRDA	RRDA	シフト/ 循環	CC 1 を介して右回転 ACCU 1 (32 ビット)
S	S	ビットロジック命令	セット
S	S	カウンタ	カウンタ事前設定値をセット (カレントカウンタ値は 0 から 255 まで有効、たとえば S C 15)
SA	SF	タイマ	オフディレイタイマ
SAVE	SAVE	ビットロジック命令	RLO を BR レジスタに保存
SE	SD	タイマ	オンディレイタイマ
SET	SET	ビットロジック命令	セット
SI	SP	タイマ	パルスタイマ
SIN	SIN	浮動小数点命令	角度のサインを浮動小数点数 (32 ビット) で生成
SLD	SLD	シフト/ 循環	ダブルワード (32 ビット) の左シフト
SLW	SLW	シフト/循環	ワード (16 ビット) の左シフト

すべての STL 命令の概要

A.1 ドイツ語のプログラム表記法 (SIMATIC) に従ってソートされた STL 命令

ドイツ語の プログラム 表記法	英語の プログラム 表記法	プログラムエレメント カタログ	説明
SPA	JU	ジャンプ	条件なしジャンプ
SPB	JC	ジャンプ	RLO = 1 ならばジャンプ
SPBB	JCB	ジャンプ	RLO = 1 ならば BR に保存してジャンプ
SPBI	JBI	ジャンプ	BR = 1 ならばジャンプ
SPBIN	JNBI	ジャンプ	BR = 0 ならばジャンプ
SPBN	JCN	ジャンプ	RLO = 0 ならばジャンプ
SPBNB	JNB	ジャンプ	RLO = 0 ならば BR に保存してジャンプ
SPL	JL	ジャンプ	ラベルへジャンプ
SPM	JM	ジャンプ	負ならばジャンプ
SPMZ	JMZ	ジャンプ	マイナスまたはゼロのときにジャンプ
SPN	JN	ジャンプ	ゼロ以外のときにジャンプ
SPO	JO	ジャンプ	OV = 1 ならばジャンプ
SPP	JP	ジャンプ	正ならばジャンプ
SPPZ	JPZ	ジャンプ	プラスまたはゼロのときにジャンプ
SPS	JOS	ジャンプ	OS = 1 のときにジャンプ
SPU	JUO	ジャンプ	アンオーダのときにジャンプ
SPZ	JZ	ジャンプ	ゼロのときにジャンプ
SQR	SQR	浮動小数点命令	浮動小数点数の 2 乗を生成 (32 ビット)
SQRT	SQRT	浮動小数点命令	浮動小数点数 (32 ビット) の平方根を生成
SRD	SRD	シフト/循環	ダブルワード (32 ビット) の右シフト
SRW	SRW	シフト/循環	ワード (16 ビット) の右シフト
SS	SS	タイマ	保持型オンディレイタイマ
SSD	SSD	シフト/循環	符号シフトダブル整数 (32 ビット)
SSI	SSI	シフト/循環	符号付き整数 (16 ビット) のシフト
SV	SE	タイマ	拡張パルスタイマ
T	T	ロード/転送	転送
T STW	T STW	ロード/転送	ACCU 1 をステータスワードへ転送
TAD	CAD	変換	ACCU 1 のバイトシーケンスを変更 (32 ビット)
TAK	TAK	アキュムレータ	ACCU 1 と ACCU 2 のトグル
TAN	TAN	浮動小数点命令	角度のタンジェントを生成 (32 ビット浮動小数点数)
TAR	CAR	ロード/転送	アドレスレジスタ 1 とアドレスレジスタ 2 の交換
TAR1	TAR1	ロード/転送	アドレスレジスタ 1 を ACCU 1 へ転送
TAR1	TAR1	ロード/転送	アドレスレジスタ 1 を宛先 (32 ビットポインタ) へ転送
TAR1	TAR1	ロード/転送	アドレスレジスタ 1 をアドレスレジスタ 2 へ転送
TAR2	TAR2	ロード/転送	アドレスレジスタ 2 を ACCU 1 へ転送
TAR2	TAR2	ロード/転送	アドレスレジスタ 2 を宛先 (32 ビットポインタ) へ転送
TAW	CAW	変換	ACCU 1-L のバイトシーケンスの変更 (16 ビット)
TDB	CDB	変換	共有 DB とインスタンス DB の交換
TRUNC	TRUNC	変換	切り捨て
U	A	ビットロジック命令	AND
U (	A (	ビットロジック命令	AND / ネストを開く
UC	UC	プログラムコントロール	条件なし呼び出し

A.1 ドイツ語のプログラム表記法 (SIMATIC) に従ってソートされた STL 命令

ドイツ語の プログラム 表記法	英語の プログラム 表記法	プログラムエレメント カタログ	説明
UD	AD	ワード論理命令	ダブルワード (32 ビット) の AND
UN	AN	ビットロジック命令	AND NOT
UN (	AN (	ビットロジック命令	AND NOT/ネストを開く
UW	AW	ワード論理命令	AND ワード (16 ビット)
X	X	ビットロジック命令	排他的 OR
X (	X (	ビットロジック命令	排他的 OR/ネストを開く
XN	XN	ビットロジック命令	排他的 OR NOT
XN (	XN (	ビットロジック命令	排他的 OR NOT/ネストを開く
XOD	XOD	ワード論理命令	排他的 OR ダブルワード (32 ビット)
XOW	XOW	ワード論理命令	ワード (16 ビット) の排他的 OR
ZR	CD	カウンタ	ダウンカウンタ
ZV	CU	カウンタ	カウントアップ

