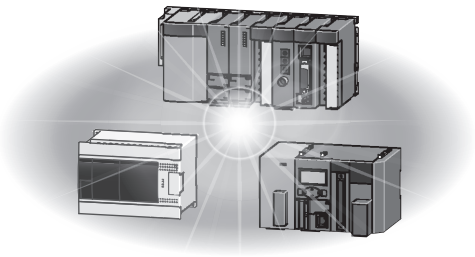
MITSUBISHI ELECTRIC

三菱電機 汎用 シーケンサ

MELSEC Q series MELSEC L series

プログラミングマニュアル (ストラクチャードテキスト編)



● 安全上のご注意 ●

（ご使用前に必ずお読みください）

MELSEC-Qシリーズ，MELSEC-Lシリーズシーケンサのご使用に際しては，各製品に付属しているマニュアルおよび付属マニュアルで紹介している関連マニュアルをよくお読みいただくと共に，安全に対して十分に注意を払って，正しい取扱いをしていただくようお願いいたします。

製品に付属しているマニュアルは必要なときに取り出して読めるように大切に保管すると共に，必ず最終ユーザまでお届けいただくようにお願いいたします。

A－1

● 製品の適用について ●

(1) 当社シーケンサをご使用いただくにあたりましては，万一シーケンサに故障・不具合などが発生した場合でも重大な事故にいたらない用途であること，および故障・不具合発生時にはバックアップやフェールセーフ機能が機器外部でシステム的に実施されていることをご使用の条件とさせていただきます。

(2) 当社シーケンサは，一般工業などへの用途を対象とした汎用品として設計・製作されています。したがいまして，以下のような機器・システムなどの特殊用途へのご使用については，当社シーケンサの適用を除外させていただきます。万一使用された場合は当社として当社シーケンサの品質，性能，安全に関る一切の責任（債務不履行責任，瑕疵担保責任，品質保証責任，不法行為責任，製造物責任を含むがそれらに限定されない）を負わないものとさせていただきます。

・ 各電力会社殿の原子力発電所およびその他発電所向けなどの公共への影響が大きい用途
・ 鉄道各社殿および官公庁殿など，特別な品質保証体制の構築を当社にご要求になる用途
・ 航空宇宙，医療，鉄道，燃焼・燃料装置，乗用移動体，有人搬送装置，娯楽機械，安全機械など生命，身体，財産に大きな影響が予測される用途

ただし，上記の用途であっても，具体的に使途を限定すること，特別な品質（一般仕様を超えた品質等）をご要求されないこと等を条件に，当社の判断にて当社シーケンサの適用可とする場合もございますので，詳細につきましては当社窓口へご相談ください。

A - 2

改 訂 履 歴

※取扱説明書番号は，本説明書の裏表紙の左下に記載してあります。

印刷日付	※取扱説明書番号	改　定　内　容
2002年12月	SH(名)-080363-A	初版印刷
2003年 3月	SH(名)-080363-B	一部修正 目次，1.2節，4.4節，5.1節，5.2節，6.1節，7章
2003年 7月	SH(名)-080363-C	一部修正 6.7.1項，6.7.2項，6.7.3項，6.9.1項，6.9.2項，6.9.3項，6.9.4項，6.9.5項，6.9.6項，7章
2004年 5月	SH(名)-080363-D	機種追加 Q12PRHCPU，Q25PRHCPU 新規追加 保証について 一部修正 本マニュアルで使用する総称・略称，2.1.1項，2.1.3項，5章，6章，7章
2006年 2月	SH(名)-080363-E	一部修正 6.1.14項
2008年 5月	SH(名)-080363-F	一部修正 本マニュアルで使用する総称・略称，2.1.1項，保証について
2008年10月	SH(名)-080363-G	機種追加 Q00UJCPU，Q00UCPU，Q01UCPU，Q10UDHCPU，Q10UDEHCPU，Q20UDHCPU，Q20UDEHCPU 一部修正 本マニュアルで使用する総称・略称，2.1.1項，3.2.1項，4.2.2項
2009年 1月	SH(名)-080363-H	一部修正 2.1.3項，5章
2009年12月	SH(名)-080363-I	機種追加 L02CPU，L26CPU-BT 一部追加 製品の適用について，付2 一部修正 安全上のご注意，マニュアルについて， 本マニュアルで使用する総称・略称，1.2節，2.1.1項，2.1.3項，3.1節，3.3節，3.3.1項，3.3.3項，4.2.1項，4.2.2項，5章，5.16.1～5.16.4項，5.20.1項，5.20.2項，6章，7章，付1
2010年10月	SH(名)-080363-J	一部修正 3.2.3項，5.16.1項，5.16.2項，5.19.5項，5.19.6項

A－3

印刷日付	※取扱説明書番号	改　定　内　容
2014年10月	SH(名)-080363-K	一部修正 マニュアルについて，マニュアルの見方，4.3.3項
2019年 2月	SH(名)-080363-L	一部修正 本マニュアルで使用する総称・略称，5.10.6項

A - 4

は　じ　め　に

このたびは，三菱統合FAソフトウェアMELSOFTシリーズをお買い上げ頂き，誠にありがとうございました。

ご使用前に本書をよくお読み頂き，MELSECシリーズシーケンサの機能・性能を十分ご理解のうえ，正しくご使用くださるようお願い致します。

なお，本マニュアルにつきましては最終ユーザまでお届け頂きますよう，宜しくお願い申し上げます。

目　　次

5.10.11　排他的論理和(2デバイス)・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ WXOR_M・・・・・・・・・・・・・・・・・ 5-44
5.10.12　排他的論理和(3デバイス)・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ WXOR_3_M・・・・・・・・・・・・・・・・ 5-44
5.10.13　32ビットデータ排他的論理和(2デバイス)・・・・・・・・・・・・・・・・・・ DXOR_M・・・・・・・・・・・・・・・・・ 5-45
5.10.14　32ビットデータ排他的論理和(3デバイス)・・・・・・・・・・・・・・・・・・ DXOR_3_M・・・・・・・・・・・・・・・・ 5-45
5.10.15　ブロックデータ排他的論理和・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ BKXOR_M・・・・・・・・・・・・・・・・ 5-46
5.10.16　否定排他的論理和(2デバイス)・・・・・・・・・・・・・・・・・・・・・・・・・・ WXNR_M・・・・・・・・・・・・・・・・・ 5-46
5.10.17　否定排他的論理和(3デバイス)・・・・・・・・・・・・・・・・・・・・・・・・・・ WXNR_3_M・・・・・・・・・・・・・・・・ 5-47
5.10.18　32ビットデータ否定排他的論理和(2デバイス)・・・・・・・・・・・・・・ DXNR_M・・・・・・・・・・・・・・・・・ 5-47
5.10.19　32ビットデータ否定排他的論理和(3デバイス)・・・・・・・・・・・・・・ DXNR_3_M・・・・・・・・・・・・・・・・ 5-48
5.10.20　ブロックデータ否定排他的論理和・・・・・・・・・・・・・・・・・・・・・・・・・・ BKXNR_M・・・・・・・・・・・・・・・・ 5-48
5.11　ローテーション・・・・・・・・・・・・・・・・・・・

マニュアルについて

本製品に関連するマニュアルには，下記のものがあります。
必要に応じて本表を参考にしてご依頼ください。

関連マニュアル

マニュアル名称	マニュアル番号(形名コード)	領布価格
GX Developer Version8 オペレーティングマニュアル　(スタートアップ編) GX Developerのシステム構成，インストール方法，立上げ方法について説明しています。 (別売)	SH-080355 (13JV68)	¥1000
GX Developer Version8 オペレーティングマニュアル GX Developerでのプログラムの作成方法，プリントアウト方法，モニタ方法，デバッグ方法などについて説明しています。 (別売)	SH-080356 (13JV69)	¥4000
GX Developer Version8 オペレーティングマニュアル（ファンクションブロック編） GX Developerでのファンクションブロックの作成方法，プリントアウト方法などについて説明しています。 (別売)	SH-080359 (13JV72)	¥1500
GX Developer Version8 オペレーティングマニュアル（ストラクチャードテキスト編） GX Developerでのストラクチャードテキスト（ST）プログラムの作成方法，プリントアウト方法などについて説明しています。 (別売)	SH-080364 (13JV73)	¥1500
ストラクチャードテキスト（ST）プログラミングガイドブック 初めてストラクチャードテキスト(ST)プログラムを作成する方を対象としています。サンプルプログラムを通して基本的な操作方法や機能を説明しています。 (別売)	SH-080365 (13JC12)	¥1000
MELSEC-Q/L プログラミングマニュアル（共通命令編） シーケンス命令，基本命令および応用命令の使用方法について説明しています。 (別売)	SH-080804 (13JC22)	¥4000
はじめようGX Works2 （構造化プロジェクト編） GX Works2でのストラクチャードテキストのプログラミング方法について説明しています。 (別売)	SH-080734 (13JY65)	¥1500

備　考

各オペレーティングマニュアルとストラクチャードテキスト（ST）プログラミングガイドブックは，ソフトウェアパッケージのCD-ROMにPDFファイルで格納されています。
単品でマニュアルを希望する場合は，印刷物を別売で用意していますので上記表のマニュアル番号（形名コード）にてご用命願います。

A - 13

マニュアルの見方

このマニュアル．．．

本マニュアルは，GX Developerを使ってストラクチャードテキスト（以下，STと略す）プログラミングを行うために使用してください。シーケンサ・ラダープログラムについての知識がありプログラミング経験があるユーザ，Ｃ言語についての知識がありプログラミング経験があるユーザに適しています。

『第1章 概要』は，ST言語の概要，STプログラミングの特長，STプログラムの作成手順を記載しています。

『第2章 システム構成』は，適用CPU，STプログラム仕様などを記載しています。

『第3章 STプログラムの文字や数値の扱い』は，STプログラムで使用するデータの型や表記方法を記載しています。

『第4章 STプログラムの式』は，STプログラムで使用する演算子，制御構文などの式について記載しています。

『第5章 MELSEC関数』，『第6章 IEC関数』は，STプログラムで使用する関数の引数・戻り値・記述例を記載しています。

オペレーティングマニュアル．．．

『GX Developer Version8 オペレーティングマニュアル（ストラクチャードテキスト編）』は，STプログラミングを行うために使用するすべてのメニューとメニューオプションの詳しい説明で構成されています。操作の詳細についての情報が必要な場合に参照してください。

STプログラミング以外の操作の情報が必要な場合には『GX Developer Version8 オペレーティングマニュアル』，もしくは『GX Developer Version8 オペレーティングマニュアル（スタートアップ編）』を参照してください。

ST言語の知識があり，すぐにプログラミングを行いたい場合は．．．

『第5章 MELSEC関数』に進んでください。STプログラムで関数を使うための必要事項が記載されています。STプログラムで使用するデータについて知りたい場合は，『第3章 STプログラムの文字や数値の扱い』を参照してください。STプログラムで使用するデータの型や表記方法を記載しています。STプログラムで制御構文を使用したい場合は，『第4章 STプログラムの式』を参照してください。STプログラムで使用する制御構文の書式や記述例を記載しています。

GX Works2を使用する場合は．．．

『はじめようGX Works2 （構造化プロジェクト編）』を参照してください。STプログラムをGX Works2にて作成し，シーケンサCPUユニットで動作を確認する手順などを，記載しています。

本マニュアルで使用する記号と内容について説明します。

記　号	内　　容	例
Point	その項目に関連する知識として知っておきたい内容を記載しています。	
備考	その項目に関連する知識として知っておくと便利な内容を記載しています。	備　考
[　　]	メニューバーのメニュー名	[プロジェクト]

A - 14

本マニュアルで使用する総称・略称

本マニュアルでは，下記の用語を使用しています。

総称／略称	内容／対象ユニット
GX Developer GX Works2	シーケンサの設定，プログラミング，デバッグ，保守までを行うためのプログラミングツールです。
QCPU(Qモード)	ベーシックモデルQCPU，ハイパフォーマンスモデルQCPU，プロセスCPU，二重化CPU，ユニバーサルモデルQCPUの総称です。
ST	ストラクチャードテキストの略称です。
FB	ファンクションブロックの略称です。

A - 15

1 概　　要

1

1.1 ST言語とは

ST言語は，オープン・コントローラでのロジックの記述方式について規定した国際規格IEC61131-3で定義されている言語です。

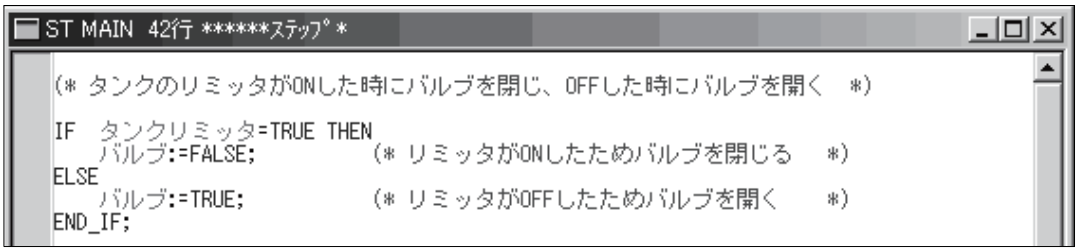
ST言語では演算子・制御構文・関数をサポートし，以下のような記述ができます。

- 条件文による選択分岐，反復文による繰り返しなどの制御構文
- 演算子（*，/，+，-，<，>，= など）を使用した式
- ユーザが定義したファンクションブロック(FB)の呼出し
- 関数の呼出し（MELSEC関数・IEC関数）
- 漢字などの全角文字を含むコメント記述

ST言語の主な特長は，以下の通りです。

(1) テキスト形式での自由な記述

ST言語は，半角英数字のテキスト形式で記述されます。コメント内や文字列内では，漢字などの全角文字も使用することができます。



(2) C言語などの高級言語と同等のプログラミングが可能

ST言語は，C言語などの高級言語と同様に**条件文による選択分岐や，反復文による繰り返しなど**の構文による制御が記述できます。このため，見やすいプログラムを簡潔に書くことができます。

```
(* ラインA～Cで制御を行う *)
CASE ライン OF
    1:開始スイッチ:=TRUE;   (* コンベア稼動          *)
    2:開始スイッチ:=FALSE;  (* コンベア停止          *)
    3:開始スイッチ:=TRUE;   (* コンベア停止  警告    *)
      警告ランプ:=TRUE;
END_CASE;

IF 開始スイッチ=TRUE THEN   (* コンベア稼動  処理100回  *)

    FOR 処理回数:=0
        TO 100
        BY 1 DO
        処理数:=処理数+1;
    END_FOR;
```


1 - 1

(3) 演算処理が容易に記述可能

ST言語はリストやラダーでは記述の難しい演算処理が簡潔で見やすく記述できるため，プログラムの可視性が良く，複雑な算術演算・比較演算などを行う分野に適しています。

```
CASE ライン OF
    1: Speed_A:=Distance_B / Hour_C * 3600 ;    (* 計測 *)
       (* FB呼出し *)
       FB1(I_Test:=D0,O_Test:=D1,IO_Test:=D100);

    2: M0:=GT_E(X0,D0,D1,D2,D3,Result);
       (* 実行条件X0がONすると、D0からD3のデータが右側のデータより大きい順に   *)
       (* 並んでいるかどうかを調べて、結果をResultに格納する。                 *)

           バルブ:=FALSE;
           RETURN;
END_CASE;
```


1 - 2

1.2 MELSEC-Q/LシリーズでのSTプログラムの特長

STプログラムは，ST言語で記述されたプログラムです。

STプログラミングを行う時にGX Developerを使用することにより，優れた操作環境で効率的なプログラミングを行うことが可能です。MELSEC-Q/LシリーズでのSTプログラムの主な特長は，以下の通りです。

(1) 部品化による設計の効率化を図ることが可能

ST言語ではよく使う処理をファンクションブロック(FB)として部品化してあらかじめ定義しておき，各プログラムの必要な部分で呼び出すことができます。これによりプログラム開発を効率化するとともに，プログラムミスを削減しプログラムの品質を向上することができます。

詳細は，関連マニュアルに記載している『GX Developer Version8 オペレーティングマニュアル（ファンクションブロック編）』を参照してください。

(2) シーケンサから読み出してSTプログラムを復元することが可能

MELSEC-Q/LシリーズにおけるSTプログラムでは，作成したプログラムをシーケンサに書き込み実行しますが，シーケンサから読み出した後に復元してST言語形式で編集することができます。