A.2 英語のプログラム表記法（インターナショナル）に従ってソートされた STL 命令

英語の プログラム 表記法	ドイツ語の プログラム 表記法	プログラムエレメント カタログ	説明
+	+	整数値演算命令	整数の追加（16、32 ビット）
=	=	ビットロジック命令	割り付け
)	)	ビットロジック命令	ネストを閉じる
+AR1	+AR1	アキュムレータ	AR1 ACCU 1 をアドレスレジスタ 1 へ加算
+AR2	+AR2	アキュムレータ	AR2 ACCU 1 をアドレスレジスタ 2 へ加算
+D	+D	整数値演算命令	ACCU 1 と ACCU 2 を加算（32 ビットダブル整数）
−D	−D	整数値演算命令	ACCU 2 から ACCU 1 を倍長整数として減算（32 ビット）
*D	*D	整数値演算命令	ACCU 1 と ACCU 2 の倍長整数（32 ビット）での乗算
/D	/D	整数値演算命令	倍長整数（32 ビット）として、ACCU 2 を ACCU 1 で割る
? D	? D	比較	倍長整数（32-Bit）の比較 ==, <>, >, <, >=, <=
+I	+I	整数値演算命令	ACCU 1 と ACCU 2 の整数（16 ビット）での加算
−I	−I	整数値演算命令	ACCU 2 から ACCU 1 を整数として減算（16 ビット）
*I	*I	整数値演算命令	ACCU 1 と ACCU 2 を整数として乗算（16 ビット）
/I	/I	整数値演算命令	整数（16 ビット）として、ACCU 2 を ACCU 1 で割る
? I	? I	比較	整数（16 ビット）の比較 ==, <>, >, <, >=, <=
+R	+R	浮動小数点命令	ACCU 1 と ACCU 2 の加算（32 ビットの IEEE 754 浮動小数点数）
−R	−R	浮動小数点命令	ACCU 2 から ACCU 1 を減算（32 ビットの IEEE 754 浮動小数点数）
*R	*R	浮動小数点命令	ACCU 1 と ACCU 2 の乗算（32 ビットの IEEE 754 浮動小数点数）
/R	/R	浮動小数点命令	ACCU 2 を ACCU 1 で除算（32 ビットの IEEE 754 浮動小数点数）
? R	? R	比較	浮動小数点数（32 ビット）の比較 ==, <>, >, <, >=, <=
A	U	ビットロジック命令	AND
A (	U (	ビットロジック命令	AND/ネストを開く
ABS	ABS	浮動小数点命令	浮動小数点数の絶対値 （32 ビットの IEEE 754）
ACOS	ACOS	浮動小数点命令	浮動小数点数（32 ビット）のアーコサインを生成
AD	UD	ワード論理命令	ダブルワード（32 ビット）の AND
AN	UN	ビットロジック命令	AND NOT
AN (	UN (	ビットロジック命令	AND NOT/ネストを開く
ASIN	ASIN	浮動小数点命令	浮動小数点数（32 ビット）のアークサインを生成
ATAN	ATAN	浮動小数点命令	浮動小数点数（32 ビット）のアークタンジェントを生成
AW	UW	ワード論理命令	AND ワード（16 ビット）
BE	BE	プログラムコントロール	ブロックの終了
BEC	BEB	プログラムコントロール	ブロックの条件付き終了
BEU	BEA	プログラムコントロール	ブロックの条件なし終了
BLD	BLD	プログラムコントロール	プログラム表示命令（Null）
BSD	BSD	変換	整数への BCD（32 ビット）
BTI	BTI	変換	整数への BCD（16 ビット）
CAD	TAD	変換	ACCU 1 のバイトシーケンスを変更（32 ビット）
CALL	CALL	プログラムコントロール	[ブロック呼び出し]

A.2 英語のプログラム表記法（インターナショナル）に従ってソートされた STL 命令

英語の プログラム 表記法	ドイツ語の プログラム 表記法	プログラムエレメント カタログ	説明
CALL	CALL	プログラムコントロール	複数インスタンスの呼び出し
CALL	CALL	プログラムコントロール	ライブラリからのブロックの呼び出し
CAR	TAR	ロード/転送	アドレスレジスタ 1 とアドレスレジスタ 2 の交換
CAW	TAW	変換	ACCU 1-L のバイトシーケンスの変更 (16 ビット)
CC	CC	プログラムコントロール	条件付き呼び出し
CD	ZR	カウンタ	ダウンカウンタ
CDB	TDB	変換	共有 DB とインスタンス DB の交換
CLR	CLR	ビットロジック命令	RLO のクリア (= 0)
COS	COS	浮動小数点命令	角度のコサインを生成 (32 ビット浮動小数点数)
CU	ZV	カウンタ	カウントアップ
DEC	DEC	アキュムレータ	ACCU 1-L-L のデクリメント
DTB	DTB	変換	BCD へのダブル整数 (32 ビット)
DTR	DTR	変換	倍長整数 (32 ビット) から浮動小数点数への変換 (32 ビット、IEEE 754)
ENT	ENT	アキュムレータ	ACCU スタックに入る
EXP	EXP	浮動小数点命令	浮動小数点数の指数値を生成 (32 ビット)
FN	FN	ビットロジック命令	立ち下がりパルス
FP	FP	ビットロジック命令	立ち上がりパルス
FR	FR	カウンタ	イネーブルカウンタ (フリー) (フリー、FR C 0 ~ o C 255)
FR	FR	タイマ	イネーブルタイマ (フリー)
INC	INC	アキュムレータ	ACCU 1-L-L のインクリメント
INVD	INVD	変換	1 の補数ダブル整数 (32 ビット)
INVI	INVI	変換	1 の補数整数 (16 ビット)
ITB	ITB	変換	整数 (16 ビット) から BCD への変換
ITD	ITD	変換	ダブル整数 (32 ビット) への整数 (16 ビット)
JBI	SPBI	ジャンプ	BR = 1 ならばジャンプ
JC	SPB	ジャンプ	RLO = 1 ならばジャンプ
JCB	SPBB	ジャンプ	RLO = 1 ならば BR に保存してジャンプ
JCN	SPBN	ジャンプ	RLO = 0 ならばジャンプ
JL	SPL	ジャンプ	ラベルへジャンプ
JM	SPM	ジャンプ	負ならばジャンプ
JMZ	SPMZ	ジャンプ	マイナスまたはゼロのときにジャンプ
JN	SPN	ジャンプ	ゼロ以外のときにジャンプ
JNB	SPBNB	ジャンプ	RLO = 0 ならば BR に保存してジャンプ
JNBI	SPBIN	ジャンプ	BR = 0 ならばジャンプ
JO	SPO	ジャンプ	OV = 1 ならばジャンプ
JOS	SPS	ジャンプ	OS = 1 のときにジャンプ
JP	SPP	ジャンプ	正ならばジャンプ
JPZ	SPPZ	ジャンプ	プラスまたはゼロのときにジャンプ
JU	SPA	ジャンプ	条件なしジャンプ
JUO	SPU	ジャンプ	アンオーダのときにジャンプ
JZ	SPZ	ジャンプ	ゼロのときにジャンプ
L	L	ロード/転送	ロード
L DBLG	L DBLG	ロード/転送	共有 DB の長さを ACCU 1 にロード
L DBNO	L DBNO	ロード/転送	共有 DB の数を ACCU 1 にロード