(3) システム稼動中にプログラムの変更（RUN中書込み）が可能

システムを停止させることなく実行中のプログラムの一部を変更することができます。

(4) 他言語プログラムとの連携

MELSEC-Q/LシリーズはST言語以外の言語もサポートしているため，処理に適した言語を使用して効率的なプログラムを作成することができます。

各プログラムはファイル単位で実行条件を設定でき，1つのCPUに複数のプログラムファイルを書き込むことができます。

複数の言語をサポートすることにより，最適制御で幅広い用途に対応しています。

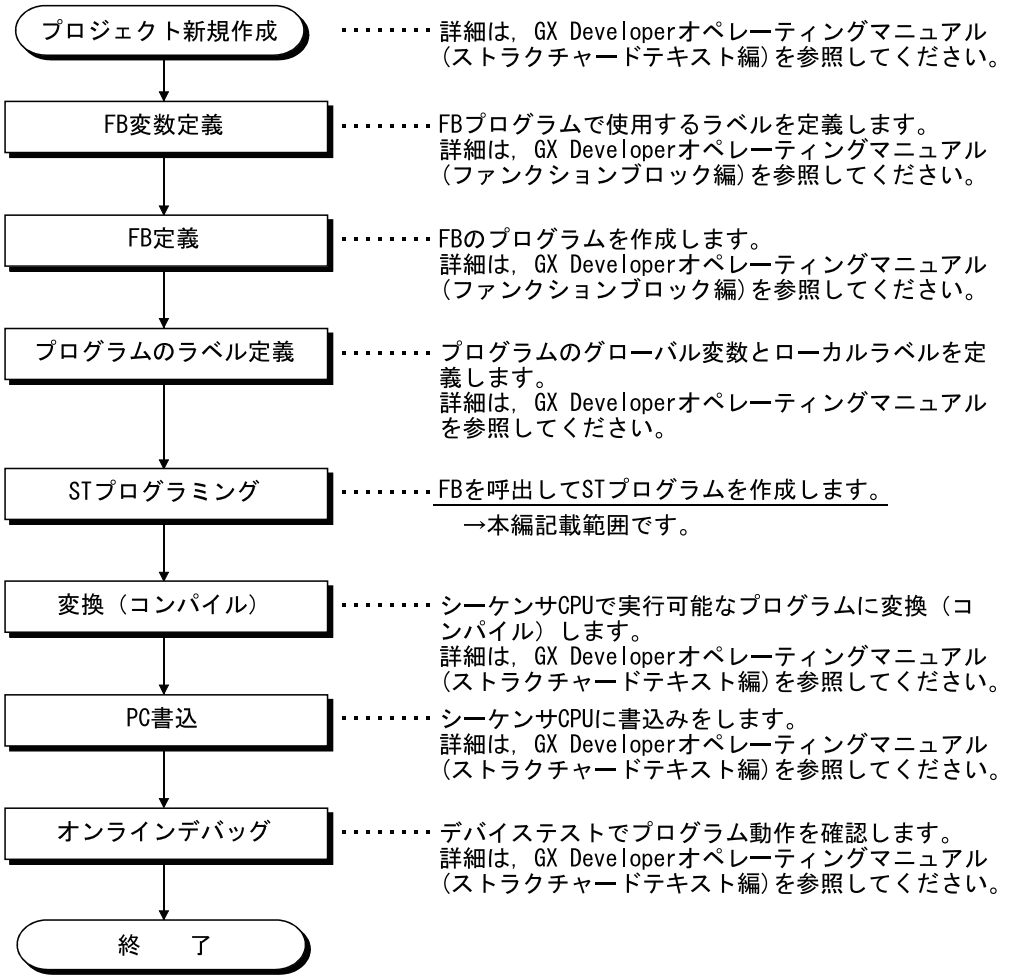
(5) 豊富な関数群

MELSEC-Q/LシリーズにおけるSTプログラムでは，MELSEC-Q/Lシリーズ用の各種共通命令に対応するMELSEC関数，IEC61131-3で定義されているIEC関数が用意されています。

1.3 STプログラム作成手順

STプログラミングの概略手順をフローチャートで示します。

以下の例は，ファンクションブロックで部品を作成し，その後，メインプログラムをST言語で作成した例です。



２　システム構成

2.1 システム構成

本節ではSTプログラムを使用する場合のシステム構成について説明します。

2.1.1 適用CPU

STプログラムは，下記CPUユニットで対応しています。

ベーシックモデルQCPU	ハイパフォーマンスモデルQCPU	ユニバーサルモデルQCPU	プロセスCPU	二重化CPU	LCPU
Q00JCPU Q00CPU Q01CPU	Q02CPU Q02HCPU Q06HCPU Q12HCPU Q25HCPU	Q00UJCPU Q00UCPU Q01UCPU Q02UCPU Q03UDCPU Q03UDECPU Q04UDHCPU Q04UDEHCPU Q06UDHCPU Q06UDEHCPU Q10UDHCPU Q10UDEHCPU Q13UDHCPU Q13UDEHCPU Q20UDHCPU Q20UDEHCPU Q26UDHCPU Q26UDEHCPU	Q02PHCPU Q06PHCPU Q12PHCPU Q25PHCPU	Q12PRHCPU Q25PRHCPU	L02CPU L26CPU-BT

2.1.2 STプログラム用プログラミングツール

STプログラムの作成・編集・モニタは，下記プログラミングツールで行います。

ソフトウェアパッケージ名	動作環境
GX Developer Version8.00A以降	『GX Developer Version8　オペレーティングマニュアル（スタートアップ編）』を参照してください。

2.1.3 STプログラム仕様

STの仕様および使用可能デバイスを説明します。

(1) プログラムサイズ

1プログラムあたりのファイルサイズは839680（半角）です。

Point

● ファイル内の文字数を数える場合は以下にご注意ください。

・キャリッジリターン(CR)，ラインフィード(LF)は2文字として扱います。
・全角文字は2文字として扱います。
・半角スペースは1文字として扱います。
・TABコードは1文字として扱います。

2 - 1

(2) 使用可能デバイス

STプログラムで使用できるデバイス名は以下の通りです。デバイス点数は，パラメータ設定で変更することができます。

デバイスの表記方法の詳細は，『3.3.3　デバイス』を参照してください。

分類	種別	デバイス	表現
内部ユーザデバイス	ビット	入力	X
		出力	Y
		内部リレー	M
		ラッチリレー	L
		アナンシェータ	F
		リンクリレー	B
		リンク特殊リレー	SB
	ワード	データレジスタ	D
		リンクレジスタ	W
		リンク特殊レジスタ	SW
内部システムデバイス	ビット	特殊リレー	SM
	ワード	特殊レジスタ	SD
リンクダイレクトデバイス	ビット	リンク入力	Jn¥X
		リンク出力	Jn¥Y
		リンクリレー	Jn¥B
		リンク特殊リレー	Jn¥SB
	ワード	リンクレジスタ	Jn¥W
		リンク特殊レジスタ	Jn¥SW
ユニットアクセスデバイス	ワード	インテリジェント機能ユニットデバイス	Un¥G
インデックスレジスタ	ワード	インデックスレジスタ	Z*1
ファイルレジスタ	ワード	ファイルレジスタ	R
			ZR
定数	ビット／ワード／ダブルワード	10進定数	K
		16進定数	H
	実数	実数定数	E
	文字列	文字列定数	"ABC"など
その他	ビット	SFCブロックデバイス	BL
	ビット	SFC移行デバイス	BL¥TR
	ビット	SFCステップリレー	BL¥S
	ビット	ダイレクト入力	DX
	ビット	ダイレクト出力	DY

*1：　Z0,Z1は使用不可です。ユニバーサルモデルQCPU/LCPUの場合，Z16～Z19が使用不可です。

2 - 2

(3)STプログラムでのみ使用可能なデバイス

STプログラムではタイマ・カウンタの接点・コイル・現在値を，個別のデバイスとして表記し，使用します。

タイマ・カウンタの接点・コイル・現在値のデバイス表現と種別は以下の通りです。

分類	種別	デバイス	デバイス表現
内部ユーザデバイス	ビット	タイマ接点	TS
		タイマコイル	TC
		積算タイマ接点	STS
		積算タイマコイル	STC
		カウンタ接点	CS
		カウンタコイル	CC
	ワード	タイマ現在値	TN/T
		積算タイマ現在値	STN/ST
		カウンタ現在値	CN/C

使用例

(1) 【STプログラム】
```
M0:=TS0;
```
【等価リストプログラム】
```
LD    T0
OUT   M0
```

(2) 【STプログラム】
```
COUNTER_M(X0, CC20, 10);
```
【等価リストプログラム】
```
LD    X0
OUT   C20    K10
```

Point

使用できる命令の詳細は，下記マニュアルを参照ください。
・MELSEC-Q/L プログラミングマニュアル（共通命令編）

2 - 3

３　ＳＴプログラムの文字や数値の扱い

3.1　使用できる文字

ST言語は，テキスト形式で記述するプログラミング言語です。
一般のテキストエディタでの文書編集と同様に記述することができますが，文法や使用できる文字や記号は定義されています。

(1) 使用できる文字

STプログラムで使用できる文字は以下の通りです。

文字種別	使用箇所				文字例
	プログラム文	コメント	文字列	ラベル*1	
英数字	○	○	○	○	ABC，IF，D0
半角記号 + − ＊ ／ = ＜ ＞ [ ] ( ) . , _ : ; $ # ″ ’ { }	○	○	△*2	×	(D0 ＊ D1)
半角カナ	×	○	○	○	ｽｲｯﾁ
全角文字	×	○	○	○	かな・漢字
半角スペース	○	○	○	×	
全角スペース	×	○	○	×	
改行コード	○	○	×	×	
TABコード	○	○	×	×	

○：使用できる　×：使用できない　△：一部使用できない

*1：ラベルで使用できない文字については『付.1　ラベル・FB名で使用できない文字列』を参照してください。

*2：文字列中で半角ダブルコーテーション(″)を使用することはできません。
使用すると変換エラーとなります。

3 − 1

(2) 文字の種類

STプログラムで使用する文字は以下のように分類できます。

分類項目		内容	例
ラベル名		ユーザが任意に定義する文字列 ファンクションブロック名・配列名・構造体名などを含みます。	Switch_A
定数		プログラムに直接書き込む値 （整数・実数・文字列など）	123,″abc″
コメント		プログラム中で制御の処理対象にならない注釈文	(* ONする *)
予約語	データ型名	データの種類を表す語	BOOL,DWORD
	制御構文	制御構文として使用するために文法上意味が定義されている語	IF,CASE,WHILE,RETURN
	デバイス名	MELSECでのシーケンサ用データ名	X,Y,M,ZR
	関数名	定義済みのMELSEC関数･IEC関数名	OUT_M REAL_TO_STR_E
演算子		式や代入文のために意味が定義されている文字コード	+ - < > =
区切り記号		プログラムの構造を明確にするために意味が定義されている文字コード	; ( )
その他の記号		レイアウトを整えるためのコード	半角スペース 改行コード, TAB

3 - 2

3.2 データの扱い

STプログラムでは使用するデータの型が定義されています。
3.2節，3.3節では，STプログラムにおけるデータ型とその表記方法を示します。

3.2.1 データ型

STプログラムで使用できるデータ型は以下の通りです。

データ型	内容	範囲	ラダーでの型	C言語での型
BOOL	ブール型	TRUE・FALSE,1・0[*1]	ビット	bool
INT	整数型	-32768～32767	ワード	signed short
DINT	倍精度整数型	-2147483648～2147483647	ダブルワード	signed long
REAL	実数型	-3.402823^{+38}～-1.175495^{-38}, 0.0, $+1.175495^{-38}$～$+3.402823^{+38}$	実数	float
STRING	文字列型	最大50文字までの定義が可能	文字列	char
ARRAY	配列データ型	指定されている要素のデータ型による。	配列	char[ ]等
STRUCT	構造化データ型	指定されている要素のデータ型による。	構造体	struct

*1： K,H指定のK0・K1・H0・H1についてはBOOL型として扱うことはできません。

Point

● **演算の結果がデータ型の範囲を超える場合の注意事項**
演算の結果がデータ型の範囲を超える場合，正しい結果が得られません。

3 - 3

3.2.2 ANY型について

関数の引数・戻り値などで複数のデータ型が許されている場合には，ANY型を使用します。ANY型は任意のデータ型を扱うデータ型で下記の表に示す種類があります。

例えば，関数の引数がANY_NUMと定義されていた場合は，引数として，ワード型・ダブルワード型・実数型から任意のデータ型を指定できます。

【記述例】

```
REAL EXPT(REAL In1,ANY_NUM In2);        (* 関数EXPTの関数定義 *)
                    └▶ワード型・ダブルワード型・実数型が指定可能
```

・ワード型デバイスを指定した場合

```
RealLabel := EXPT(E1.0, D0);
```

・ダブルワード型ラベルを指定した場合

```
RealLabel := EXPT(E1.0, DWLabel);
```

・実数を指定した場合

```
RealLabel := EXPT(E1.0, E1.0);
```

ANY型の種類と対応するデータ型・デバイス型は以下の通りです。

ANY型名	データ型	BOOL	INT	DINT	REAL	STRING
	ラダーでの型	ビット	ワード	ダブルワード	実数	文字列
ANY		○	○	○	○	○
ANY_SIMPLE		○	○	○	○	○
ANY_BIT		○	△	□	－	－
ANY_NUM		－	○	○	○	－
ANY_REAL		－	－	－	○	－
ANY_INT		－	○	○	－	－
ANY16		－	○	－	－	－
ANY32		－	－	○	－	－

○：対応する型として指定できる

－：指定できない

△：デバイス・定数・桁指定は使用できる／ラベルは使用できない

□：定数・桁指定は使用できる

3 - 4

3.2.3　配列と構造体

STプログラムでは，データとして配列と構造体を使用することができます。
配列と構造体は，使用前に各要素をローカルラベルもしくはグローバルラベルにて定義することで，プログラム中でひとかたまりにして扱える構造を持ったデータです。

(1) 配列

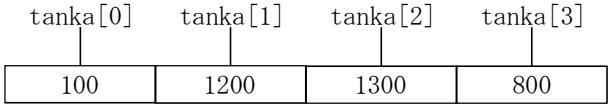
配列は，同じ型の複数のデータを組み合わせて定義したデータ型です。
STプログラム中での配列は，配列型で定義した変数（ラベル）名の後に，要素番号を[ ]で括って指定することで，各要素を個別に参照できます。
配列要素の指定番号は0から数えます。

【書　式】

配列名[配列要素の指定番号]

【イメージ図】

要素数4個のワード型配列を配列名tankaとした場合
配列要素の指定番号は0,1,2,3となります。



ワード型配列の場合，各配列要素にワードデータが入ります。

【記述例】

```
tanka[0] := 100;        (* 配列1要素目に100を代入 *)

(* デバイスD1を利用して配列2要素目に1200を代入 *)
D1 := 1;
tanka[D1] := 1200;
```

配列要素の指定番号としてデータ型INTによる指定ができます。

```
(* 配列3要素目にtanka[0]+tanka[1]を代入 *)
tanka[2] := tanka[0] + tanka[1];

pen1 := 3;
tanka[pen1] := 800;
```

配列要素の指定番号としてラベルを使用できます。

3 - 5

Point

● **配列要素の指定番号を使用する場合の注意事項**

要素数n個の配列の場合，配列要素の指定番号は0～n-1のため
n以上を指定すると変換時にエラーとなります。

例）要素数4個の配列の場合

```
tanka[4] := 100;          ←エラーとなります。
```

● **配列要素の指定番号に配列を使用する場合の注意事項**

配列要素の指定番号に配列を使用することができます。入れ子として最大5個まで使用することができます。17個以上使用すると変換エラーとなります。

例）入れ子が5個の場合

```
tanka[tanka[tanka[tanka[tanka[D1]]]]] := 100;
```

● **配列要素の指定番号を設定する場合の注意事項**

他のデバイスの情報を壊す可能性があるため,配列要素番号に指定した値が配列要素数を超えないよう注意してください。

● **配列の要素数を設定する場合の注意事項**

グローバル（ローカル）変数設定画面にて登録します。登録可能な要素数は,256個です。

3 - 6

(2) 構造体

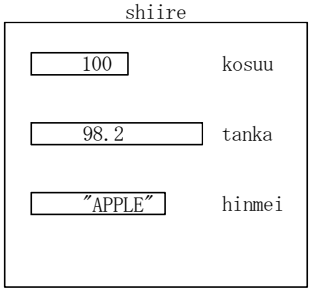
構造体は，任意の型のデータを組み合わせて定義したデータ型です。
構造体型で定義した変数（ラベル）名の後に，要素名をピリオド（.）で区切って記述することで，各要素を個別に参照することができます。
要素名をメンバ変数とも呼びます。

【書　式】

構造体名.構造体要素名

【イメージ図】

構造体名　shiire,
構造体要素：
ワード型1個　構造体要素名　kosuu
実数型1個　　構造体要素名　tanka
文字列型1個　構造体要素名　hinmei
とした場合



【記述例】

```
(* 構造体の要素kosuuに100を代入 *)
shiire.kosuu := 100;

(* 構造体の要素tankaに98.2を代入 *)
shiire.tanka := E98.2;

(* 構造体の要素hinmeiに"APPLE"を代入 *)
shiire.hinmei := "APPLE";
```

Point

● **構造体のメンバ変数を使用する場合の注意事項**
構造体変数設定画面にて登録可能なメンバ数は，128個です。

3 - 7

3.3　データの表記方法

STプログラムで使用できるデータには，定数・ラベル・デバイスがあります。

項目	内容	表記例
定数	プログラムに直接書き込まれる数値や文字列データ。プログラム実行中に変化しない。	123, "ABC"
ラベル	ユーザが型と名前を定義するデータ。	Switch_A
デバイス	QCPU（Qモード)/LCPUで使用されているデバイス。 デバイス名・デバイス番号で識別される。	X0,Y0,D100, J1¥X0

3.3.1　定数

STプログラムで各定数は，以下のように表記されます。

データ型	進数	表記方法	例
BOOL		TRUE・FALSE　1・0	M0 := TRUE;
	2進数	使用する2進数の前に“2#”をつける。	M0 := 2#0; M0 := 2#1;
	8進数	使用する8進数の前に“8#”をつける。	M0 := 8#0; M0 := 8#1;
	16進数	使用する16進数の前に“16#”をつける。	M0 := 16#0; M0 := 16#1;
INT DINT	2進数	使用する2進数の前に“2#”をつける。	D0 := 2#110;
	8進数	使用する8進数の前に“8#”をつける。	D0 := 8#377;
	10進数	使用する10進数を直接入力する。 (数値の前に“K”をつけても可)	D0 := 123; D0 := K123;
	16進数	使用する16進数の前に“16#”をつける。 (数値の前に“H”をつけても可)	D0 := 16#FF; D0 := HFF;
REAL		使用する実数を直接入力する。 (数値の前に“E”を付けても可)	ABC := 2.34; Rtest := E2.34;
STRING		文字列を''(もしくは"")で囲む。	Stest := 'ABC'; Stest := "ABC";

各定数で指定可能な範囲については，3.2.1項 データ型を参照ください。
3.2.1項 データ型に記載のない部分は，以下の範囲となります。

3 - 8

【K, H表記】

値の範囲	IECデータ型
K-32768 - K32767	INT,ANY16
K-2147483648 - K2147483647	DINT,ANY32
K0 - K32767	ANY_BIT（ワード）*1
K0 - K2147483647	ANY_BIT（ダブルワード）*2
H0 - HFFFF	INT,ANY16, ANY_BIT（ワード）*1
H0 - HFFFFFFFF	DINT,ANY32, ANY_BIT（ダブルワード）*2

【K, Hなし表記】

値の範囲	IECデータ型
0 － 1	BOOL
-32768 - 32767	INT
-2147483648 - 2147483647	DINT
0 - 4294967295	ANT_BIT（ダブルワード）*2
0 - 65535	ANT_BIT（ワード）*1
-32768 - 65535	ANY16
-2147483648 - 4294967295	ANY32
2#0 - 2#1 8#0 - 8#1 16#0 - 16#1	BOOL
2#0 - 2#1111_1111_1111_1111 8#0 - 8#177777 16#0 - 16#FFFF	INT ANY16 ANY_BIT（ワード）*1
2#0 - 2#1111_1111_1111_1111_1111_1111_1111_1111 8#0 - 8#37777777777 16#0 - 16#FFFFFFFF	DINT ANY_BIT（ダブルワード）*2 ANY32

*1：ワードデバイスとして扱う場合を示します。
　　＜例＞ D0 := NOT(K32767);
*2：ダブルワードデバイスとして扱う場合を示します。
　　＜例＞ K8M0 := NOT(K2147483647);

3 - 9

Point

● **ワードラベル，ワードデバイスの演算式でH,2#,8#,16#指定の数値を使用する場合の注意事項**

演算で扱う値がH8000～HFFFFの範囲である場合，STプログラムを変換した場合の演算結果とシーケンサCPUでデバイスに値を代入した場合の演算結果とで結果が異なります。

STプログラムを変換した場合の演算結果は，扱う値がワード型かダブルワード型かの判断がつかないため，符号無しとして演算しますが，シーケンサCPUでは符号付として演算します。

＜使用例＞

```
Data1 = -32768;
Data2 = 16#8000;
```

・ ST　　Result := Data1 / Data2;　→ -32768÷32768 = -1

・ CPU　Result := Data1 / Data2;　→ -32768÷-32768 = 1

● **文字列型データで“$”“’”を使用する場合の注意事項**

“$”をエスケープシーケンスとして使用します。

“$”に続く2つの16進数字は，ASCIIコードとして認識され，ASCIIコードに対応する文字が文字列に挿入されます。

“$”に続く2つの16進数字がASCIIコードに対応していない場合，変換エラーとなります。

ただし，“$”に続く文字が下記の場合はエラーとなりません。

表記	文字列中で使用する記号・プリンタコード
$$	$
$’	’
$Lまたは$l	ラインフィード
$Nまたは$n	改行
$Pまたは$p	ページ送り
$Rまたは$r	復帰
$Tまたは$t	TAB

例）Value := ″$’APPLE$’ $$100″;

● **2進数・8進数・10進数・16進数・実数表記する場合の注意事項**

2進数・8進数・10進数・16進数・実数表記では，見やすくするために“_（アンダースコア）”が使用できます。“_”は数値としては無視されます。

例）2#1101_1111　8#377_1　16#01FF_ABCD　22_323　1.0_1

（K,H,E指定時は“_”を使用することはできません。）

3 - 10

3.3.2　ラベル

STプログラムではデータをラベルを用いて使用することができます。
STプログラムでラベルを使用する場合は，使用する前にローカル変数設定画面もしくはグローバル変数設定画面でラベルの宣言を行う必要があります。
（ラベル，構造体ラベルの宣言方法は，『GX Developer Version8 オペレーティングマニュアル』を参照してください。）
STプログラムでのラベルの表記例は，以下の通りです。

```
例）Switch_A := FALSE;              (* Switch_AにFALSEを代入します。          *)

例）IF INT_TO_BOOL(Kosuu) = FALSE THEN
        Daisuu := 2147483647;
    END_IF;
                                    (* INT_TO_BOOL(Kosuu)がFALSEならば        *)
                                    (* Daisuuに2147483647を代入します。       *)

例）Limit_A := E1.0;                (* Limit_Aに1.0を代入します。             *)

例）コンベア[4] := Kosuu;           (* コンベアの5要素目にKosuuの値を         *)
                                    (* 代入します。                           *)

例）stPressure.Status := TRUE;      (* stPressureの要素名Statusに             *)
                                    (* TRUEを代入します。                     *)

例）stPressure.eLimit := E1.0;      (* stPressureの要素名eLimitに             *)
                                    (* 1.0を代入します。                      *)
```


3 - 11

参考

● ラベル宣言を行う手順

ラベルの宣言は，ローカル変数設定画面もしくはグローバル変数設定画面で行います。
ローカル変数設定画面は以下の操作で開くことができます。

GX Developer起動 →[プロジェクトを開く] →

→ローカルラベルアイコンをダブルクリック →ローカル変数設定画面

下記は，ローカル変数設定画面でラベルを設定した例です。

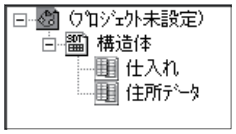
	Au	ﾗﾍﾞﾙ	定数値	ﾃﾞﾊﾞｲｽ種別	
1	○	Switch_A		ﾋﾞｯﾄ	
2	○	Kosuu		ﾜｰﾄﾞ	
3	○	Daisuu		ﾀﾞﾌﾞﾙﾜｰﾄﾞ	
4	○	Limit_A		実数	
5	○	ｺﾝﾍﾟｱ		ﾜｰﾄﾞ(20)	

構造体ラベルを宣言する場合

① 構造体の要素を宣言します。

GX Developer起動 →[プロジェクトを開く] →構造体タブをクリック →構造体新規追加

→



→“仕入れ”“住所データ”をダブルクリック

→ 構造体変数設定画面

下記は，構造体変数設定画面で構造体の要素ラベルを設定した例です。

	ﾗﾍﾞﾙ	ﾃﾞﾊﾞｲｽ種別
1	kosuu	ﾜｰﾄﾞ
2	tanka	実数

② 構造体ラベルを宣言します。

構造体ラベルの宣言は，ローカル変数設定画面もしくはグローバル変数設定画面で行います。
ローカル変数設定画面は以下の操作で開くことができます。

→ローカルラベルをダブルクリック →ローカル変数設定画面

下記は，ローカル変数設定画面で構造体ラベルを設定した例です。

	Au	ﾗﾍﾞﾙ	定数値	ﾃﾞﾊﾞｲｽ種別
1	○	仕入れデータ	詳細設定	構造体(仕入れ)

3 - 12

Point

● **ポインタ型・タイマ型・カウンタ型・積算タイマ型ラベル使用時の注意事項**
ポインタ型・タイマ型・カウンタ型・積算タイマ型のラベルは宣言できますが，STプログラム中でラベルとして使用した場合，変換エラーとなり使用できません。

● **タイマ型・カウンタ型・積算タイマ型ラベル使用時の注意事項**
構造体のメンバ変数において，タイマ型・カウンタ型・積算タイマ型のラベルを定義した場合，そのメンバ変数はST編集画面では使用できません。ただし，タイマ型・カウンタ型・積算タイマ型のラベルを含む構造体の他のメンバ変数は使用することができます。

3 - 13

3.3.3　デバイス

(1) デバイスの使用方法

STプログラム上でラベルを使用しないで，QCPU（Qモード)/LCPUのデバイスを直接記述して使用することができます。デバイスは式の左辺・右辺，関数の引数・戻り値などに使用することができます。

【記述例】

```
M0 := TRUE;                          (* M0をONします。                    *)

IF INT_TO_BOOL(D0) = FALSE THEN      (* INT_TO_BOOL(D0)がFALSEならば     *)
      W0 := 1000;                    (* W0に1000を代入します。            *)
END_IF;
```

備考

● **デバイスの指定を行う場合には．．．**

デバイスの指定は大文字，小文字共に可能です。

● **使用可能デバイスは？**

使用可能デバイスについては，本書『2.1.3 (2) 使用可能デバイス』を参照してください。

(2) その他の使用方法

デバイスの修飾方法・指定方法として以下の３方法を使用することができます。これらはラダープログラムで使用する場合と同じ使用方法で使用することができます。以下にSTプログラムで使用する場合の記述例と説明を記載します。（各使用方法の詳細については，『MELSEC-Q/L プログラミングマニュアル(共通命令編)』を参照してください。）

(a) **インデックス修飾**

(b) **ビット指定**

(c) **桁指定**

3 - 14

(a) インデックス修飾

インデックス修飾は，インデックスレジスタを使用した間接アドレス指定です。

インデックスレジスタを使用すると，デバイス番号は（直接指定しているデバイス番号）+（インデックスレジスタの内容）になります。

【記述例】

```
(* Z2に入っている数値により対象となるDデバイス番号を変更します。    *)
(* Z2に1が入っている場合は対象となるデバイス番号がD(0+1)→D1となります。 *)
  Z2 := 1;          (* インデックスレジスタZ2に1を代入   *)
  D0Z2 := K0;       (* D0Z2にK0を代入                    *)
    └▶D1
```

(b) ビット指定

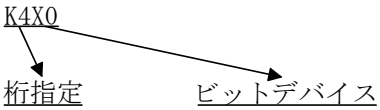
ワードデバイスの各ビットNo.を指定することにより，ビットデバイスとして使用することができます。

【記述例】

```
D0.0 = TRUE;      (* D0デバイスの0ビット目をONにします。   *)
W0.F = FALSE;     (* W0デバイスの15ビット目をOFFにします。*)
```

(c) 桁指定

ビットデバイスの4ビット，8ビット，12ビット．．．を1桁として桁指定することにより，ビットデバイスでワードデータ，もしくはダブルワードデータを扱うことができます。



【記述例】

```
K4X0 := D0;       (* X0デバイスから16ビット分を整数型(INT)として
                                        使用し，D0を代入します。*)

Wtest := K1X0;    (* X0デバイスから4ビット分をワード型ラベルWtestに
                                        代入します。*)

Dwtest := K5X0;   (* X0デバイスから20ビット分をダブルワード型ラベル
                                        Dwtestに代入します。*)
```


3 - 15

備考

● **桁指定を使用する場合のデータ型は．．．**

桁指定を使用する場合は，以下データ型となります。

例）X0を使用した場合

整数型(INT)　　：K1X0,K2X0, K3X0,K4X0

倍精度整数型(DINT)　　：K5X0,K6X0, K7X0,K8X0

Point

● **桁指定を使用する場合の注意事項1**

右辺と左辺のデータ型が異なる場合，変換エラーとなります。

例）D0 := K5X0;

K5X0がダブルワード,D0がワード型となるため上記プログラムはエラーとなります。

● **桁指定を使用する場合の注意事項2**

右辺>左辺の場合，左辺の対象点数範囲内にデータ転送します。

（桁指定の対象点数については，『MELSEC-Q/L プログラミングマニュアル（共通命令編）』を参照してください）

例）K5X0 := 2#1011_1101_1111_0111_0011_0001;

K5X0 : 対象点数20点

K5X0には1101_1111_0111_0011_0001(20桁)を代入します。

3 - 16

４　ＳＴプログラムの式

4.1　代入文

代入文は右辺の式の結果を左辺のラベルやデバイスに代入する機能を持ちます。
代入文では，右辺の式の結果と左辺のデータ型が同じである必要があります。異なる場合は変換エラーとなります。

【記述例】

・実デバイスを使用した場合

```
D0 := 0;
```

この式を実行した時に10進数の0をD0に代入します。

・ラベルを使用した場合

Stestという文字列型ラベルを使用した場合

```
Stest := "APPLE";
```

この式を実行した時に文字列"APPLE"をStestに代入します。

Point

● **文字列を代入する場合の注意事項**

文字列の代入は最大文字列長32文字です。文字列長32文字を超える文字列を代入した場合は変換エラーとなります。

● **代入文の左辺にデバイスを使用する場合の注意事項**

TS・TC・STS・STC・CS・CC・BL・DX・BL¥S・BL¥TRデバイスは代入文の左辺には使用できません。上記デバイスを左辺に使用した場合は変換エラーとなります。

4 - 1

4.2　演算子

本節では，STプログラムにて使用できる演算子の一覧とその使用例を示します。

4.2.1　演算子一覧

STプログラムで使用する演算子一覧と演算実行時の優先順位を以下に示します。

演算子	内　　容	優先順位
( )	丸括弧式	最上位
関数( )	関数のパラメータリスト	↑
**	指数(べき乗)　　tei**shisuu	↑
NOT	ブール補数 (ビットを反転した値)	↑
* / MOD	乗算 除算 剰余	↑
+ −	加算 減算	↑
<,>,<=,>=	比較	↑
= <>	等式 不等式	↑
AND,&	論理積	↑
XOR	排他的論理和	↑
OR	論理和	最下位

優先順位が同一の場合は，左側の演算子から評価されます。

演算子と対象データ型および演算結果データ型について以下の表にまとめます。

演算子	対象データ型	演算結果データ型
*, /, +, −	ANY_NUM	ANY_NUM
< , >, <=, >=,=, <>	ANY_SIMPLE	BOOL
MOD	ANY_INT	ANY_INT
AND, &, XOR, OR, NOT	ANY_BIT*1	ANY_BIT*1
**	ANY_REAL（底） ANY_NUM（指数）	ANY_REAL

*1： ラベル，定数（負の範囲）を除きます。

4 − 2

Point

● **演算子を使用する場合の注意事項1**
演算子の右辺と左辺のデータ型が異なる場合，変換エラーとなります。

● **演算子を使用する場合の注意事項2**
演算子は半角で記述してください。

● **演算子を使用する場合の注意事項3**
1式に記述できる演算子の使用個数は，最大1024個です。1025以上使用すると，変換エラーとなります。

備考

● **ANY型の説明は．．．**
ANY型の説明は，『3.2.2 ANY型について』を参照してください。

4 - 3

4.2.2　演算子の使用例

以下にSTプログラムでの演算子の使用例を記載します。

(1) 整数型(INT)の演算

(a) 実デバイスを使用した場合

＜使用例＞

```
D0 := D1 * (D2 + K3) / K100;
```

≪演算順≫

① D2＋K3
② (D2＋K3)*D1
③ (D2＋K3)*D1/K100
④ ③の結果をD0に代入

(b) ラベルを使用した場合

・ワード型ラベルDtest1, Dtest2を使用した場合

＜使用例＞

```
Dtest2 := Dtest1 MOD (D2 + K3) * K100;
```

≪演算順≫

① D2 + K3
② Dtest1 MOD (D2 + K3)
③ Dtest1 MOD (D2 + K3) * K100
④ ③の結果をDtest2に代入

・ダブルワード型ラベルDwtest1, Dwtest2を使用した場合

＜使用例＞

```
Dwtest2 := Dwtest1 - Dwtest1 / K100;
```

≪演算順≫

① Dwtest1/K100
② Dwtest1-Dwtest1/K100
③ ②の結果をDwtest2に代入

Point

● **演算の結果がデータ型の範囲を超える場合の注意事項**
演算の結果がデータ型の範囲を超える場合，正しい結果が得られません。
データ型の範囲については、3.2.1項を参照してください。

4 - 4

(2) ブール型(BOOL)の演算

(a) 実デバイスを使用した場合

＜使用例＞

```
M0 := X0 AND X1 AND (D1 = 100);
```

≪演算順≫

① X0 AND X1の結果がONでDに100の場合のみM0はONする

(b) ラベルを使用した場合

・ビット型ラベルBtest1, Btest2を使用した場合

＜使用例＞

```
Btest2 := Btest2 OR Btest1;
```

≪演算順≫

① Btest2またはBtest1がONの場合にBtest2はONする

4 - 5

4.3　制御構文

STプログラムでは比較・繰り返しを行うために条件文と反復文が用意されています。

条件文：ある一定の条件が成立したときに選択した文を実行します。
反復文：ある特定の変数や条件の状態に応じて，1つ以上の文を何度も繰り返し実行します。

4.3.1　制御構文一覧

下記に制御構文一覧を示します。

条件文	IF条件文
	CASE条件文
反復文	FOR・・・DO構文
	WHILE・・・DO構文
	REPEAT・・・UNTIL構文
その他の制御構文	RETURN構文
	EXIT構文

● **制御構文に階層を使用する場合の注意事項**
制御構文の階層は16段まで可能です。17段以上使用しても変換エラーとはなりませんが階層が深くなるとわかりにくいプログラムになる恐れがありますので，階層は深くても4～5段になるようプログラムすることをお奨めします。

4 - 6

4.3.2　条件文

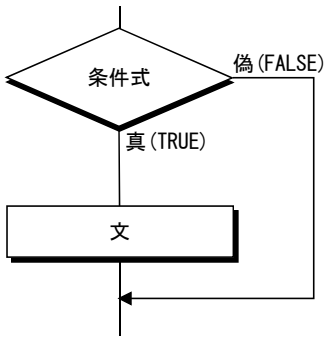
(1) IF　THEN　条件文

【書　式】

```
IF ＜ブール式＞ THEN
      ＜文 ・・・＞
END_IF;
```

【説　明】

ブール式(条件式)が真(TRUE)の時に，文を実行します。ブール式が偽(FALSE)の場合，文は実行されません。
ブール式は，単一のビット型変数の状態，あるいは数多くの変数を含んだ複雑な式のブール演算の結果として，真(TRUE)か偽(FALSE)を返す式であればどのような式でも使用できます。



【記述例】

(a) ブール式に実デバイスを使用する場合

```
IF X0 THEN              (* X0がONであればD0に0を代入します。    *)
     D0 := 0;           (* X0の部分がX0= TRUEでも同じ意味と     *)
                        (* なります。                           *)
END_IF;
```