A.2 英語のプログラム表記法（インターナショナル）に従ってソートされた STL 命令

英語の プログラム 表記法	ドイツ語の プログラム 表記法	プログラムエレメント カタログ	説明
L DILG	L DILG	ロード/転送	インスタンス DB の長さを ACCU 1 にロード
L DINO	L DINO	ロード/転送	インスタンス DB の数を ACCU 1 にロード
L STW	L STW	ロード/転送	ステータスワードを ACCU 1 にロード
L	L	タイマ	カレントタイマ値を整数として ACCU 1 にロード（カレントタイマ値は 0 から 255 まで有効、たとえば L T 32）
L	L	カウンタ	カレントカウンタ値を ACCU 1 にロード（カレントカウンタ値は 0 から 255 まで有効、たとえば L C 15）
LAR1	LAR1	ロード/転送	ACCU 1 からアドレスレジスタ 1 をロード
LAR1 <D>	LAR1 <D>	ロード/転送	アドレスレジスタ 1 をダブル整数（32 ビットポインタ）でロード
LAR1 AR2	LAR1 AR2	ロード/転送	アドレスレジスタ 2 からアドレスレジスタ 1 をロード
LAR2	LAR2	ロード/転送	ACCU 1 からアドレスレジスタ 2 をロード
LAR2 <D>	LAR2 <D>	ロード/転送	ダブル整数（32 ビットポインタ）でアドレスレジスタ 2 をロード
LC	LC	カウンタ	カレントカウンタ値を BCD として ACCU 1 にロード（カレントタイマ値は 0 から 255 まで有効、たとえば LC C 15）
LC	LC	タイマ	カレントタイマ値を BCD として ACCU 1 にロード（カレントカウンタ値は 0 から 255 まで有効、たとえば LC T 32）
LEAVE	LEAVE	アキュムレータ	ACCU スタックから出る
LN	LN	浮動小数点命令	自然対数を浮動小数点数（32 ビット）で生成
LOOP	LOOP	ジャンプ	ループ
MCR (	MCR (	プログラムコントロール	MCR スタックに RLO を保存、MCR の開始
) MCR	) MCR	プログラムコントロール	MCR の終了
MCRA	MCRA	プログラムコントロール	MCR 領域の有効化
MCRD	MCRD	プログラムコントロール	MCR 領域の無効化
MOD	MOD	整数値演算命令	除算による余り（32 ビットダブル整数）
NEGD	NEGD	変換	倍長整数（32 ビット）の 2 の補数
NEGI	NEGI	変換	2 の補数整数（16 ビット）
NEGR	NEGR	変換	浮動小数点数（32 ビット、IEEE 754）の否定
NOP 0	NOP 0	アキュムレータ	Null 命令
NOP 1	NOP 1	アキュムレータ	Null 命令
NOT	NOT	ビットロジック命令	RLO の否定
O	O	ビットロジック命令	OR
O (	O (	ビットロジック命令	OR/ネストを開く
OD	OD	ワード論理命令	ダブルワード（32 ビット）の OR
ON	ON	ビットロジック命令	OR NOT
ON (	ON (	ビットロジック命令	OR NOT/ネストを開く
OPN	AUF	DB 呼び出し	データブロックを開く
OW	OW	ワード論理命令	OR ワード（16 ビット）
POP	POP	アキュムレータ	ACCU が 2 つの CPU
POP	POP	アキュムレータ	ACCU が 4 つの CPU
PUSH	PUSH	アキュムレータ	ACCU が 2 つの CPU
PUSH	PUSH	アキュムレータ	ACCU が 4 つの CPU
R	R	ビットロジック命令	リセット
R	R	カウンタ	カウンタをリセット（カレントカウンタ値は 0 から 255 まで有効、たとえば R C 15）

A.2 英語のプログラム表記法（インターナショナル）に従ってソートされた STL 命令

英語の プログラム 表記法	ドイツ語の プログラム 表記法	プログラムエレメント カタログ	説明
R	R	タイマ	タイマをリセット（カレントタイマ値は 0 から 255 まで有効、たとえば R T 32）
RLD	RLD	シフト/循環	左回転ダブルワード（32 ビット）
RLDA	RLDA	シフト/循環	CC 1 を介して ACCU 1 を左回転（32 ビット）
RND	RND	変換	丸め
RND-	RND-	変換	下位ダブル整数へ丸め
RND+	RND+	変換	上位ダブル整数へ丸め
RRD	RRD	シフト/循環	ダブルワード（32 ビット）の右回転
RRDA	RRDA	シフト/循環	CC 1 を介して右回転 ACCU 1（32 ビット）
S	S	ビットロジック命令	セット
S	S	カウンタ	カウンタ事前設定値をセット（カレントカウンタ値は 0 から 255 まで有効、たとえば S C 15）
SAVE	SAVE	ビットロジック命令	RLO を BR レジスタに保存
SD	SE	タイマ	オンディレイタイマ
SE	SV	タイマ	拡張パルスタイマ
SET	SET	ビットロジック命令	セット
SF	SA	タイマ	オフディレイタイマ
SIN	SIN	浮動小数点命令	角度のサインを浮動小数点数（32 ビット）で生成
SLD	SLD	シフト/循環	ダブルワード（32 ビット）の左シフト
SLW	SLW	シフト/循環	ワード（16 ビット）の左シフト
SP	SI	タイマ	パルスタイマ
SQR	SQR	浮動小数点命令	浮動小数点数の 2 乗を生成（32 ビット）
SQRT	SQRT	浮動小数点命令	浮動小数点数（32 ビット）の平方根を生成
SRD	SRD	シフト/循環	ダブルワード（32 ビット）の右シフト
SRW	SRW	シフト/循環	ワード（16 ビット）の右シフト
SS	SS	タイマ	保持型オンディレイタイマ
SSD	SSD	シフト/循環	符号シフトダブル整数（32 ビット）
SSI	SSI	シフト/循環	符号付き整数（16 ビット）のシフト
T	T	ロード/転送	転送
T STW	T STW	ロード/転送	ACCU 1 をステータスワードへ転送
TAK	TAK	アキュムレータ	ACCU 1 と ACCU 2 のトグル
TAN	TAN	浮動小数点命令	角度のタンジェントを生成（32 ビット浮動小数点数）
TAR1	TAR1	ロード/転送	アドレスレジスタ 1 を ACCU 1 へ転送
TAR1	TAR1	ロード/転送	アドレスレジスタ 1 を宛先（32 ビットポインタ）へ転送
TAR1	TAR1	ロード/転送	アドレスレジスタ 1 をアドレスレジスタ 2 へ転送
TAR2	TAR2	ロード/転送	アドレスレジスタ 2 を ACCU 1 へ転送
TAR2	TAR2	ロード/転送	アドレスレジスタ 2 を宛先（32 ビットポインタ）へ転送
TRUNC	TRUNC	変換	切り捨て
UC	UC	プログラムコントロール	条件なし呼び出し
X	X	ビットロジック命令	排他的 OR
X (	X (	ビットロジック命令	排他的 OR/ネストを開く
XN	XN	ビットロジック命令	排他的 OR NOT
XN (	XN (	ビットロジック命令	排他的 OR NOT/ネストを開く
XOD	XOD	ワード論理命令	排他的 OR ダブルワード（32 ビット）
XOW	XOW	ワード論理命令	ワード（16 ビット）の排他的 OR

B プログラミング例

B.1 プログラミングの概要例

実際の応用例

各ステートメントリスト命令は、特定の操作をトリガします。これらの命令を結合して1つのプログラムにすると、各種のオートメーションタスクを実行できます。この章では、以下のようなステートメントリスト命令の適用例を紹介しています。

- ビット論理命令によるコンベアベルトのコントロール
- ビットロジック命令を使用した搬送機ベルトの走行方向の検知
- タイマ命令を使用したクロックパルスの生成
- カウンタ命令と比較命令を使用した格納庫の管理
- 整数演算命令による問題の解決
- オープンの加熱時間の設定

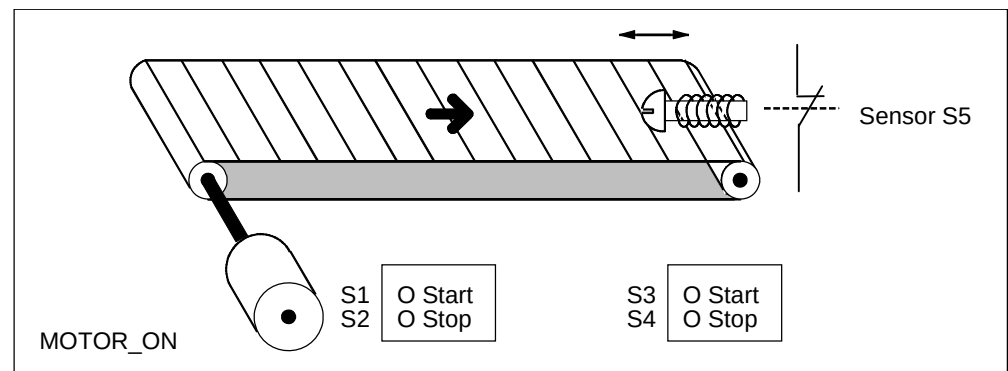
使用する命令

略号	プログラムエレメントカタログ	説明
AW	ワード論理命令	ワードの論理積 (AND WORD)
OW	ワード論理命令	ワードの論理和 (OR WORD)
CD、CU	カウンタ	カウントアップ、カウントダウン
S、R	ビット論理命令	設定、リセット
NOT	ビット論理命令	RLO の否定
FP	ビット論理命令	立ち上がりパルス
+I	浮動小数点命令	アキュムレータ 1 とアキュムレータ 2 加算 (整数)
/I	浮動小数点命令	アキュムレータ 2 をアキュムレータ 1 で除算 (整数)
*I	浮動小数点命令	アキュムレータ 1 とアキュムレータ 2 の乗算 (整数)
>=I、<=I	比較	整数の比較
A、AN	ビット論理命令	AND、AND NOT
O、ON	ビット論理命令	OR、OR NOT
=	ビット論理命令	割り付け
INC	アキュムレータ	アキュムレータ 1 をインクリメント
BE、BEC	プログラムコントロール	ブロック終了、およびと条件付きブロック終了
L、T	ロード/転送	ロードと転送
SE	タイマ	拡張パルスタイマ

B.2 例: ビットロジック命令

例 1: 搬送機ベルトの制御

以下の図に、電動式搬送機ベルトを示します。ベルトの始点には、START 用 S1 ボタンと STOP 用 S2 ボタンの 2 つの押しボタンがあります。ベルトの終点には、START 用 S3 スイッチと STOP 用 S4 スイッチの 2 つの押しボタンスイッチもあります。どちらの端からでもベルトを起動または停止できます。また、ベルト上の物体が終点に達すると、センサ S5 がこれを感じ、ベルトを停止させます。



絶対プログラミングとシンボルプログラミング

コンベアベルトをコントロールするプログラムを記述するには、コンベアシステムの各種コンポーネントを表す絶対値またはシンボルを使用します。

シンボルテーブルを 1 つ作成して、選択したシンボルと絶対値を対応させる必要があります。STEP 7 オンラインヘルプを参照してください。

システムコンポーネント	絶対アドレス	シンボル	シンボルテーブル
START ボタン	I 1.1	S1	I 1.1 S1
STOP ボタン	I 1.2	S2	I 1.2 S2
START ボタン	I 1.3	S3	I 1.3 S3
STOP ボタン	I 1.4	S4	I 1.4 S4
センサ	I 1.5	S5	I 1.5 S5
Motor	Q 4.0	MOTOR_ON	Q 4.0 MOTOR_ON

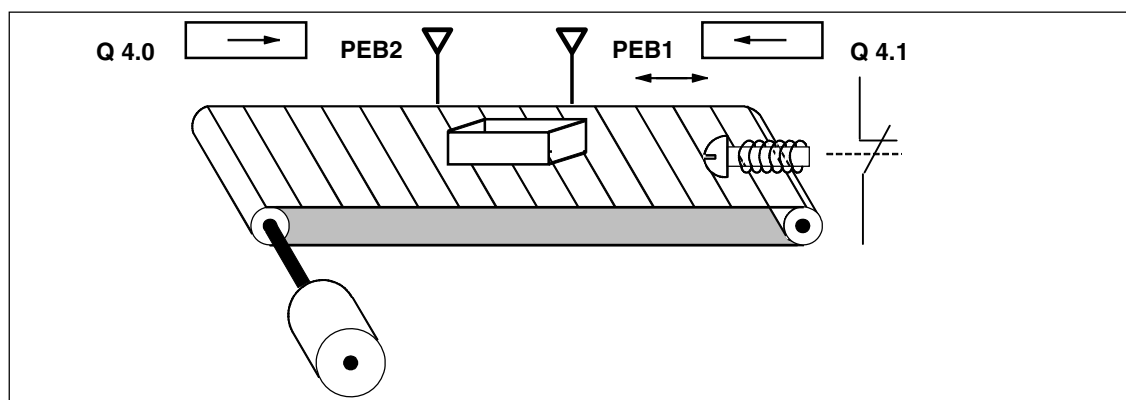
絶対アドレスを使ったプログラム	シンボルを使ったプログラム
O I 1.1	O S1
O I 1.3	O S3
S Q 4.0	S MOTOR_ON
O I 1.2	O S2
O I 1.4	O S4
ON I 1.5	ON S5
R Q 4.0	R MOTOR_ON