(b) ブール式に演算子を使用する場合

```
IF (D0*D1) <= 200 THEN      (* D0*D1が200以下であれば          *)
     D0 := 0;               (* D0に0を代入します。             *)
END_IF;
```

(c) ブール式にラベルを使用する場合

① ラベルw_Real1を実数型として指定した場合

```
IF w_Real1 > 2.0 THEN       (* w_Real1が2.0より大きければ      *)
     D0 := 0;               (* D0に0を代入します。             *)
END_IF;
```


4 - 7

② ラベルw_Strを文字列型として指定した場合

```
IF w_Str = "ABC" THEN          (* w_Strが"ABC"であれば          *)
     D0 := 0;                  (* D0に0を代入します。          *)
END_IF;
```

③ ラベルw_Strを文字列型として指定した場合

```
IF w_Str = 'ABC' THEN          (* w_Strが'ABC'であれば          *)
     D0 := 0;                  (* D0に0を代入します。          *)
END_IF;
```

(d) ブール式にファンクションブロックを使用する場合

ファンクションブロック名w_FBがローカル変数設定に設定されており，ファンクションブロックの出力変数としてワード型ラベルw_Outが設定されている場合

ファンクションブロックを実行後

（ファンクションブロックの使用方法については『GX Developer Version8 オペレーティングマニュアル』を参照してください。）

```
IF w_FB.w_Out = 100 THEN       (* w_Outが100であれば           *)
     D0 := 0;                  (* D0に0を代入します。          *)
END_IF;
```

(e) ブール式に関数を使用する場合

```
IF INT_TO_BOOL( D0 ) = FALSE THEN
     D0 := 0;                  (* INT_TO_BOOL(D0)がFALSEであれば *)
END_IF;                        (* D0に0を代入します。          *)
```

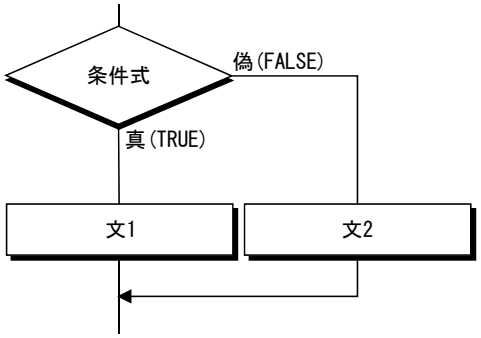
(2) IF　...ELSE　条件文

【書　式】

```
IF ＜ブール式＞ THEN
      ＜文1 ・・・＞
ELSE
      ＜文2 ・・・＞
END_IF;
```

【説　明】

ブール式（条件式）が真（TRUE）の時に，文1を実行します。
ブール式の値が偽（FALSE）の場合は文2を実行します。



【記述例】

(a) ブール式に実デバイスを使用する場合

```
IF X0 THEN                (* X0の部分がX0= TRUEでも同じ意味と      *)
                          (* なります。                            *)
     D0 := 0;             (* X0がONであればD0に0を代入します。     *)
ELSE                      (* X0がONでなければD0に1を代入します。   *)
     D0 := 1;
END_IF;
```

(b) ブール式に演算子を使用する場合

```
IF (D0*D1) <= 200 THEN    (* D0*D1が200以下であれば                *)
     D0 := 0;             (* D0に0を代入します。                   *)
ELSE                      (* D0*D1が200以下でなければ              *)
     D0 := 1;             (* D0に1を代入します。                   *)
END_IF;
```

(c) ブール式に関数を使用する場合

```
IF INT_TO_BOOL(D0) = FALSE THEN  (* INT_TO_BOOL(D0)がFALSEであれば *)
     D0 := 0;                    (* D0に0を代入します。             *)
ELSE                             (* INT_TO_BOOL(D0)がFALSEでなけれ *)
     D0 := 1;                    (* ばD0に1を代入します。           *)
END_IF;
```


4 - 9

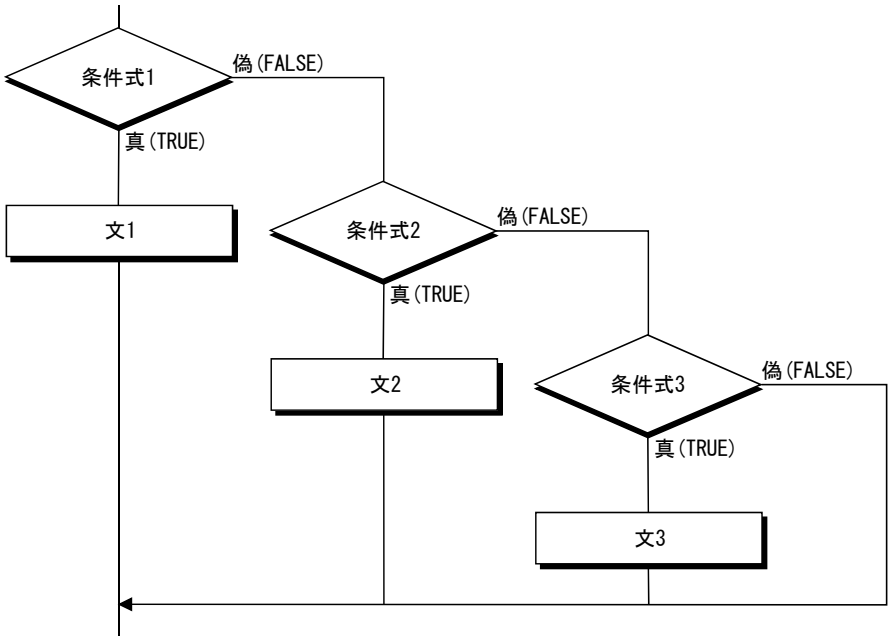
(3) IF　...ELSIF　条件文

【書　式】

```
IF <ブール式1 > THEN
      <文1 ・・・>
ELSIF <ブール式2 > THEN
      <文2 ・・・>
ELSIF <ブール式3 > THEN
      <文3 ・・・>
END_IF;
```

【説　明】

ブール式（条件式）1が真（TRUE）の時に，文1を実行します。ブール式1の値が偽（FALSE）でブール式2の値が真（TRUE）の場合は文2を実行します。
ブール式2の値が偽（FALSE）でブール式3の値が真（TRUE）の場合は文3を実行します。



4 - 10

【記述例】

(a) ブール式に実デバイスを使用する場合

```
IF D0 < 100 THEN                (* D0が100より小さいならば      *)
    D1 := 0;                    (* D1に0を代入します。          *)
ELSIF D0 <= 200 THEN            (* D0が200以下であれば          *)
    D1 := 1;                    (* D1に1を代入します。          *)
ELSIF D0 <= 300 THEN            (* D0が300以下であれば          *)
    D1 := 2;                    (* D1に2を代入します。          *)
END_IF;
```

(b) ブール式に演算子を使用する場合

```
IF (D0*D1) < 100 THEN           (* D0*D1が100より小さいならば   *)
    D1 := 0;                    (* D1に0を代入します。          *)
ELSIF (D0*D1) <= 200 THEN       (* D0*D1が200以下であれば       *)
    D1 := 1;                    (* D1に1を代入します。          *)
ELSIF (D0*D1) <= 300 THEN       (* D0*D1が300以下であれば       *)
    D1 := 2;                    (* D1に2を代入します。          *)
END_IF;
```

(c) ブール式に関数を使用する場合

```
IF INT_TO_BOOL(D0) = TRUE THEN      (* INT_TO_BOOL(D0)がTRUE *)
                                    (* ならば                *)
    D1 := 0;                        (* D1に0を代入します。   *)
ELSIF INT_TO_BOOL(D2) = TRUE THEN   (*INT_TO_BOOL(D2)がTRUE  *)
                                    (* ならば                *)
    D1 := 1;                        (* D1に1を代入します。   *)
END_IF;
```


4 - 11

(4) CASE　条件文

【書　式】

```
CASE ＜整数式＞ OF
        ＜整数選択値1＞：＜文1 ・・・＞
        ＜整数選択値2＞：＜文2 ・・・＞
                ：
                ：
        ＜整数選択値n＞：＜文n ・・・＞
ELSE
        ＜文n+1 ・・・＞
END_CASE;
```

● CASE条件文の＜整数選択値*＞で使用できる指定方法

CASE条件文中の＜整数選択値*＞については，以下のように1つ，複数もしくは範囲指定も可能です。

例)

```
1:          (* 整数式の値が，1の場合          *)
2,3,4:      (* 整数式の値が，2,3,4の場合     *)
5..10:      (* 整数式の値が，5から10の場合 *)
```

".."を用いて範囲を指定する場合，".."より前の値よりも".."より後の値を大きくしてください。

また複数および範囲指定を組み合わせて指定することも可能です。

```
1,2..5,9    (* 整数式の値が，1,2..5,9の場合 *)
```

● CASE条件文の＜整数式＞で使用できるデータ型

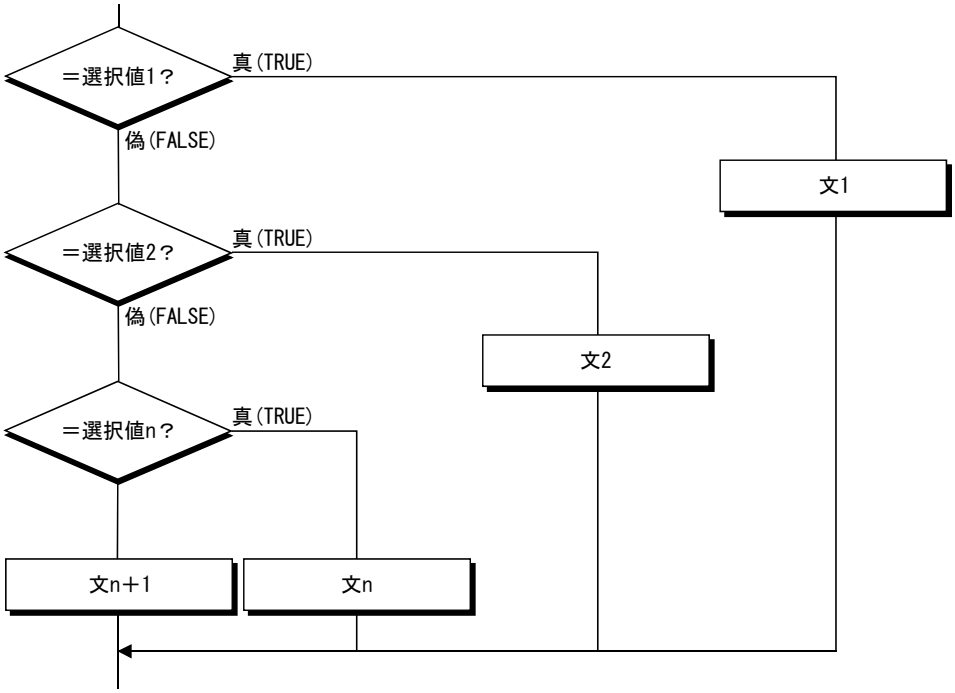
CASE条件文中の＜整数式＞として指定可能なデータ型は整数型(INT),倍精度整数型(DINT)です。ワードデバイス，ワード型・ダブルワード型ラベルを指定することができます。

【説　明】

CASE条件文の式の結果は整数値で返されます。この条件文は，例えば単一の整数値や複雑な式の結果の整数値によって，選択文を実行する場合に使用できます。

整数式の値と一致する整数の選択値を持った文がまず実行され，一致するものがない場合は，ELSEの後に続く文を実行します。

4 - 12



【記述例】

(a) 整数式に実デバイスを使用する場合

```
CASE D0 OF
   1:
      D1 := 0;       (* D0が1であればD1に0を代入します。          *)
   2,3:
      D1 := 1;       (* D0が2,3であればD1に1を代入します。        *)
   4..6:
      D1 := 2;       (* D0が4～6であればD1に2を代入します。       *)
   ELSE
      D1 := 3;       (* D0が上記以外であればD1に3を代入します。   *)
END_CASE;
```

(b) 整数式に演算結果を使用する場合

```
CASE D0*D1 OF
   1:
      D1 := 0;       (* D0*D1が1であればD1に0を代入します。       *)
   2,3:
      D1 := 1;       (* D0*D1が2,3であればD1に0を代入します。     *)
   4..6:
      D1 := 2;       (* D0*D1が4～6であればD1に2を代入します。    *)
   ELSE
      D1 := 3;       (* D0*D1が上記以外であればD1に3を代入します。*)
END_CASE;
```


4 - 13

(c) 整数式に関数を使用する場合

```
CASE DINT_TO_INT(dData) OF
   1:              (* DINT_TO_INT(dData)が1であれば          *)
      D1 := 0;     (* D1に0を代入します。                    *)
   2,3:            (* DINT_TO_INT(dData)が2,3であれば        *)
      D1 := 1;     (* D1に1を代入します。                    *)
   4..6:           (* DINT_TO_INT(dData)が4～6であれば       *)
      D1 := 2;     (* D1に2を代入します。                    *)
   ELSE            (* DINT_TO_INT(dData)が上記以外であれば   *)
      D1 := 3;     (* D1に3を代入します。                    *)
END_CASE;
```

Point

● **整数選択値を使用する場合の注意事項**

CASE条件文中に同一整数選択値が複数存在する場合は，上行に記述されている文を優先的に実行し，以降に記述されている同一整数選択値を持った文は実行しません。例えば，以下のCASE条件文において，D100の値が3である場合,整数選択値3を持つ文3が実行され，同一整数選択値を持つ文4は実行されません。

```
CASE D100 OF
   1:      ＜文1 ・・・＞
   2:      ＜文2 ・・・＞
   3:      ＜文3 ・・・＞
   3,4:    ＜文4 ・・・＞
ELSE
           ＜文5 ・・・＞
END_CASE;
```

＜整数選択値*＞の指定には，K指定無しの10進数のみ指定可能です。

4 - 14

4.3.3　反復文

(1) FOR...DO　構文

【書　式】

```
FOR ＜反復変数初期化＞
    TO ＜最終値の式＞
    BY ＜増加式＞ DO
    ＜文・・・＞
END_FOR;
```

反復変数初期化　　：反復変数として使用するデータの初期化を行います。
最終値の式・増加式：初期化された反復変数を増加式に従って加算もしくは減算し，最終値に達するまで繰り返し処理を行います。

● FOR構文の＜最終値の式・増加式＞で使用できるデータ型
整数値や演算式の結果の整数値を指定することができます。

【説　明】

反復変数として使用するデータの初期化を行います。
初期化された反復変数を増加式に従って加算もしくは減算し，最終値を超えるまで，DOからEND_FOR内の1つ以上の文を繰り返し実行します。
FOR...DO文終了後の反復変数は，終了時点での値を保持しています。

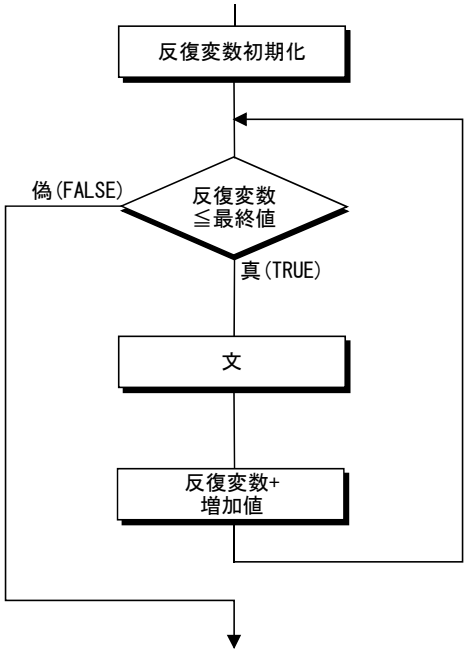
Point

● **反復変数を使用する場合の注意事項**
反復変数には，倍精度整数型(DINT)・整数型(INT)を使用することが可能ですが，構造体要素および配列要素を使用することはできません。
また，反復変数で使用した型と＜最終値の式＞・＜増加式＞の型を一致させてください。

● **増加式を使用する場合の注意事項**
＜増加式＞は省略可能です。省略した場合＜増加式＞は1として実行します。
また，＜増加式＞に”0”を代入すると，FOR構文以下が実行されなくなることや，無限ループとなることがあります。

● **FOR・・・DO構文を使用する場合の注意事項**
FOR・・・DO構文ではFOR構文中の＜文・・・＞の実行後に反復変数のカウント処理を行います。反復変数のデータ型の最大値を上回るまたは最小値を下回るカウント処理が実行された場合，無限ループが発生します。

4 - 15



【記述例】

(a) 反復変数に実デバイスを使用する場合

```
FOR W1 := 0              (* W1を0で初期化します。              *)
    TO 100               (* W1が100まで処理を繰り返します。    *)
    BY 1 DO              (* W1を1づつ増加させます。            *)
    W3 := W3 + 1;        (* 繰り返し処理の間W3を+1します。     *)
END_FOR;
```


4 - 16

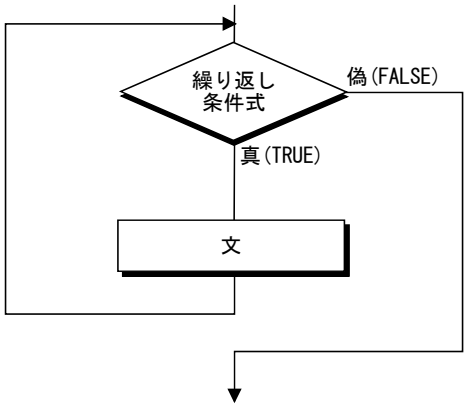
(2) WHILE...DO　構文

【書　式】

```
WHILE ＜ブール式＞ DO
      ＜文・・・＞
END_WHILE;
```

【説　明】

WHILE・・・DO構文はブール式（条件式）が真(TRUE)の間，1つ以上の文を実行します。
ブール式は，文の実行に先立って判定されブール式が偽(FALSE)の場合は，DO・・・END_WHILE内の文が実行されません。WHILE構文中の＜ブール式＞は，結果が真か偽かを返すものであれば良いため，IF条件文中の＜ブール式＞で指定可能な式はすべて使用することができます。