搬送機ベルトを制御するステートメントリスト

STL	説明
O I 1.1	//どちらかの START スイッチを押すと、モータがオンになります。
O I 1.3	
S Q 4.0	
O I 1.2	//どちらかの STOP スイッチを押すか、ベルト端末の b 接点を開くと、 //モータがオフになります。
O I 1.4	
ON I 1.5	
R Q 4.0	

例 2: 搬送機ベルトの走行方向の検知

以下の図に、光電スイッチを 2 個（PEB1 と PEB2）を装備した搬送機ベルトを示します。この 2 個の光電スイッチは、ベルト上のパッケージの移動方向を検知できます。各光電スイッチは、a 接点と同じように機能します。



絶対プログラミングとシンボルプログラミング

コンベアベルトシステムの方向表示を有効にするプログラムを記述するには、コンベアシステムの各種コンポーネントを表す**絶対値**または**シンボル**を使用します。

シンボルテーブルを 1 つ作成して、選択したシンボルと絶対値を対応させる必要があります。STEP 7 オンラインヘルプを参照してください。

システムコンポーネント	絶対アドレス	シンボル	シンボルテーブル
光電スイッチ 1	I 0.0	PEB1	I 0.0 PEB1
光電スイッチ 2	I 0.1	PEB2	I 0.1 PEB2
右移動の表示	Q 4.0	RIGHT	Q 4.0 RIGHT
左移動の表示	Q 4.1	LEFT	Q 4.1 LEFT
パルスメモリビット 1	M 0.0	PMB1	M 0.0 PMB1
パルスメモリビット 2	M 0.1	PMB2	M 0.1 PMB2

絶対アドレスを使ったプログラム	シンボルを使ったプログラム
A I 0.0	A PEB1
FP M 0.0	FP PMB1
AN I 0.1	AN PEB 2
S Q 4.1	S LEFT
A I 0.1	A PEB 2
FP M 0.1	FP PMB 2
AN I 0.0	AN PEB 1
S Q 4.0	S RIGHT
AN I 0.0	AN PEB 1
AN I 0.1	AN PEB 2
R Q 4.0	R RIGHT
R Q 4.1	R LEFT

ステートメントリスト

STL	説明
A I 0.0	//入力 I 0.0 で信号状態が 0 から 1 に遷移し（信号立ち上がり）、同時に、 //入力 I 0.1 の信号状態が 0 になっている場合、 //ベルト上のパッケージは左方向へ移動しています。
FP M 0.0	
AN I 0.1	
S Q 4.1	
A I 0.1	//入力 I 0.1 で信号状態が 0 から 1 に遷移し（信号立ち上がり）、同時に、 //入力 I 0.0 の信号状態が 0 になっている場合、 //ベルト上のパッケージは右方向へ移動しています。一方の光電スイッチをパッケージが通過すると、 //2 個の光電スイッチの間にパッケージがあることがわかります。
FP M 0.1	
AN I 0.0	
S Q 4.0	
AN I 0.0	//どちらの光電スイッチもパッケージが通過しないと、 //2 個の光電スイッチの間にパッケージがないことがわかります。この場合、走行方向は表示されません。
AN I 0.1	
R Q 4.0	
R Q 4.1	

B.3 例: タイマ命令

クロックパルスジェネレータ

周期反復信号の生成が必要な場合、クロックパルスジェネレータまたはフラッシュ信号を使用できます。これは、表示ランプの点滅を制御する信号システムによく使用されています。

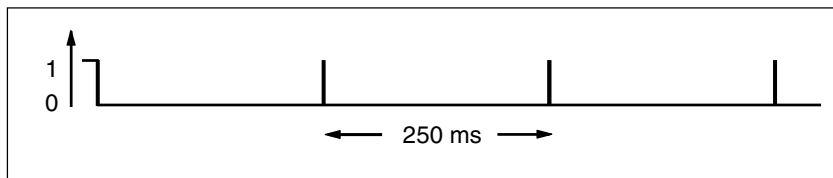
S7-300 を使用する場合、特殊なオーガニゼーションブロックでタイムドリブン処理を使用すれば、クロックパルスジェネレータファンクションを実行できます。ただし、以下のステートメントリストに示されている例で、タイマファンクションを使用してクロックパルスを生成する方法について説明します。このサンプルプログラムで、タイマによるフリーホイーリングクロックパルスジェネレータの実現方法について説明します。

クロックパルスを生成するステートメントリスト（パルスデューティファクタ 1:1）

STL	説明
AN T1	//タイマ T1 が満了した場合、
L S5T#250ms	//時間値 250 ms を T1 にロードする。および
SV T1	//T1 を拡張パルスタイマとして起動する。
NOT	//論理演算の結果を否定（反転）する。
BEB	//タイマが実行されている場合、現在のブロックを終了する。
L MB100	//タイマが満了している場合、メモリバイト MB100 の内容をロードする。
INC 1	//内容を 1 だけ増加し、
T MB100	//その結果をメモリバイト MB100 にロードする。

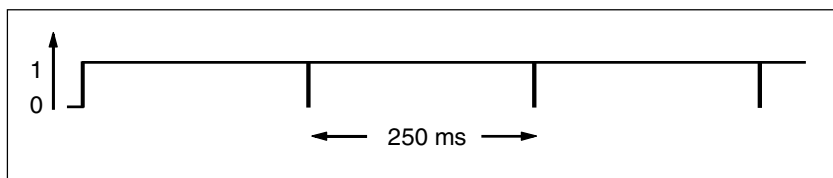
信号チェック

タイマ T の信号チェックにより、以下の論理演算結果（RLO）が生成されます。



タイマは満了すると直ちに再起動します。このため、ステートメント **AN T1** で実行された信号チェックにより、ほんの少しの間、信号状態 1 が生成されます。

否定（反転）RLO:



RLO ビットは、250ms ごとに 0 になります。その場合、BEC ステートメントは、ブロックの処理を終了しません。代わりに、メモリバイト MB100 の内容が 1 増加します。

メモリバイト MB100 の内容は、250 ms ごとに以下のように変わります。

0 -> 1 -> 2 -> 3 -> ... -> 254 -> 255 -> 0 -> 1 ...