【記述例】

(a) ブール式に実デバイス・演算子を使用する場合

```
WHILE W100 < (W2-100) DO      (* W100<(W2-100)が真の間処理を    *)
                              (* 繰り返します。                 *)
      W100 := W100 + 1;       (* 繰返し処理の間W100を+1します。 *)
END_WHILE;
```

(b) ブール式に関数を使用する場合

```
WHILE BOOL_TO_DINT(M0) < BOOL_TO_DINT(M1) DO
      D4 := D4 + 1;           (* BOOL_TO_DINT(M0) <             *)
                              (* BOOL_TO_DINT(M1) が真の間      *)
                              (* 処理を繰り返します。           *)
END_WHILE;                    (* 繰り返し処理の間D4を+1します。 *)
```


4 - 17

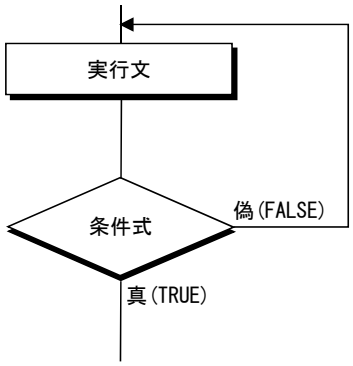
(3) REPEAT・・・UNTIL構文

【書　式】

```
REPEAT
      ＜文・・・＞
      UNTIL　＜ブール式＞
END_REPEAT;
```

【説　明】

REPEAT・・・UNTIL構文は，ブール式（条件式）が偽(FALSE)の間，1つ以上の文を実行します。
ブール式は，文の実行後に判定され，値が真(TRUE)の場合は，REPEAT・・・UNTIL内の文が実行されません。
REPEAT構文中の＜ブール式＞は，結果が真か偽かを返すものであれば良いため，IF条件文中の＜ブール式＞で指定可能な式はすべて使用することができます。



【記述例】

(a) ブール式に実デバイスを使用する場合

```
REPEAT
      D1 := D1 + 1;          (* D1が100より小さい値になるまで  *)
      UNTIL D1 < 100         (* D1を+1します。                 *)
END_REPEAT;
```

(b) ブール式に演算子を使用する場合

```
REPEAT
      W1 := W0*W1 - D0;      (* W0*W1が100より小さい値になる   *)
                             (* までW0*W1 - D0をW1に            *)
      UNTIL W0*W1 < 100      (* 代入します。                   *)
END_REPEAT;
```


4 - 18

(c) ブール式に関数を使用する場合

```
REPEAT
    D1 := D1 + 1;                    (* BOOL_TO_DINT(M0)が     *)
                                     (* 100より                *)
    UNTIL  BOOL_TO_DINT(M0) < 100    (* 小さい値になるまで     *)
END_REPEAT;                          (* D1を+1します。         *)
```

Point

- **反復文を使用する場合の注意事項1**
  反復文を使用する場合は，無限ループ処理にならないように注意してください。
- **反復文を使用する場合の注意事項2**
  反復文を多用するとシーケンサのスキャンタイムが著しく増加しますので，注意してください。

4 - 19

4.3.4　その他の制御構文

(1) RETURN　構文

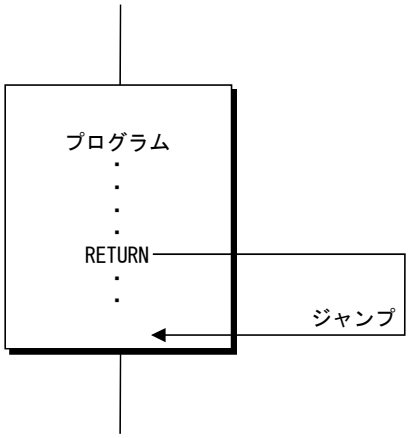
【書　式】

```
RETURN;
```

【説　明】

RETURN構文はファンクションブロック内のプログラム・STプログラムを途中で終了するために使用します。

RETURN構文をプログラムで使用すると，RETURN構文以降の処理がすべて無視され，RETURNが実行される場所からSTプログラム・ファンクションブロック内のプログラム最終行までジャンプを行います。



【記述例】

(a) IF条件文ブール式に実デバイスを使用する場合

```
IF  X0 THEN              (* X0がONであればIF内の文を実行します。 *)
    RETURN;              (* RETURN行以降のプログラムを無視します *)
END_IF;
```


4 - 20

(2) EXIT　構文

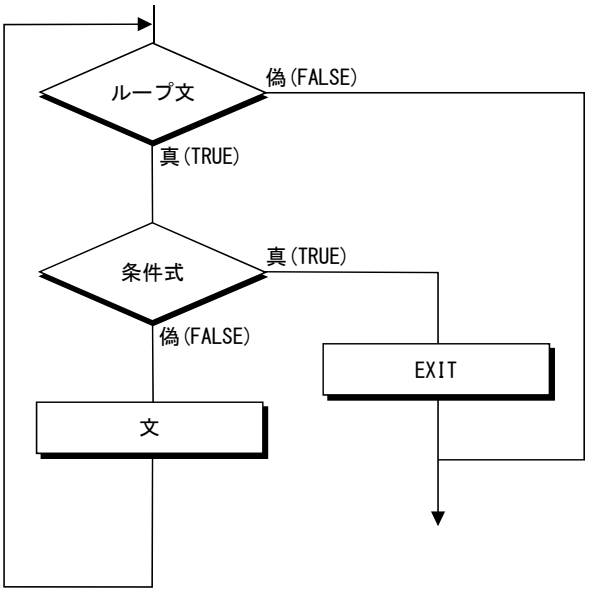
【書　式】

```
EXIT;
```

【説　明】

EXIT構文はSTプログラムの反復文の中でのみ使用可能な構文で，反復ループを途中で終了させます。

反復ループを実行中にEXIT構文に達すると，EXIT構文以降の反復ループ処理は実行されません。反復ループ処理を終了した次の行からプログラムを継続します。



【記述例】

(a) IF条件文ブール式に実デバイスを使用する場合

```
FOR D0 := 0 TO 10 DO      (* D0の値が10以下の場合に反復          *)
                          (* します。                            *)
  IF D1 > 10 THEN         (* D1の値が10より上かをチェック        *)
    EXIT;                 (* D1の値が10より上の場合は反復処理    *)
                          (* 終了                                *)
  END_IF;
END_FOR;
```


4 - 21

4.3.5　制御構文使用時の注意点

STプログラムで，制御構文を使用した場合の使用ステップ数・演算処理時間，注意点について説明します。

(1) 制御構文使用ステップ数と演算処理時間

制御構文を使用する時の使用ステップ数，演算処理時間について説明します。
演算処理時間は，各命令の処理時間の加算により算出しています。プログラム作成時の参考としてください。

(a) IF条件文

IF条件文 1　　単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	IF X0 THEN 　D0 := 100; END_IF;	7	1.534	10.9
リスト プログラム	LD X0 MOV K100 D0	3	0.134	0.90
【備　考】 条件文部分のみではSTを使用しない方が処理時間が早くなります。 ただしSTのIF条件文の比較対象はブール式のため複雑な比較が容易に可能です。				

IF条件文 2　　単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	IF D0 = 0 THEN 　D0 := 100; END_IF;	9	1.6	11.5
リスト プログラム	LD= D0 K0 MOV K100 D0	5	0.20	1.50
【備　考】 条件文部分のみではSTを使用しない方が処理時間が早くなります。 ただしSTのIF条件文の比較対象はブール式のため複雑な比較が容易に可能です。				

4 - 22

(b) CASE条件文

単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	CASE D0 OF 1,2: D0 := 100; 3..10: D1 := D1 + 1; END_CASE;	29	5.004	36.1
リスト プログラム	LD= D0 K1 AND= D0 K2 MOV K100 D0 LD>= D0 K3 AND<= D0 K10 INC D1	16	0.64	4.6
【備　考】 リストではSTのようにCJ,JMP等を実行する必要がないため，比較部分のみ測定しています。 比較部分は導通しているものとして時間を算出しています。				

(c) FOR...DO文

単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	FOR D0 := 0 TO 10 BY 1 DO D1 := D1 + 1; END_FOR;	16	初期化：0.134 反復：3.308 この場合反復部は，10回動作する。	初期化：0.9 反復：24.0
リスト プログラム	FOR K10 LD SM400 INC D1 NEXT	6	2.574	21.6
【備　考】 上記演算処理時間が反復数分かかります。 リストでは反復回数の指定のみ可能です。STでは条件比較による反復などの演算処理を行うことが可能です。				

4 - 23

(d) WHILE...DO文

WHILE…DO文 1　　単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	WHILE X0 DO 　D0 := 100; END_WHILE;	10	3.034 反復は，X0がTRUEになるまで実施されます。	21.9
リスト プログラム	同上	同上	同上	同上
【備　考】 リストで記述してもSTプログラム変換結果と同様のプログラムになるため，処理時間もSTと同様となります。				

WHILE…DO文 2　　単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	WHILE D0= 100 DO 　D0 := 100; END_WHILE;	15	3.1	22.5
リスト プログラム	同上	同上	同上	同上
【備　考】 リストで記述してもSTプログラム変換結果と同様のプログラムになるため，処理時間もSTと同様となります。				

4 - 24

(e) REPEAT...UNTIL文

REPEAT…UNTIL文 1　　単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	REPEAT 　D0 := 100; 　UNTIL X0 END_REPEAT;	6	1.534 反復は，X0がTRUEになるまで実施されます。	10.9
リスト プログラム	同上	同上	同上	同上

【備　考】
リストで記述してもSTプログラム変換結果と同様のプログラムになるため，処理時間もSTと同様となります。

REPEAT…UNTIL文 2　　単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	REPEAT 　D0 := 100; 　UNTIL D0= 0 END_REPEAT;	9	1.6 反復は，D0が0になるまで実施されます。	11.5
リスト プログラム	同上	同上	同上	同上

【備　考】
リストで記述してもSTプログラム変換結果と同様のプログラムになるため，処理時間もSTと同様となります。

4 - 25

(f) EXIT文

単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	－	3	1.4	11
リスト プログラム	同上	同上	同上	同上

【備　考】

JMP命令により反復処理終了直後のポインタに移動します。

リストでもSTプログラムと同様の動作であるため，処理時間もSTと同様となります。

(g) RETURN文

単位(μs)

		ステップ数	演算処理時間 (Q25H)	演算処理時間 (Q00J)
ST プログラム	－	3	1.4	11
リスト プログラム	同上	同上	同上	同上

【備　考】

JMP命令により反復処理終了直後のポインタに移動します。

リストでもSTプログラムと同様の動作であるため，処理時間もSTと同様となります。

4 - 26

(2) ビットデバイス使用時の注意点

STプログラムでIF・CASE条件文を使用してプログラムを作成する場合に，同じ動作を行うプログラムをラダープログラムで作成する場合に注意する点について説明します。
IF条件文でブール式（条件式）が1度成り立つと，IF条件文内でビットデバイスをONしている場合，そのビットデバイスは常時ONとなります。

【STプログラム例1】

```
IF M0 THEN
      Y0 := TRUE;
END_IF;
```

上記プログラムは

```
LD M0;
SET Y0;
```

と等価となります。

常時ONとなるのを避けるためには，プログラムを下記の様に変更してください。

【STプログラム例2】

```
IF M0 THEN
      Y0 := TRUE;
ELSE
      Y0 := FALSE;
END_IF;
```

上記プログラムは

```
(a) LD M0;
    OUT Y0;

(b) Y0 := M0;

(c) OUT_M (M0,Y0);
```

と等価となります。

ただし，OUT_M()をIF条件文の文内で使用した場合，【STプログラム例1】と同様の状態となります。

上記注意点は，CASE条件文使用時も同様です。
CASE条件文で整数式（条件式）が1度成り立つと，CASE条件文内でビットデバイスをONしている場合，そのビットデバイスは常時ONとなります。

4 - 27

(3) タイマ，カウンタ使用時の注意点

STプログラムでIF・CASE条件文を使用してプログラムを作成する場合に，同じ動作を行うプログラムをラダープログラムで作成する場合に注意する点について説明します。
IF条件文でブール式（条件式）はタイマ・カウンタ命令の実行条件とは異なります。

例）タイマの場合

【STプログラム例1】

```
IF M0 THEN
    TIMER_M (M1,TC0,K10);
END_IF;
(* M0=ONかつM1=ON時，計数を開始します。                          *)
(* M0=ONかつM1=OFF時，計数をクリアします。                       *)
(* M0=OFFかつM1=ON時，計数を停止します。計数値はクリアしません。*)
(*M0=OFFかつM1=OFF時，計数を停止します。計数値はクリアしません。*)
```

例）カウンタの場合

【STプログラム例2】

```
IF M0 THEN
    COUNTER_M (M1,CC0,K10);
END_IF;
(* M0=ONかつM1=ON/OFF時，計数を+1します。                       *)
(* M0=OFFかつM1=ON/OFF時，計数しません。                        *)
(* M0=ON/OFFと計数+1とは連動していません。                      *)
```

上記はIF条件文が成立しない場合はタイマ・カウンタに関連する文が実行されないために起こります。
M0の条件とM1のAND条件によりタイマ・カウンタを動作させる場合は制御構文ではなく，MELSEC関数のみを使用して下さい。

【変更STプログラム例】

```
・タイマ使用時          TIMER_M (M0 & M1,TC0,K10);
・カウンタ使用時        COUNTER_M (M0 & M1,CC0,K10);
```

変更後のプログラムを使用することによりM0とM1のAND条件によりタイマ・カウンタを動作させることができます。

上記注意点は，CASE条件文使用時も同様です。
CASE条件文で整数式（条件式）はタイマ・カウンタ命令の実行条件とは異なります。

4 - 28

4.4　ファンクションブロックの呼出し

STプログラムではファンクションブロック（FB）を使用することができます。
STプログラムにおいて，ユーザが作成したFBを使用する方法について以下に説明します。（FBの作成方法については，『GX Developer Version8 オペレーティングマニュアル(ファンクションブロック編）』を参照してください。）

(1) ファンクションブロックの呼出し

作成したFBをSTプログラム上で使用する場合は，はじめにローカル変数設定画面にてFB名の定義を行う必要があります。（ 参考 を参照してください。）
STプログラム中で，FB名定義したFB名を記述すること（FB呼出し）によりFBが使用できます。

FB呼出し時に入力変数，入出力変数はすべて記述してください。また入力変数および入出力変数には必ず値を指定してください。
出力変数は出力変数の結果が不要であれば，記述を省略することができます。

【記述例】

FBデータ名　：LINE1_FB
入力変数　　：I_Test
出力変数　　：O_Test
入出力変数　：IO_Test
FBラベル名　：FB1　　というFBを作成した場合，

FB呼出しの記述例は以下のようになります。

```
FB1( I_Test := D0,O_Test := D1,IO_Test := D100);
                        ↓
                出力変数は記述を省略することができます。
```

(2) 出力結果の取得方法

FB名の後に”.”を付けて出力変数名を指定することでFBの出力を取得することができます。

【記述例】

出力変数の結果をD1に代入する場合は以下のように記述します。

```
D1 := FB1.O_Test;
```


4 - 29

参考

● FBの入力変数・入出力変数・出力変数のラベル宣言を行うには．．．

GX Developer起動 →[プロジェクトを開く] →FBタブをクリック →FB新規追加

→ 

→FB変数アイコンをダブルクリック

→ FBラベル設定画面

下記は，FBラベル設定画面でFBの入・出力変数ラベルを設定した例です。

	入出力種別	ﾗﾍﾞﾙ	定数値	ﾃﾞﾊﾞｲｽ種別
1	入力変数	I_TEST		ﾜｰﾄﾞ
2	出力変数	O_TEST		ﾜｰﾄﾞ
3	入出力変数	IO_TEST		ﾜｰﾄﾞ

●FBデータ名のラベル宣言を行うには．．．

FBの呼出し前に，使用するFBのラベル宣言を行う必要があります。

→ローカルラベル（もしくはグローバルラベル）を
ダブルクリック →ローカル（もしくはグローバル）変数設定画面

	Au	ﾗﾍﾞﾙ	定数値	ﾃﾞﾊﾞｲｽ種別
1	○	BLabel		ﾋﾞｯﾄ
2	○	DwLabel		ﾀﾞﾌﾞﾙﾜｰﾄﾞ
3	○	label2	詳細設定	FB(TEST_FB)

ｶｳﾝﾀ
積算ﾀｲﾏ
ﾎﾟｲﾝﾀ
構造体
FB
FB(TEST_FB)

↓
デバイス種別でFBを選択

下記は，ローカル変数設定画面でFBラベルを定義した例です。

3	○	label2	詳細設定	FB(TEST_FB)

4 - 30

Point

● **FBの出力取得を行う場合の注意事項**

FBの出力取得はFB呼出しの後で行ってください。FB呼出しの前に実行するとエラーとなります。

例）FB名 ： FB1
　　入力変数 ： I_Test
　　出力変数 ： O_Test

```
D1 := FB1.O_Test;                    (* FBの出力取得  *)
FB1(I_Test := D0,O_Test := D1);      (* FB呼出し      *)
```

FBの出力取得，FB呼出しの順なのでエラーとなります。

● **入出力変数を使用する場合の注意事項**

入出力変数の結果は出力変数のように使用するとエラーとなります。
入出力変数の値は入力変数と同じようにFB呼出し時に指定する必要があります。

例）FB名：FB1
　　入出力変数 ： IO_Test
　　出力変数　 ： O_Test

【記述例】

```
FB1( IO_Test := D1);
D1 := FB1.IO_Test; →エラーとなります。
```

● **FB呼出しを行う場合の注意事項**

STプログラム上では，ローカル変数設定画面で設定されたFBは1回のみ使用できます。（複数回使用した場合はエラーが発生します。）同じFBを複数回使用する場合は，事前にローカル変数設定画面で使用する回数分のFBの宣言を行ってください。