特定周波数への到達

メモリバイト MB100 の個々のビットから、以下の周波数に到達できます。

MB100 のビット	周波数 (Hz)	パルス幅
M 100.0	2.0	0.5 s (250 ms オン/ 250 ms オフ)
M 100.1	1.0	1 s (0.5 s オン/ 0.5 s オフ)
M 100.2	0.5	2 s (1 s オン/ 1 s オフ)
M 100.3	0.25	4 s (2 s オン/ 2 s オフ)
M 100.4	0.125	8 s (4 s オン/ 4 s オフ)
M 100.5	0.0625	16 s (8 s オン/ 8 s オフ)
M 100.6	0.03125	32 s (16 s オン/ 16 s オフ)
M 100.7	0.015625	64 s (32 s オン/ 32 s オフ)

ステートメントリスト

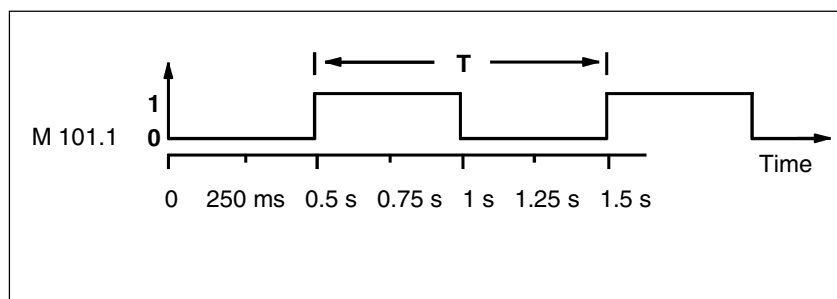
STL	説明
A M10.0	//故障が発生すると、M 10.0 = 1 になる。故障が発生すると、故障表示ランプが周波数 1Hz
	//で点滅する。
A M100.1	
= Q 4.0	

メモリ MB 101 の各ビットの信号状態

スキャン サイクル	ビット 7	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2	ビット 1	ビット 0	時間値 (単位: 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

MB 101 (M 101.1) のビット 1 の信号状態

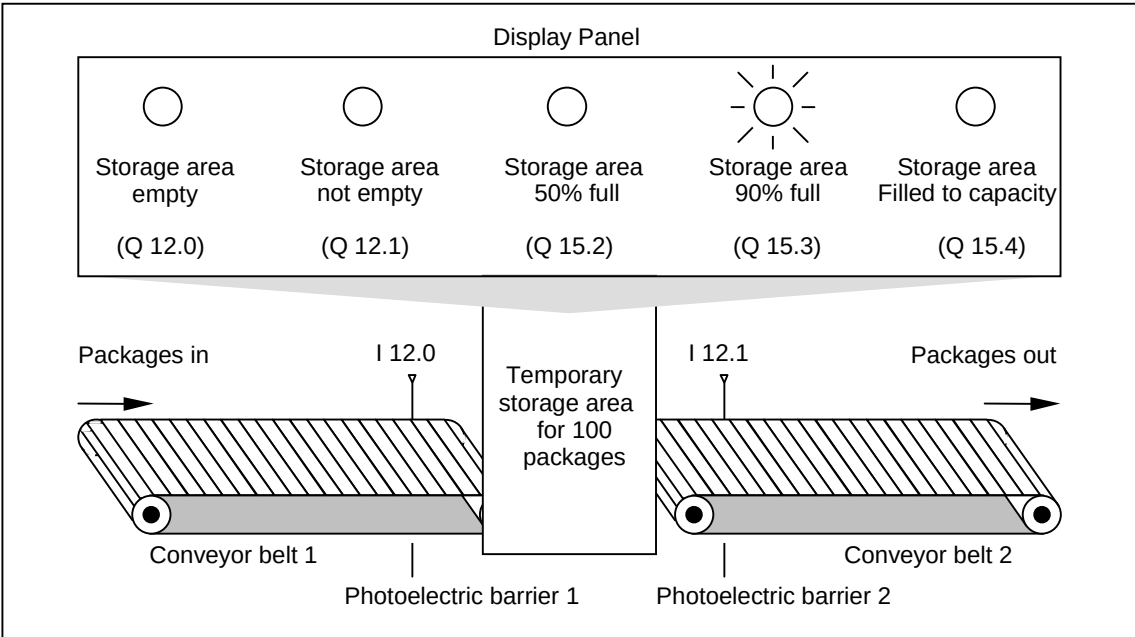
周波数 = $1/T = 1/1\text{ s} = 1\text{ Hz}$



B.4 例: カウンタ命令と比較命令

カウンタ命令と比較命令を使用した格納庫の管理

以下の図に、搬送機ベルトを2つ装備し、この2つのベルトの間にテンポラリ格納庫を装備したシステムを示します。搬送機ベルト1は、パッケージを格納庫まで配送します。搬送機ベルト1の末端、格納庫側にある光電スイッチにより、格納庫に搬入された荷物の数が確定されます。搬送機ベルト2は、パッケージをこのテンポラリ格納庫から積載ドックへ搬送します。パッケージは、この積載ドックからトラックで発送され、顧客に配送されます。搬送機ベルト2の末端、格納庫側にある光電スイッチにより、格納庫から積載ドックへ搬出された荷物の数が確定されます。テンポラリ格納庫の格納率は、表示パネルの5つのランプで示されます。



表示パネルの表示ランプ点灯のステートメントリスト

STL	説明
A I 0.0	//光源スイッチ 1 でパルスが生成されるたびに、
CU C1	//カウンタ C1 のカウント値が 1 だけ増加する。これにより、 //格納庫に搬入されたパッケージの数がカウントされる。 //
A I 0.1	//光源スイッチ 2 でパルスが生成されるたびに、
CD C1	//カウンタ C1 のカウント値が 1 だけ増加する。これにより、 //格納庫から搬出されたパッケージの数がカウントされる。 //
AN C1	//カウント値が 0 の場合、
= Q 4.0	//"格納率 0%"に対応するインジケータランプが点灯する。 //
A C1	//カウント値が 0 でない場合、
= A 4.1	//"格納率が 0%でない"に対応するインジケータランプが点灯する。 //
L 50	
L C1	
<=I	//カウント値が 50 以上の場合、
= Q 4.2	//"格納率 50%"に対応するインジケータランプが点灯する。 //
L 90	
>=I	//カウント値が 90 以上の場合、
= Q 4.3	//"格納率 90%"に対応するインジケータランプが点灯する。 //
L Z1	
L 100	
>=I	//カウント値が 100 以上の場合、
= Q 4.4	//"格納率 100%"に対応するインジケータランプが点灯する。（出力 Q 4.4 を使って、 //搬送機ベルト 1 を停止させることもできます）