例）下記は，ローカル変数設定画面でFBラベルを複数回定義した例です。

	Au	ﾗﾍﾞﾙ	定数値	ﾃﾞﾊﾞｲｽ種別
1	○	label	詳細設定	FB(TEST_FB)
2	○	label1	詳細設定	FB(TEST_FB)
3	○	label2	詳細設定	FB(TEST_FB)

プログラムでは下記のように使用します。

```
label(I_Test := D0,IO_Test := D100);
label1(I_Test := D1,IO_Test := D150);
label2(I_Test := D3,IO_Test := D200);
```


4 - 31

4.5 コメント

STプログラム上では，コメントを入力することが可能です。“(*”と“*)”で囲まれている部分がコメントとして扱われます。 コメントの中にコメントを入れるとエラーが発生します。

【正常例】

例1)　(* ポンプを活性にする。 *)
例2)　(**********)
例3)　(* スイッチ入力後にモータを稼動する *)
例4)　(* Flag_A = TRUE　制御開始(*　Flag_B = TRUE　制御停止　*)

【エラー例】

例5)　(* Flag_A = TRUE 制御開始 *) Flag_A = FALSE 制御停止 *)
例6)　(* START (* 処理中断 *)再開　終了 *)

4 - 32

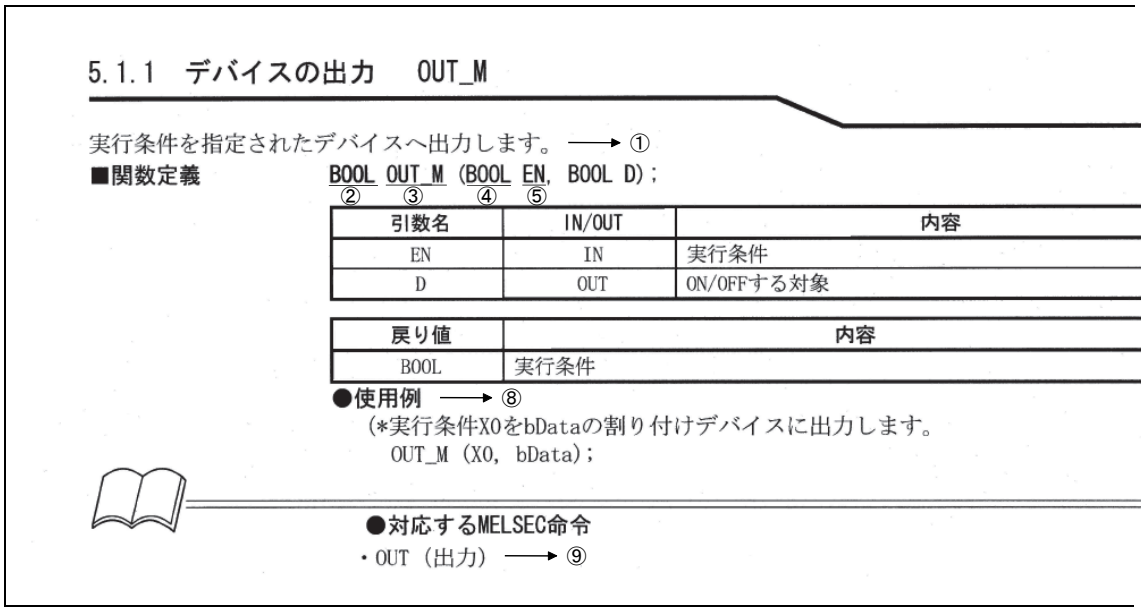
メ　モ

4 - 33

５　MELSEC関数

関数の見方

本書では，MELSEC関数の関数定義・引数・戻り値・使用例を記載しています。
MELSEC関数は，MELSEC共通命令を元に作られています。関数の対応CPU，基本的な動作，詳細機能，使用可能デバイス，関数実行時に発生するエラーについては
『MELSEC-Q/L プログラミングマニュアル（共通命令編）』を参照してください。
参照項は，本書『●対応するMELSEC命令』に記載されている項となります。



5.1.1　デバイスの出力　OUT_M

実行条件を指定されたデバイスへ出力します。 ⟶ ①

■関数定義　BOOL OUT_M (BOOL EN, BOOL D);
②　③　④　⑤

引数名	IN/OUT	内容
EN	IN	実行条件
D	OUT	ON/OFFする対象

⟶ ⑥

戻り値	内容
BOOL	実行条件

⟶ ⑦

●使用例 ⟶ ⑧

```
(*実行条件X0をbDataの割り付けデバイスに出力します。          *)
  OUT_M (X0, bData);
```

●対応するMELSEC命令

・OUT（出力） ⟶ ⑨

① 関数の機能を示します。
② 関数のデータ型を示します。
③ 関数名を示します。
④ 引数のデータ型を示します。（STRING型の表記はSTRING（文字数）です。文字数6の場合，STRING(6)となります。ARRAY型の表記はデータ型（要素数）です。ANY16型の配列で要素数3の場合，ANY16(3)となります。）
⑤ 引数名を示します。
⑥ 関数で使用する引数の一覧表（引数名・IN/OUT・内容）を示します。（STRING型の表記はARRAY［0..要素数-1］OF データ型です。ANY16型の配列で要素数3の場合，ARRAY［0..2］OF ANY16となります。）
⑦ 関数で使用する戻り値の一覧表（戻り値名・内容）を示します。
⑧ 関数の使用例を示します。（実デバイス・ラベルを使用した例を示します。）
⑨ 関数に対応するQCPU(Qモード)/LCPU MELSEC命令を示します。

5 - 1

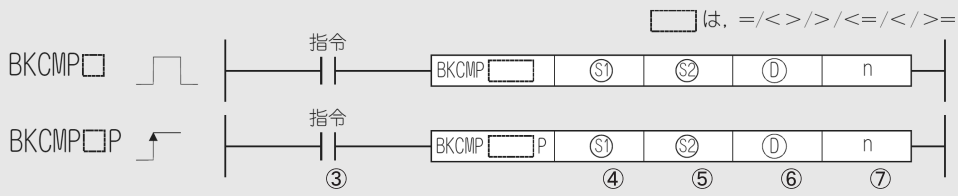
『MELSEC-Q/L プログラミングマニュアル（共通命令編）』MELSEC命令と本書MELSEC関数との対応を下記に示します。

MELSEC-Q/Lプログラミングマニュアル（共通命令編）『MELSEC命令』

6.1.6 BIN16 ビットブロックデータ比較 (BKCMP □ ,BKCMP □ P)

Basic　High performance　Process　Redundant　Universal　LCPU → ①

□は，=/<>/>/<=/</>=を示す。



Ⓢ1 ：比較されるデータまたは，比較されるデータが格納されているデバイスの先頭番号（BIN16 ビット）
Ⓢ2 ：比較データが格納されているデバイスの先頭番号（BIN16 ビット）
Ⓓ ：比較演算結果を格納するデバイスの先頭番号（ビット）
n ：比較するデータ数（BIN16 ビット）

設定データ	内部デバイス ビット	内部デバイス ワード	R, ZR	J□¥□ ビット	J□¥□ ワード	U□¥G□	Zn	定数 K, H	その他
Ⓢ1	—	○		—				○	—
Ⓢ2	—	○		—				—	—
Ⓓ	○	○		—				—	—
n	○	○		○				○	—

→ ②

本書『MELSEC関数』

5.4.1　ブロックデータ比較(=)　BKCMP_EQ_M

指定されたデバイスからn点のBIN16ビットデータ（ワード単位）を“=”で比較します。

■関数定義　　BOOL BKCMP_EQ_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);
⑧ ⑨ ⑩ ⑪ ⑫

5 - 2

① 使用可能CPUタイプ
　命令を使用可能なCPUタイプを示します。
② 使用可能デバイス
・MELSEC関数とMELSEC命令の引数の対応は下記となります。（同じ引数名同士が対応しています。）

③←→⑧	④←→⑨	⑤←→⑩	⑦←→⑪	⑥←→⑫

Point

● **MELSEC・IEC関数の引数を使用する場合の注意事項**

引数がANY32型の場合，指定可能なデータ型はDINT型なので
実デバイスは指定できません。ダブルワード型のラベルのみ
指定可能です。ただし桁指定は指定可能です。

```
例）BSQR_MD(BOOL EN,ANY16 s,ANY32 d);
                                    (* BSQR_MDの関数定義 *)
    BSQR_MD(X0,D0,dData);           (* プログラム例       *)
    MELSEC共通命令では,
        BSQR(D0,W0);
```

と実デバイスが記述できますがMELSEC・IEC関数では記述できません。

```
        BSQR_MD(X0,D0,W0);      ←エラーとなります。
```

引数がREAL型の場合，指定可能なデータ型は実数型のラベル
もしくは実数値を直接記述することが可能です。
実デバイスは指定できません。

```
  例）ESTR_M(BOOL EN,REAL s1,ANY16(3) s2,STRING d);
                                    (* ESTR_Mの関数定義 *)
      ESTR_M(X0,rData,ArrayData,sData);
                                    (* プログラム例     *)
      MELSEC共通命令では,
        ESTR(R0,R10,D10);
```

と実デバイスが記述できますがMELSEC・IEC関数では記述できません。

```
        ESTR_M(X0,R0,ArrayData,sData);←エラーとなります。
```


5 - 3

5.1　出力

5.1.1　デバイスの出力　OUT_M

実行条件を指定されたデバイスへ出力します。

■関数定義　　**BOOL** OUT_M (BOOL EN, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件
D	OUT	ON/OFFする対象

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0をbDataの割り付けデバイスに出力します。　　*)
  OUT_M (X0, bData);
```

●対応するMELSEC命令

・OUT（出力）

5.1.2　低速タイマ　TIMER_M

タイマ（低速タイマ，低速積算タイマ）のコイルがONして設定値まで計測し，タイムアップ（計算値≧設定値）すると接点は次のようになります。

a 接点：導通　　b接点：非導通

■関数定義　　**BOOL** TIMER_M (BOOL EN, BOOL TCoil, ANY16 TValue);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
TCoil	IN	TS,TCデバイスまたはSTS,STCデバイス（ビットデータ）
TValue	IN	タイマの設定値（BIN16ビットデータ）

備考）タイマの設定値に定数指定する場合，10進数のみ指定可能です。
　　タイマの設定値の範囲は0～32767まで指定可能。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，TC0はONしてTValueまで計算し，タイムアップ　　*)
(*（計算値≧設定値）すると接点は次のようになる。　　*)
(*a 接点：導通　b接点：非導通　　*)
  TIMER_M (X0, TC0, TValue);
```

●対応するMELSEC命令

・OUT T（低速タイマ）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5.1.3　高速タイマ　TIMER_H_M

タイマ（高速タイマ，高速積算タイマ）のコイルがONして設定値まで計算し，タイムアップ（計算値≧設定値）すると接点は次のようになります。
　a 接点：導通　　b接点：非導通

■関数定義

BOOL TIMER_H_M (BOOL EN, BOOL TCoil, ANY16 TValue);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
TCoil	IN	TS,TCデバイスまたはSTS,STCデバイス（ビットデータ）
TValue	IN	タイマの設定値（BIN16ビットデータ）

備考）タイマの設定値に定数を指定する場合，10進数のみ指定可能です。
　　タイマの設定値の範囲は0から32767まで指定可能。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，TC0はONしてTValueまで計算し，タイムアップ　　*)
(*（計算値≧TValueすると接点は次のようになる。　　*)
(*a 接点：導通　b接点：非導通　　*)
  TIMER_H_M (X0, TC0, TValue);
```

●対応するMELSEC命令

・OUTH T（高速タイマ）

5.1.4　カウンタ　COUNTER_M

カウンタの現在値（カウント値）を+1し，カウントアップ（現在値=設定値）すると接点は次のようになります。
　a 接点：導通　　b接点：非導通

■関数定義

BOOL COUNTER_M (BOOL EN, BOOL CCoil, ANY16 CValue);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
CCoil	IN	CS,CCデバイスの番号（ビットデータ）
CValue	IN	カウンタの設定値（BIN16ビットデータ）

備考）カウンタの設定値に定数を指定する場合，10進数のみ指定可能です。
　　カウンタの設定値の範囲は0から32767まで指定可能。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，CC0がOFF→ONに変化したとき現在値（カウント値）を*)
(*+1し，カウントアップ（現在値= CValue）すると接点は次のようになる。　*)
(*a 接点：導通　b接点：非導通　　*)
  COUNTER_M (X0, CC0, CValue);
```

●対応するMELSEC命令

・OUT C（カウンタ）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 5

5.1.5　デバイスのセット　SET_M

実行条件が成立時，指定されたデバイスを以下のようにします。
・ビットデバイス：コイル・接点をONします。
・ワードデバイスのビット指定時：指定ビットを1にします。

■関数定義　　**BOOL** SET_M (BOOL EN, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	OUT	セットするデータ

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONするとbDataの割り付けデバイスをONする。　　*)
  SET_M (X0, bData);
```

●対応するMELSEC命令
・SET（デバイスのセット）

5.1.6　デバイスのリセット　RST_M

実行条件が成立時，指定されたデバイスを以下のようにします。
・ビットデバイス：コイル・接点をOFFします。
・タイマ，カウンタ：現在値に0を代入し，コイル・接点をOFFします 。
・ワードデバイスのビット指定時：指定ビットを0にします 。
・タイマ，カウンタ以外のワードデバイス：デバイス内容に0を代入します。

■関数定義　　**BOOL** RST_M (BOOL EN, ANY_SIMPLE D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	OUT	リセットするデータ

備考）引数“D”にDINT/REAL/STRING型は使用できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONするとbDataの割り付けデバイスがOFFします。　　*)
  RST_M (X0, bData);
```

●対応するMELSEC命令
・RST（デバイスのリセット）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5.1.7　ダイレクト出力のパルス化　DELTA_M

実行条件成立時，指定されたダイレクトアクセス出力(DY)をパルス出力します。

■関数定義　　**BOOL** DELTA_M (BOOL EN, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	OUT	パルス出力するデータ（DYデバイス）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONするとデバイスDY0がパルス化します。　　*)
  DELTA_M (X0, DY0);
```

●対応するMELSEC命令

・DELTA（ダイレクト出力のパルス化）

5.2　シフト

5.2.1　デバイスの1ビットシフト　SFT_M

実行条件が成立時，指定されたデバイスを以下のようにします。

・ビットデバイスの場合：

指定されたデバイス番号より1つ前のデバイス番号のON・OFF状態を指定したデバイスにシフトし，1つ前のデバイス番号をOFFにします。

・ワードデバイスのビット指定の場合：

指定されたデバイスのビットより1つ前のビットの1・0状態を指定のビットにシフトし，1つ前のビットを0にします。

■関数定義　　**BOOL** SFT_M (BOOL EN, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	OUT	シフトするデータ

戻り値	内容
BOOL	実行条件

●使用例

```
(* 実行条件X0がONすると，M10のON・OFFをM11にシフトしM10をOFFします。　　*)
  SFT_M (X0, M11);
(* 実行条件X0がONすると，W100.1のON・OFFをW100.2にシフトしW100.1をOFFします。*)
  SFT_M (X0, W100.2);
```

●対応するMELSEC命令

・SFT（ビットデバイスシフト）

5 - 8

5.3　終了

5.3.1　停止　　STOP_M

実行条件が成立時，出力Yをリセットし，CPUの演算を停止します。
（RUN・STOPディップスイッチをSTOP側にした場合と同一です。）

■関数定義　　**BOOL** STOP_M (BOOL EN);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）

戻り値	内容
BOOL	実行条件

●使用例

```
(* 実行条件X0がONするとCPUの演算を停止します。                    *)
   STOP_M (X0);
```

●対応するMELSEC命令

・STOP（シーケンスプログラム停止）

5 - 9

5.4 比較演算

5.4.1 ブロックデータ比較(=) BKCMP_EQ_M

指定されたデバイスからn点のBIN16ビットデータ（ワード単位）を“=”で比較します。

■関数定義 **BOOL** BKCMP_EQ_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

引数名	IN/OUT	内容			
EN	IN	実行条件（TRUE時のみ関数を実行します）			
S1	IN	比較されるデータ（BIN16ビットデータ）			
S2	IN	比較するデータ（BIN16ビットデータ）			
n	IN	比較するデータ数（BIN16ビットデータ）			
D	OUT	比較結果（ビット）	比較結果	比較条件成立時	ON
				比較条件不成立時	OFF

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと  *)
(*D200からD0に格納されている値の点数分のデータを“=”で比較演算し，その結果を*)
(*M0以降に格納します。                                                *)
  BKCMP_EQ_M (X0, D100, D200, D0, M0);
```

●対応するMELSEC命令

・BKCMP= (BIN16ビットブロックデータ比較(=))

5.4.2 ブロックデータ比較(<>) BKCMP_NE_M

指定されたデバイスからn点のBIN16ビットデータ（ワード単位）を“<>”で比較します。

■関数定義 **BOOL** BKCMP_NE_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

引数名	IN/OUT	内容			
EN	IN	実行条件（TRUE時のみ関数を実行します）			
S1	IN	比較されるデータ（BIN16ビットデータ）			
S2	IN	比較するデータ（BIN16ビットデータ）			
n	IN	比較するデータ数（BIN16ビットデータ）			
D	OUT	比較結果（ビット）	比較結果	比較条件成立時	ON
				比較条件不成立時	OFF