B.5 例: 整数値演算命令

演算問題の解決

このサンプルプログラムでは、3つの整数値演算命令を使用して、以下の方程式と同じ結果を求める方法を示します。

$$MD4 = ((IW0 + DBW3) \times 15) / MW2$$

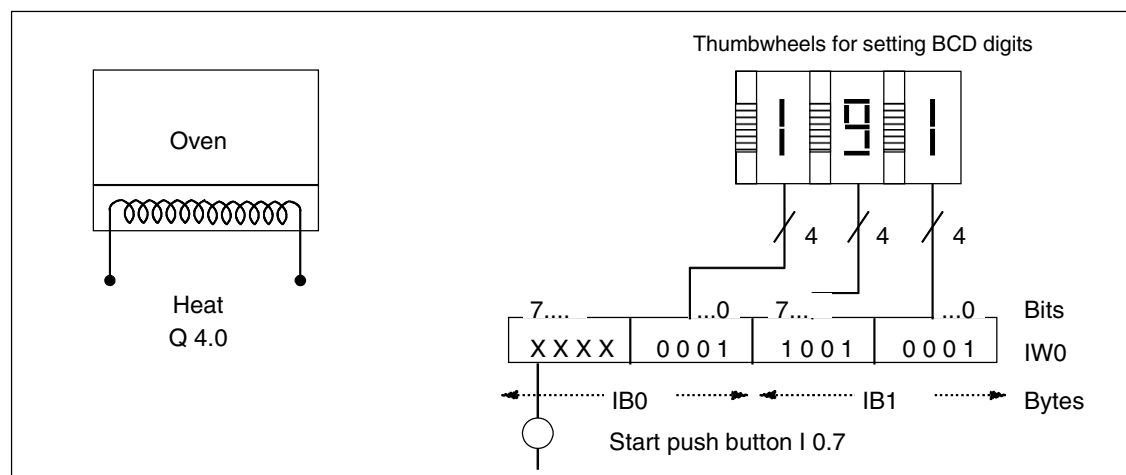
ステートメントリスト

STL	説明
L EW0	//入力ワード IW0 の値をアキュムレータ 1 へロードする。
L DB5.DBW3	//DB5 の共有データワード DBW3 をアキュムレータ 1 にロードする。 //アキュムレータ 1 の元の内容は、アキュムレータ 2 にシフトされる。
+I I 0.1	//アキュムレータ 1 とアキュムレータ 2 の下位ワードの内容を加算する。この結果は、 //アキュムレータ 1 の下位ワードに保存されます。アキュムレータ 1 の上位ワードと //アキュムレータ 2 の内容は変更されない。
L +15	//定数値+15 をアキュムレータ 1 にロードする。//アキュムレータ 1 の //元の内容は、アキュムレータ 2 にシフトされる。
*I	//アキュムレータ 2 の下位ワードの内容を、アキュムレータ 1 の //下位ワードの内容で乗算する。この結果は、アキュムレータ 1 に保存されます。 //アキュムレータ 2 の内容は変化しない。
L MW2	//メモリワード MW2 の値をアキュムレータ 1 へロードする。 //アキュムレータ 1 の元の内容は、アキュムレータ 2 にシフトされる。
//	//アキュムレータ 2 の下位ワードの内容を、アキュムレータ 1 の //下位ワードの内容で除算する。この結果は、アキュムレータ 1 に保存されます。 //アキュムレータ 2 の内容は変化しない。
T MD4	//最終結果をメモリダブルワード MD4 へ転送する。両方のアキュムレータの内容は、 //変更されずに残ります。

B.6 例: ワードロジック命令

オーブンの加熱

オペレータが START 押しボタンを押すと、オーブンは加熱を開始します。オペレータは、図に示されているサムホールスイッチを使用すれば、加熱時間を設定できます。設定した値は、2 進化 10 進数 (BCD) フォーマットの秒数で示されます。



システムコンポーネント	絶対アドレス
START 押しボタン	I 0.7
1 の位のサムロータリースイッチ	I 1.0 ~ I 1.3
10 の位のサムロータリースイッチ	I 1.4 ~ I 1.7
100 の位のサムロータリースイッチ	I 0.0 ~ I 0.3
加熱開始	Q 4.0

ステートメントリスト

STL	説明
A T1	//タイマが実行されている場合、
= Q 4.0	//加熱をオンにする。
BEC	//タイマが実行されている場合、ここで加熱処理を終了する。この結果、この押しボタンを押しても、 //タイマ T1 を再起動できなくなる。
L IW0	
AW W#16#0FFF	//I 0.4~I 0.7 の入力ビットをマスクする (つまり、これらのビットを 0 にリセットする)。 //秒単位のタイマ値は、BCD 表記でアキュムレータ 1 の //下位ワードに入っています。
OW W#16#2000	//アキュムレータ 1 の下位ワードのビット 12 と 13 に、秒単位のタイムベースを割り当てます。
A I 0.7	
SE T1	//この押しボタンを押すと、タイマ T1 が拡張パルスタイマとして起動する。

C パラメータ転送

ブロックのパラメータは、値として転送されます。ファンクションブロックでは、呼び出されたブロックで、インスタンスデータブロックの実パラメータ値が使用されます。ファンクションでは、現在値のコピーがローカルデータスタックに格納されます。ポインタはコピーされません。呼び出し前、入力値がインスタンス DB または L スタックにコピーされます。呼び出し後、出力値が変数にコピーされます。呼び出されたブロック内では、コピー操作しか実行できません。この操作に必要な STL 命令は、呼び出し側ブロック内にあるため、ユーザーから隠されていることに変わりはありません。

注記

メモリビット、入力、出力、または周辺 I/O は、ファンクションの実アドレスとして使用される場合、他のアドレスと異なる方法で処理されます。ここでは、更新は、直接実行されるため、L スタックは使用されません。



注意

呼び出される側のブロックをプログラミングする場合には、出力として宣言されたパラメータも作成されていることを確認してください。このパラメータが作成されていないと、出力値はランダムになります! ファンクションブロックでは、値は、最後の呼び出しで示されたインスタンス DB の値となり、ファンクションでは、その時点で L スタックに格納されていた値になります。

以下の点を注意してください。

- 可能なら出力パラメータをすべて初期化します。
 - SetおよびResetの各命令を使用しないでください。こうした命令は、RLOによって違ってきます。RLOの値が0の場合、ランダムな値が保持されます。
 - ブロック内でジャンプする場合、出力パラメータが作成された位置をスキップしないでください。BEC、およびMCR命令の結果を忘れないでください。
-

索引

)

) , 25

) MCR, 169

*D, 112

*I, 105

*R, 120

／

/D, 113

/I, 106, 107

/R, 121

?