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと  *)
(*D200からD0に格納されている値の点数分のデータを“<>”で比較演算し，その結果*)
(*をM0以降に格納します。                                              *)
  BKCMP_NE_M (X0, D100, D200, D0, M0);
```

●対応するMELSEC命令

・BKCMP<> (BIN16ビットブロックデータ比較(<>))

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 10

5.4.3　ブロックデータ比較(＞)　BKCMP_GT_M

指定されたデバイスからn点のBIN16ビットデータ（ワード単位）を“＞”で比較します。

■関数定義　　**BOOL** BKCMP_GT_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

引数名	IN/OUT	内容			
EN	IN	実行条件（TRUE時のみ関数を実行します）			
S1	IN	比較されるデータ（BIN16ビットデータ）			
S2	IN	比較するデータ（BIN16ビットデータ）			
n	IN	比較するデータ数（BIN16ビットデータ）			
D	OUT	比較結果（ビット）	**比較結果**	比較条件成立時	ON
				比較条件不成立時	OFF

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと　*)
(*D200からD0に格納されている値の点数分のデータを“＞”で比較演算し，その結果を*)
(*M0以降に格納します。　*)
  BKCMP_GT_M (X0, D100, D200, D0, M0);
```

●対応するMELSEC命令

・BKCMP> (BIN16ビットブロックデータ比較(>))

5.4.4　ブロックデータ比較(<=)　BKCMP_LE_M

指定されたデバイスからn点のBIN16ビットデータ（ワード単位）を“＜＝”で比較します。

■関数定義　　**BOOL** BKCMP_LE_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

引数名	IN/OUT	内容			
EN	IN	実行条件（TRUE時のみ関数を実行します）			
S1	IN	比較されるデータ（BIN16ビットデータ）			
S2	IN	比較するデータ（BIN16ビットデータ）			
n	IN	比較するデータ数（BIN16ビットデータ）			
D	OUT	比較結果（ビット）	**比較結果**	比較条件成立時	ON
				比較条件不成立時	OFF

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと　*)
(*D200からD0に格納されている値の点数分のデータを“＜＝”で比較演算し，その結果*)
(*をM0以降に格納します。　*)
  BKCMP_LE_M (X0, D100, D200, D0, M0);
```

●対応するMELSEC命令

・BKCMP<= (BIN16ビットブロックデータ比較(<=))

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5.4.5　ブロックデータ比較(<)　BKCMP_LT_M

指定されたデバイスからn点のBIN16ビットデータ（ワード単位）を“<”で比較します。

■関数定義　　**BOOL** BKCMP_LT_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

引数名	IN/OUT	内容			
EN	IN	実行条件（TRUE時のみ関数を実行します）			
S1	IN	比較されるデータ（BIN16ビットデータ）			
S2	IN	比較するデータ（BIN16ビットデータ）			
n	IN	比較するデータ数（BIN16ビットデータ）			
D	OUT	比較結果（ビット）	**比較結果**	比較条件成立時	ON
				比較条件不成立時	OFF

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと　*)
(*D200からD0に格納されている値の点数分のデータを“<”で比較演算し，その結果*)
(*をM0以降に格納します。　*)
  BKCMP_LT_M (X0, D100, D200, D0, M0);
```

●対応するMELSEC命令

・BKCMP< (BIN16ビットブロックデータ比較(<))

5.4.6　ブロックデータ比較(>=)　BKCMP_GE_M

指定されたデバイスからn点のBIN16ビットデータ（ワード単位）を“＞＝”で比較します。

■関数定義　　**BOOL** BKCMP_GE_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

引数名	IN/OUT	内容			
EN	IN	実行条件（TRUE時のみ関数を実行します）			
S1	IN	比較されるデータ（BIN16ビットデータ）			
S2	IN	比較するデータ（BIN16ビットデータ）			
n	IN	比較するデータ数（BIN16ビットデータ）			
D	OUT	比較結果（ビット）	**比較結果**	比較条件成立時	ON
				比較条件不成立時	OFF

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと　*)
(*D200からD0に格納されている値の点数分のデータを“＞＝”で比較演算し，その結果*)
(*をM0以降に格納します。　*)
  BKCMP_GE_M (X0, D100, D200, D0, M0);
```

●対応するMELSEC命令

・BKCMP>= (BIN16ビットブロックデータ比較(>=))

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 12

5.5　算術演算

5.5.1　BCD4桁の加算(2デバイス)　BPLUS_M

指定された2つのBCD4桁データを加算します。

■関数定義

```
BOOL BPLUS_M (BOOL EN, ANY16 S1, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	加算するデータ（BCD4桁データ）
D	IN/OUT	加算されるデータ・加算結果（BCD4桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0とD100に格納されているBCD4桁データの加算を行い，*)
(*加算結果をD100に格納します。                                    *)
  BPLUS_M (X0, D0, D100);
```

●対応するMELSEC命令

・B+（BCD4桁データ加算）

5.5.2　BCD4桁の加算(3デバイス)　BPLUS_3_M

指定された2つのBCD4桁データを加算します。

■関数定義

```
BOOL BPLUS_3_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	加算されるデータ（BCD4桁データ）
S2	IN	加算するデータ（BCD4桁データ）
D	OUT	加算結果（BCD4桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D1とD2に格納されているBCD4桁データの加算を行い，*)
(*加算結果をD100に格納します。                                    *)
  BPLUS_3_M (X0, D1, D2, D100);
```

●対応するMELSEC命令

・B+（BCD4桁データ加算）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 13

5.5.3　BCD4桁の減算(2デバイス)　BMINUS_M

指定された2つのBCD4桁データを減算します。

■関数定義　　**BOOL** BMINUS_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	減算するデータ (BCD4桁データ)
D	IN/OUT	減算されるデータ・減算結果 (BCD4桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0とD100に格納されているBCD4桁データの減算を行い，*)
(*減算結果をD100に格納します。                                    *)
  BMINUS_M (X0, D0, D100);
```

●対応するMELSEC命令

・B- (BCD4桁データ減算)

5.5.4　BCD4桁の減算(3デバイス)　BMINUS_3_M

指定された2つのBCD4桁データを減算します。

■関数定義　　**BOOL** BMINUS_3_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	減算されるデータ (BCD4桁データ)
S2	IN	減算するデータ (BCD4桁データ)
D	OUT	減算結果 (BCD4桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D1とD2に格納されているBCD4桁データの減算を行い，*)
(*減算結果をD100に格納します。                                    *)
  BMINUS_3_M (X0, D1, D2, D100);
```

●対応するMELSEC命令

・B- (BCD4桁データ減算)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 14

5.5.5　BCD8桁の加算(2デバイス)　DBPLUS_M

指定された2つのBCD8桁データを加算します。

■関数定義　　**BOOL** DBPLUS_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	加算データ (BCD8桁データ)
D	IN/OUT	加算されるデータ・加算結果 (BCD8桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, dwData1とResultに格納されているBCD8桁データの　*)
(*加算を行い, 加算結果をResultに格納します。　*)
  DBPLUS_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DB+ (BCD8桁データ加算)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5.5.6　BCD8桁の加算(3デバイス)　DBPLUS_3_M

指定された2つのBCD8桁データを加算します。

■関数定義　　**BOOL** DBPLUS_3_M (BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	加算されるデータ (BCD8桁データ)
S2	IN	加算するデータ (BCD8桁データ)
D	OUT	加算結果 (BCD8桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, dwData1とdwData2に格納されているBCD8桁データの　*)
(*加算を行い, 加算結果をResultに格納します。　*)
  DBPLUS_3_M (X0, dwData1, dwData2, Result);
```

●対応するMELSEC命令

・DB+ (BCD8桁データ加算)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 15

5.5.7　BCD8桁の減算(2デバイス)　DBMINUS_M

指定された2つのBCD8桁データを減算します。

■関数定義　　**BOOL** DBMINUS_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	減算するデータ (BCD8桁データ)
D	IN/OUT	減算されるデータ・減算結果 (BCD8桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, dwData1とResultに格納されているBCD8桁データの　*)
(*減算を行い, 減算結果をResultに格納します。　*)
  DBMINUS_M (X0, dwData1,Result);
```

●対応するMELSEC命令

・DB- (BCD8桁データ減算)

5.5.8　BCD8桁の減算(3デバイス)　DBMINUS_3_M

指定された2つのBCD8桁データを減算します。

■関数定義　　**BOOL** DBMINUS_3_M (BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	減算されるデータ (BCD8桁データ)
S2	IN	減算するデータ (BCD8桁データ)
D	OUT	減算結果 (BCD8桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, dwData1とdwData2に格納されているBCD8桁データの　*)
(*減算を行い, 減算結果をResultに格納します。　*)
  DBMINUS_3_M (X0, dwData1, dwData2, Result);
```

●対応するMELSEC命令

・DB- (BCD8桁データ減算)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 16

5.5.9　BCD4桁の乗算　BMULTI_M

指定された2つのBCD4桁データを乗算します。

■関数定義　　**BOOL** BMULTI_M (BOOL EN, ANY16 S1, ANY16 S2,ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	乗算されるデータ（BCD4桁データ）
S2	IN	乗算するデータ（BCD4桁データ）
D	OUT	乗算結果（BCD8桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D1とD2に格納されているBCD4桁データの乗算を行い，*)
(*乗算結果をResultに格納します。                                    *)
  BMULTI_M (X0, D1, D2, Result);
```

●対応するMELSEC命令

・B*（BCD4桁データ乗算）

5.5.10　BCD4桁の除算　BDIVID_M

指定された2つのBCD4桁データを除算します。

■関数定義　　**BOOL** BDIVID_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16(2) D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	除算されるデータ（BCD4桁データ）		
S2	IN	除算するデータ（BCD4桁データ）		
D	OUT	除算結果 (ARRAY [0..1] OF ANY16)	D[0] D[1]	商 余り

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D1とD2に格納されているBCD4桁データの除算を行い，*)
(*除算結果を配列ArrayResultに格納します。                          *)
  BDIVID_M (X0, D1, D2, ArrayResult);
```

●対応するMELSEC命令

・B/（BCD4桁データ除算）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 17

5.5.11　BCD8桁の乗算　DBMULTI_M

指定された2つのBCD8桁データを乗算します。

■関数定義　　**BOOL** DBMULTI_M (BOOL EN, ANY32 S1, ANY32 S2, ANY16(4) D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	乗算されるデータ（BCD8桁データ）		
S2	IN	乗算するデータ（BCD8桁データ）		
D	OUT	乗算結果 (ARRAY [0..3] OF ANY16)	D[0]	下4桁
			D[1]	↓
			D[2]	
			D[3]	上4桁

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1とdwData2に格納されているBCD8桁データの　*)
(*乗算を行い，乗算結果を配列ArrayResultに格納します。　*)
  DBMULTI_M (X0, dwData1, dwData2, ArrayResult);
```

●対応するMELSEC命令

・DB* (BCD8桁データ乗算)

5.5.12　BCD8桁の除算　DBDIVID_M

指定された2つのBCD8桁データを除算します。

■関数定義　　**BOOL** DBDIVID_M (BOOL EN, ANY32 S1, ANY32 S2, ANY32(2) D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	除算されるデータ（BCD8桁データ）		
S2	IN	除算するデータ（BCD8桁データ）		
D	OUT	除算結果 (ARRAY [0..1] OF ANY32)	D[0]	商
			D[1]	余り

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1とdwData2に格納されているBCD8桁データの　*)
(*除算を行い，除算結果を配列ArrayResultに格納します。　*)
  DBDIVID_M (X0, dwData1, dwData2, ArrayResult);
```

●対応するMELSEC命令

・DB/ (BCD8桁データ除算)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5.5.13　文字列データ結合(2デバイス)　STRING_PLUS_M

指定された文字列データを連結します。

■関数定義　　**BOOL** STRING_PLUS_M (BOOL EN, STRING S1, STRING D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	連結するデータ（文字列データ）
D	IN/OUT	連結されるデータ・連結結果（文字列データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,StrResult に格納されている文字列の後ろに文字列"ABC"を*)
(*結合し，結合した文字列をStrResultに格納します。　　*)
  STRING_PLUS_M (X0, "ABC", StrResult);
```

●対応するMELSEC命令

・$+（文字列の結合）

5.5.14　文字列データ結合(3デバイス)　STRING_PLUS_3_M

指定された文字列データを連結します。

■関数定義　　**BOOL** STRING_PLUS_3_M (BOOL EN, STRING S1, STRING S2, STRING D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	連結されるデータ（文字列データ）
S2	IN	連結するデータ（文字列データ）
D	OUT	連結結果（文字列データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,StrData1に格納されている文字列の後ろにStrData2に格納*)
(*されている文字列を結合し，結合した文字列をStrResultに格納します。　　*)
  STRING_PLUS_3_M (X0, StrData1,StrData2, StrResult);
```

●対応するMELSEC命令

・$+（文字列の結合）

5 - 19

5.5.15　BINブロック加算　BKPLUS_M

指定されたデバイスからn点のBIN16ビットデータを加算します。

■関数定義

```
BOOL BKPLUS_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n ,ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	加算されるデータ（BIN16ビットデータ）
S2	IN	加算するデータ（BIN16ビットデータ）
n	IN	加算するデータ数（BIN16ビットデータ）
D	OUT	加算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと， *)
(*D200からD0に格納されている値の点数分のデータを加算し，その結果をD1000以降*)
(*に格納します。                                                *)
  BKPLUS_M (X0, D100, D200, D0, D1000);
```

●対応するMELSEC命令

・BK+（ブロックデータ加算）

5.5.16　BINブロック減算　BKMINUS_M

指定されたデバイスからn点のBIN16ビットデータを減算します。

■関数定義

```
BOOL BKMINUS_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	減算されるデータ（BIN16ビットデータ）
S2	IN	減算するデータ（BIN16ビットデータ）
n	IN	減算するデータ数（BIN16ビットデータ）
D	OUT	減算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと， *)
(*D200からD0に格納されている値の点数分のデータを減算し，その結果をD1000以降*)
(*に格納します。                                                *)
  BKMINUS_M (X0, D100, D200, D0, D1000);
```

●対応するMELSEC命令

・BK-（ブロックデータ減算）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 20

5.5.17　インクリメント　INC_M

指定されたBIN16ビットデータをインクリメント(+1)します。

■関数定義　　BOOL INC_M (BOOL EN, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	加算するデータ・加算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0に格納されているデータを+1します。　　*)
  INC_M (X0, D0);
```

●対応するMELSEC命令

・INC（BIN16ビットインクリメント）

5.5.18　デクリメント　DEC_M

指定されたBIN16ビットデータをデクリメント(-1)します。

■関数定義　　BOOL DEC_M (BOOL EN, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	減算するデータ・減算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0に格納されているデータを-1します。　　*)
  DEC_M (X0, D0);
```

●対応するMELSEC命令

・DEC（BIN16ビットデクリメント）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 21

5.5.19　32ビットBINインクリメント　DINC_M

指定されたBIN32ビットデータをインクリメント(+1)します。

■関数定義　　BOOL DINC_M (BOOL EN, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	加算するデータ・加算結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1に格納されているデータを+1します。　　*)
  DINC_M (X0, dwData1);
```

●対応するMELSEC命令

・DINC（BIN32ビットインクリメント）

5.5.20　32ビットBINデクリメント　DDEC_M

指定されたBIN32ビットデータをデクリメント(-1)します。

■関数定義　　BOOL DDEC_M (BOOL EN, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	減算するデータ・減算結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1に格納されているデータを-1します。　　*)
  DDEC_M (X0, dwData1);
```

●対応するMELSEC命令

・DDEC（BIN32ビットデクリメント）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 22

5.6　データ変換

5.6.1　BIN→BCD変換　BCD_M

指定されたBIN16ビットデータ(0～9999)をBCD4桁データに変換します。

■関数定義　　**BOOL** BCD_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	変換するデータ (BIN16ビットデータ)
D	OUT	変換結果 (BCD4桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0に格納されているBINデータをBCD変換してD100に　*)
(*格納します。　*)
  BCD_M (X0, D0, D100);
```

●対応するMELSEC命令

・BCD (BIN→BCD4桁)

5.6.2　32ビットBIN→BCD変換　DBCD_M

指定されたBIN32ビットデータ(0～99999999)をBCD8桁データに変換します。

■関数定義　　**BOOL** DBCD_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	変換するデータ (BIN32ビットデータ)
D	OUT	変換結果 (BCD8桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1に格納されているBINデータをBCD変換して　*)
(*Resultに格納します。　*)
  DBCD_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DBCD (BIN→BCD8桁)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 23

5.6.3　BCD→BIN変換　BIN_M

指定されたBCD4桁データ(0～9999)をBIN16ビットデータに変換します。

■関数定義　　**BOOL** BIN_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	変換するデータ (BCD4桁データ)
D	OUT	変換結果 (BIN16ビットデータ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0に格納されているBCDデータをBIN変換してD100に　*)
(*格納します。　*)
  BIN_M (X0, D0, D100);
```

●対応するMELSEC命令

・BIN (BCD4桁→BIN)

5.6.4　32ビットBCD→BIN変換　DBIN_M

指定されたBCD8桁データ(0～99999999)をBIN32ビットデータに変換します。

■関数定義　　**BOOL** DBIN_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	変換するデータ (BCD8桁データ)
D	OUT	変換結果 (BIN32ビットデータ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1に格納されているBCDデータをBIN変換して　*)
(*Resultに格納します。　*)
  DBIN_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DBIN (BCD8桁→BIN)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 24

5.6.5　浮動小数点→BIN変換　INT_E_MD

指定された実数データをBIN16ビットデータに変換します。

■関数定義　　**BOOL** INT_E_MD (BOOL EN, REAL S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（実数データ）
D	OUT	変換結果（BIN16ビットデータ）