? D, 41

? I, 40

+

+, 108

+AR1, 238

+AR2, 239

+D, 110

+I, 103

+R, 117, 118

=

=, 27

A

A, 15

A (, 22

ABS, 122

ACCU 1 からアドレスレジスタ 1 をロード, 137

ACCU 1 からアドレスレジスタ 2 をロード, 139

ACCU 1 のビット構成, 195

ACOS, 131

AD, 222, 223

AN, 16

AN (, 23

AND, 15

AND NOT, 16

AND の後に OR, 21

ASIN, 130

ATAN, 132

AW, 216, 217

B

BE, 148

BEC, 149

BEU, 150

BLD, 240

BTD, 46

BTI, 44

C

CAD, 56

CALL, 151, 152, 153

CAR, 143

CAW, 55

CC, 162

CD, 69

CDB, 73

CLR, 32

COS, 128

CU, 68

D

-D, 111

DEC, 237

DTB, 48

DTR, 49

E

ENT, 235

EXP, 125

F

FB の呼び出し, 154

FC の呼び出し, 156
FN, 34
FP, 36
FR, 62, 197

I

-I, 104
INC, 236
INVD, 51
INVI, 50
ITB, 45
ITD, 47

J

JB, 86
JC, 82
JCB, 84
JCN, 83
JL, 80, 81
JM, 94
JMZ, 96
JN, 92
JNB, 85
JNBI, 87
JO, 88
JOS, 89, 90
JP, 93
JPZ, 95
JU, 79
JUO, 97, 98
JZ, 91

L

L, 134, 199
L DBLG, 73
L DBNO, 74
L DILG, 74
L DINO, 75
L STW, 136
LAR1, 137
LAR1 <D> アドレスレジスタ 1 を倍長整数 (32 ビットポ
インタ) と共にロード, 138
LAR1 AR2, 139
LAR2, 139
LAR2 <D>, 140
LC, 201, 202
LEAVE, 235

LN, 126
LOOP, 99

M

MCR, 170, 171
MCR (, 167, 168
MCR (マスタコントロールリレー) , 164
MCRA, 170
MCRD, 171
MCR の終了, 169
MCR ファンクションの使用方法に関する重要事項, 166
MCR 領域, 168, 169, 170
MCR 領域の無効化, 171
MCR 領域の有効化, 170
MOD, 114

N

NEGD, 53
NEGI, 52
NEGR, 54
NOP 0, 240
NOP 1, 241
NOT, 30

O

O, 17, 21
O (, 23
OD, 224, 225
ON, 18
ON (, 24
OPN, 72
OR, 17
OW, 218, 219

P

POP, 231, 232
PUSH, 233, 234

R

R, 28, 66, 203
-R, 119
-R, 119
RLD, 187, 188
RLDA, 191
RLO の設定 (= 1) , 30
RLO を MCR スタックに保存し
MCR オン, 167

RND, 57
 RND-, 60
 RND-, 60
 RND-, 60
 RND-, 60
 RND-, 60
 RND+, 59
 RRD, 189, 190
 RRDA, 192

S

S, 29, 67
 SAVE, 33
 SD, 208, 209
 SE, 206, 207
 SET, 30
 SF, 212, 213
 SFB の呼び出し, 158
 SFC の呼び出し, 160
 SIN, 127
 SLD, 182, 183
 SLW, 178, 179
 SP, 204, 205
 SQR, 123
 SQRT, 124
 SRD, 184, 185
 SRW, 180, 181
 SS, 210, 211
 SSD, 176, 177
 SSI, 174

T

T, 141
 T STW, 142
 TAK, 230
 TAN, 129
 TAR1, 143
 TAR1 <D>, 144
 TAR1 AR2, 145
 TAR2, 145
 TAR2 <D>, 146
 TRUNC, 58

U

UC, 163

X

X, 19
 X (, 24
 XN, 20
 XN (, 25
 XOD, 226, 227
 XOW, 220, 221

あ

アキュムレータの演算とアドレスレジスタ命令, 229
 アドレスレジスタ 1 を ACCU 1 へ転送, 143
 アドレスレジスタ 1 を宛先 (32 ビットポインタ) へ転送,
 144
 アドレスレジスタ 1 をアドレスレジスタ 2 へ転送, 145
 アドレスレジスタ 2 からアドレスレジスタ 1 をロード,
 139
 アドレスレジスタ 2 を ACCU 1 へ転送, 145
 アドレスレジスタ 2 を宛先 (32 ビットポインタ) へ転送,
 146

い

インスタンス DB の数を ACCU 1 にロード, 75
 インスタンス DB の長さを ACCU 1 にロード, 74

え

英語のプログラム表記法 (インターナショナル) に従って
 ソートされた STL 命令, 248

お

オンラインヘルプ, 5

か

カウンタ値, 61
 カウンタの有効化 (空き), 62
 カウンタのリセット, 66
 カウンタプリセット値の設定, 67
 カウンタ命令の概要, 61

き

共有 DB とインスタンス DB の交換, 73
 共有 DB の数を ACCU 1 にロード, 74
 共有 DB の長さを ACCU 1 にロード, 73

け

現在のタイマ値を ACCU 1 に整数でロード, 199

し

時間値, 194, 195, 196

四捨五入, 57

システムファンクションブロック呼び出し, 159

システムファンクション呼び出し, 160

実践的な応用例, 253

シフト命令の概要, 173

循環命令の概要, 186

せ

整数 (16 ビット) の比較, 40

整数値演算命令におけるステータスワードのビットの評価, 102

整数値演算命令の概要, 101

整定数の加算 (16 ビット、32 ビット), 108

セット, 29

た

タイマのコンポーネント, 193, 194

タイマのリセット, 203

タイムベース, 194, 195

ダブル整数 (32 ビット) への整数 (16 ビット), 47

て

データブロック命令の概要, 71

適切なタイマの選択, 196

転送, 141

と

ドイツ語のプログラム表記法 (SIMATIC) に従ってソートされた STL 命令, 243

ね

ネストを閉じる, 25

は

排他的 OR, 19

倍長整数 (32 ビット) の比較, 41

パラメータ転送, 265

ひ

比較命令の概要, 39

ビット論理命令の概要, 13

ふ

ファンクションブロック呼び出し, 155

ファンクション呼び出し, 156

複数インスタンスの呼び出し, 161

浮動小数点数 (32 ビット) の比較, 42

浮動小数点数 (32 ビット) の比較

IEEE 754) の否定, 54

浮動小数点数値演算命令におけるステータスワードのビットの評価, 116

浮動小数点数値演算命令の概要, 115

プログラミングの概要例, 253

プログラム制御命令の概要, 147

プログラム表記法

英語, 248

プログラム表記法 ドイツ語/SIMATIC, 243

プログラム表示命令, 240

へ

変換命令の概要, 43

め

メモリ内のタイマの位置, 193, 194

メモリ内の領域, 61, 194

ら

ライブラリからのブロックの呼び出し, 161

り

リセット, 28

る

ループ, 99

れ

例, 253

カウンタ命令と比較命令, 260

整数演算命令, 262

タイマ命令, 257

ビットロジック命令, 254

ワード論理命令, 263

ろ

ロード, 135

ロード命令と転送命令の概要, 133

ロジックコントロール命令の概要, 77

わ

ワード論理命令の概要, 215