備考）引数“S1”で指定する実数データは，-32768～32767の範囲内が指定できます。
変換後のデータは，実数の小数点以下1桁目を四捨五入した値となります。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，RealData1の実数データをBIN16ビットデータに変換し，*)
(*D0に格納します。                                                *)
  INT_E_MD (X0, RealData1, D0);
```

●対応するMELSEC命令

・INT（浮動小数点データ→BIN16ビット変換（単精度））

5.6.6　32ビット浮動小数点→BIN変換　DINT_E_MD

指定された実数データをBIN32ビットデータに変換します。

■関数定義　　**BOOL** DINT_E_MD (BOOL EN, REAL S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（実数データ）
D	OUT	変換結果（BIN32ビットデータ）

備考）引数“S1”で指定する実数データは，-2147483648～2147483647の範囲内が指定できます。
変換後のデータは，実数の小数点以下1桁目を四捨五入した値となります。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，実数データE2.6をBIN16ビットデータに変換し，      *)
(*Resultに格納します。                                              *)
  DINT_E_MD (X0, E2.6, Result);
```

●対応するMELSEC命令

・DINT（浮動小数点データ→BIN32ビット変換（倍精度））

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 25

5.6.7　BIN→浮動小数点変換　FLT_M

指定されたBIN16ビットデータを実数データに変換します。

■関数定義　　**BOOL** FLT_M (BOOL EN, ANY16 S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN16ビットデータ）
D	OUT	変換結果（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100のBIN16ビットデータを実数データに変換し，　*)
(*Resultに格納します。　*)
  FLT_M (X0, D100, Result);
```

●対応するMELSEC命令

・FLT（BIN16ビット→浮動小数点変換（単精度））

5.6.8　32ビットBIN→浮動小数点変換　DFLT_M

指定されたBIN32ビットデータを実数データに変換します。

■関数定義　　**BOOL** DFLT_M (BOOL EN, ANY32 S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN32ビットデータ）
D	OUT	変換結果（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1のBIN32ビットデータを実数データに変換し，*)
(*Resultに格納します。　*)
  DFLT_M (X0, dwData1, RealResult);
```

●対応するMELSEC命令

・DFLT（BIN32ビット→浮動小数点（倍精度））

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 26

5.6.9　16ビットBIN→32ビットBIN変換　DBL_M

指定されたBIN16ビットデータを符号付でBIN32ビットデータに変換します。

■関数定義　　**BOOL** DBL_M (BOOL EN, ANY16 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN16ビットデータ）
D	OUT	変換結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0のBIN16ビットデータ を符号付BIN32ビットデータに *)
(*変換しResultに格納します。                                          *)
  DBL_M (X0, D0, Result);
```

●対応するMELSEC命令

・DBL（BIN16ビット→BIN32ビット）

5.6.10　32ビットBIN→16ビットBIN変換　WORD_M

指定されたBIN32ビットデータを符号付でBIN16ビットデータに変換します。

■関数定義　　**BOOL** WORD_M (BOOL EN, ANY32 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN32ビットデータ）
D	OUT	変換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1に格納されているBIN32ビットデータを符号付  *)
(*BIN16ビットデータに変換し，D0に格納します。                              *)
  WORD_M (X0, dwData1, D0);
```

●対応するMELSEC命令

・WORD（BIN32ビット→BIN16ビット）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 27

5.6.11　BIN→グレイコード変換　GRY_M

指定されたBIN16ビットデータをグレイコード16ビットデータに変換します。

■関数定義　　**BOOL** GRY_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN16ビットデータ）
D	OUT	変換結果（グレイコード16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0のBIN16ビットデータをグレイコード16ビット　　*)
(*データに変換しD100に格納します。　　*)
  GRY_M (X0, D0, D100);
```

●対応するMELSEC命令

・GRY（BIN16ビット→グレイコード）

5.6.12　32ビットBIN→グレイコード変換　DGRY_M

指定されたBIN32ビットデータをグレイコード32ビットデータに変換します。

■関数定義　　**BOOL** DGRY_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN32ビットデータ）
D	OUT	変換結果（グレイコード32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1のBIN32ビットデータをグレイコード32　　*)
(*ビットデータに変換しResultに格納します。　　*)
  DGRY_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DGRY（BIN32ビット→グレイコード）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 28

5.6.13　グレイコード→BIN変換　GBIN_M

指定されたグレイコード16ビットデータをBIN16ビットデータに変換します。

■関数定義　　**BOOL** GBIN_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（グレイコード16ビットデータ）
D	OUT	変換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，D100のグレイコード16ビットデータをBIN16ビットデータ*)
(*  に変換しD200へ格納します。                                          *)
    GBIN_M (X0, D100, D200);
```

●対応するMELSEC命令

・GBIN（グレイコード→BIN16ビット）

5.6.14　32ビットグレイコード→BIN変換　DGBIN_M

指定されたグレイコード32ビットデータをBIN32ビットデータに変換します。

■関数定義　　**BOOL** DGBIN_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（グレイコード32ビットデータ）
D	OUT	変換結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，dwData1のグレイコード32ビットデータをBIN32     *)
(*  ビットデータに変換しResultに格納します。                             *)
    DGBIN_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DGBIN（グレイコード→BIN16ビット）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 29

5.6.15　16ビットBINの2の補数　NEG_M

指定されたBIN16ビットデータの符号を反転します。（2の補数）

■関数定義　　BOOL NEG_M (BOOL EN, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	符号反転するデータ・符号反転結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,D0のBIN16ビットデータの符号を反転してD0に格納します。*)
  NEG_M (X0, D0);
```

●対応するMELSEC命令

・NEG（BIN16ビットデータ2の補数）

5.6.16　32ビットBINの2の補数　DNEG_M

指定されたBIN32ビットデータの符号を反転します。（2の補数）

■関数定義　　BOOL DNEG_M (BOOL EN, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	符号反転するデータ・符号反転結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，ResultのBIN32ビットデータの符号を反転して　　*)
(*Resultに格納します。　　*)
  DNEG_M (X0, Result);
```

●対応するMELSEC命令

・DNEG（BIN32ビットデータ2の補数）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 30

5.6.17　浮動小数点の2の補数　ENEG_M

指定された実数データの符号を反転します。（2の補数）

■関数定義　　BOOL ENEG_M (BOOL EN, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	符号反転するデータ・変換結果（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，Resultの実数データの符号を反転して　　*)
(*Resultに格納します。　　*)
  ENEG_M (X0, Result);
```

●対応するMELSEC命令

・ENEG（浮動小数点データ符号反転（単精度））

5.6.18　ブロック変換BIN→BCD変換　BKBCD_M

指定されたデバイスからn点のBIN16ビットデータ(0～9999)をBCD4桁データに変換します。

■関数定義　　BOOL BKBCD_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN16ビットデータ）
n	IN	変数するデータ数
D	OUT	変換結果（BCD4桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0からW0に格納されている値の点数分のBIN16ビットデータ*)
(*をBCD変換し，その結果をD100以降に格納します。　　*)
  BKBCD_M (X0, D0, W0, D100);
```

●対応するMELSEC命令

・BKBCD（ブロックBIN16ビットデータ→BCD4桁変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 31

5.6.19　ブロック変換BCD→BIN変換　BKBIN_M

指定されたデバイスからn点のBCD4桁データ(0～9999)をBIN16ビットデータに変換します。

■関数定義　　**BOOL** BKBIN_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BCD4桁データ）
n	IN	変数するデータ数
D	OUT	変換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0からW0に格納されている値の点数分のBCDデータを *)
(*BIN変換して，その結果をD100以降に格納します。                          *)
  BKBIN_M (X0, D0, W0, D100);
```

●対応するMELSEC命令

・BKBIN（ブロックBCD4桁データ→BIN16ビット変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 32

5.7　データ転送

5.7.1　16ビットデータ否定転送　CML_M

指定されたBIN16ビットデータをビットごとに反転します。

■関数定義　　**BOOL** CML_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	ビット反転するデータ（BIN16ビットデータ）
D	OUT	反転結果転送先（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，M0～M7のデータを反転してD0へ転送します。　　*)
  CML_M (X0, K2M0, D0);
```

●対応するMELSEC命令

・CML（16ビット否定転送）

5.7.2　32ビットデータ否定転送　DCML_M

指定されたBIN32ビットデータをビットごとに反転します。

■関数定義　　**BOOL** DCML_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	ビット反転するデータ（BIN32ビットデータ）
D	OUT	反転結果転送先（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1のデータをビットごとに反転してResultへ　*)
(*転送します。　　*)
  DCML_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DCML（32ビット否定転送）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 33

5.7.3　ブロック転送　BMOV_M

指定されたデバイスからn点のBIN16ビットデータを一括転送します。

■関数定義　　**BOOL** BMOV_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	転送するデータ（BIN16ビットデータ）
n	IN	転送するデータ数（BIN16ビットデータ）
D	OUT	転送先（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0で指定されたデバイスからW0に格納されている値の*)
(*点数分の16ビットデータをD100からW0に格納されている値の点数分へ転送　　*)
(*します。　　*)
  BMOV_M (X0, D0, W0, D100);
```

●対応するMELSEC命令

・BMOV（ブロック16ビット転送）

5.7.4　同一データブロック転送　FMOV_M

指定されたデバイスの16ビットデータを指定されたデバイスからn点分転送します。

■関数定義　　**BOOL** FMOV_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	転送するデータ（BIN16ビットデータ）
n	IN	転送するデータ数（BIN16ビットデータ）
D	OUT	転送先（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,D0の16ビットデータをD100からW0に格納されている点数分*)
(*転送します。　　*)
  FMOV_M (X0, D0, W0, D100);
```

●対応するMELSEC命令

・FMOV（ブロック16ビットデータ転送）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 34

5.7.5　16ビットデータ交換　XCH_M

指定された2つのBIN16ビットデータを交換します。

■関数定義　　**BOOL** XCH_M (BOOL EN, ANY16 D1, ANY16 D2);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D1	IN/OUT	交換するデータ・交換結果（BIN16ビットデータ）
D2	IN/OUT	交換するデータ・交換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100とD200の16ビットデータの交換をします。　　*)
  XCH_M (X0, D100, D200);
```

●対応するMELSEC命令

・XCH（16ビットデータ交換）

5.7.6　32ビットデータ交換　DXCH_M

指定された2つのBIN32ビットデータを交換します。

■関数定義　　**BOOL** DXCH_M (BOOL EN, ANY32 D1, ANY32 D2);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D1	IN/OUT	交換するデータ・交換結果（BIN32ビットデータ）
D2	IN/OUT	交換するデータ・交換結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1とdwData2の32ビットデータの交換をします。*)
  DXCH_M (X0, dwData1, dwData2);
```

●対応するMELSEC命令

・DXCH（32ビットデータ交換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 35

5.7.7　ブロックデータ交換　BXCH_M

指定されたデバイスからn点のBIN16ビットデータを交換します。

■関数定義　　**BOOL** BXCH_M (BOOL EN, ANY16 n, ANY16 D1, ANY16 D2);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	交換するデータ数（BIN16ビットデータ）
D1	IN/OUT	交換するデータ・交換結果（BIN16ビットデータ）
D2	IN/OUT	交換するデータ・交換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*D100から3点分の16ビットデータとD200から3点分の16ビットデータを交換します。*)
BXCH_M (X0, K3, D100, D200);
```

●対応するMELSEC命令

・BXCH（ブロック16ビット交換）

5.7.8　上下バイト交換　SWAP_MD

指定されたデバイスの上位8ビット，下位8ビットを交換します。

■関数定義　　**BOOL** SWAP_MD (BOOL EN, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	IN/OUT	交換するデータ・交換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0の上位8ビットと下位8ビットを交換します。　　*)
SWAP_MD (X0, D0);
```

●対応するMELSEC命令

・SWAP（上下バイト交換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 36

5.8　プログラム実行制御

5.8.1　割込禁止　DI_M

割込みプログラムの割込み要因が発生しても，EI_Mが実行されるまで割込みプログラムの実行を禁止します。

■関数定義　　BOOL DI_M (BOOL EN);

引数名	IN/OUT	内容
EN	IN	実行条件 （常に有効を示す値TRUEまたは常時ONデバイスSM400のみ指定可能。）

戻り値	内容
BOOL	実行条件（常時TRUE）

●使用例

```
(*EI_Mが実行されるまで割込みプログラムの実行を禁止します。　　*)
  DI_M (TRUE);
```

●対応するMELSEC命令

・DI（割込み禁止）

5.8.2　割込許可　EI_M

DI_M実行時の割込み禁止状態を解除し，IMASKによって許可された割込みポインタ番号の割込みプログラムの実行を許可します。

■関数定義　　BOOL EI_M (BOOL EN);

引数名	IN/OUT	内容
EN	IN	実行条件 （常に有効を示す値TRUEまたは常時ONデバイスSM400のみ指定可能。）

戻り値	内容
BOOL	実行条件（常時TRUE）

●使用例

```
(*DI_M実行時の割込み禁止状態を解除します。　　*)
  EI_M (TRUE);
```

●対応するMELSEC命令

・EI（割込み許可）

5 - 37

5.9　I/Oリフレッシュ

5.9.1　I/Oリフレッシュ　RFS_M

指定されたデバイスからｎ点分のI/Oデバイスをリフレッシュします。

■関数定義　　**BOOL** RFS_M (BOOL EN, BOOL S1, ANY16 n);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	リフレッシュするデバイス（ビットデータ）
n	IN	リフレッシュするデータ数（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件M0がONすると，X100から32点分のデバイスをリフレッシュします。　*)
  RFS_M (M0, X100, H20);
```

●対応するMELSEC命令

・RFS（I/Oリフレッシュ）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 38

5.10　論理演算命令

5.10.1　論理積(2デバイス)　WAND_M

指定された2つのBIN16ビットデータをビットごとに論理積演算します。

■関数定義

```
BOOL WAND_M (BOOL EN, ANY16 S1, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理積演算するデータ（BIN16ビットデータ）
D	IN/OUT	論理積演算されるデータ・演算結果（BIN16ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0 とD10の16ビットデータをビットごとに論理積を行い，*)
(*その結果をD10に格納します。                                    *)
  WAND_M (X0, D0, D10);
```

●対応するMELSEC命令

・WAND（16ビットデータ論理積）

5.10.2　論理積(3デバイス)　WAND_3_M

指定された2つのBIN16ビットデータをビットごとに論理積演算します。

■関数定義

```
BOOL WAND_3_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D1);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理積演算されるデータ（BIN16ビットデータ）
S2	IN	論理積演算するデータ（BIN16ビットデータ）
D1	OUT	演算結果（BIN16ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0 とD10の16ビットデータをビットごとに論理積を行い，*)
(*その結果をD100に格納します。                                   *)
  WAND_3_M (X0, D0, D10, D100);
```

●対応するMELSEC命令

・WAND（16ビットデータ論理積）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 39

5.10.3　32ビットデータ論理積(2デバイス)　DAND_M

指定された2つのBIN32ビットデータをビットごとに論理積演算します。

■関数定義　　**BOOL** DAND_M (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理積演算するデータ（BIN32ビットデータ）
D	IN/OUT	論理積演算されるデータ・演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1とX30～X47の24ビットデータの論理積を行い，*)
(*結果をdwData1に格納します。                                    *)
  DAND_M (X0, K6X30, dwData1);
```

●対応するMELSEC命令

・DAND（32ビットデータ論理積）

5.10.4　32ビットデータ論理積(3デバイス)　DAND_3_M

指定された2つのBIN32ビットデータをビットごとに論理積演算します。

■関数定義　　**BOOL** DAND_3_M (BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理積演算されるデータ（BIN32ビットデータ）
S2	IN	論理積演算するデータ（BIN32ビットデータ）
D	OUT	演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1とdwData2の32ビットデータの論理積を行い，*)
(*結果をResultに格納します。                                    *)
  DAND_3_M (X0, dwData1, dwData2, Result);
```

●対応するMELSEC命令

・DAND（32ビットデータ論理積）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 40

5.10.5　ブロックデータ論理積　BKAND_M

指定された2つのデバイスからn点分の16ビットデータをビットごとに論理積演算します。

■関数定義

```
BOOL BKAND_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理演算されるデータ（BIN16ビットデータ）
S2	IN	論理演算するデータ（BIN16ビットデータ）
n	IN	演算するデータ数（BIN16ビットデータ）
D	OUT	演算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと　*)
(*D200からD0に格納されている値の点数分のデータの論理積を行い，その結果を　*)
(*D1000以降に格納します。　*)
  BKAND_M (X0, D100, D200, D0, D1000);
```

●対応するMELSEC命令

・BKAND（ブロック論理積）

5.10.6　論理和(2デバイス)　WOR_M

指定された2つのBIN16ビットデータをビットごとに論理和演算します。

■関数定義

```
BOOL WOR_M (BOOL EN, ANY16 S1, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理和演算するデータ（BIN16ビットデータ）
D	IN/OUT	論理和演算されるデータ・演算結果（BIN16ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D10とD20のデータの論理和を行い，その結果を　*)
(*D20に格納します。　*)
  WOR_M (X0, D10, D20);
```

●対応するMELSEC命令

・WOR（16ビットデータ論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 41

5.10.7　論理和(3デバイス)　WOR_3_M

指定された2つのBIN16ビットデータをビットごとに論理和演算します。

■関数定義

```
BOOL WOR_3_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D1);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理和演算されるデータ（BIN16ビットデータ）
S2	IN	論理和演算するデータ（BIN16ビットデータ）
D1	OUT	演算結果（BIN16ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件M0がONすると，X10～X1Bのデータと D0のデータの論理和を行い，      *)
(*その結果をY10～Y1Bに出力します。                                        *)
  WOR_3_M (M0, K3X10, D0, K3Y10)
```

●対応するMELSEC命令

・WOR（16ビットデータ論理和）

5.10.8　32ビットデータ論理和(2デバイス)　DOR_M

指定された2つのBIN32ビットデータをビットごとに論理和演算します。

■関数定義

```
BOOL DOR_M (BOOL EN, ANY32 S1, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理和演算するデータ（BIN32ビットデータ）
D	IN/OUT	論理和演算されるデータ・演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1とResultのデータの論理和を行い，その結果を*)
(*Resultに出力します。                                                    *)
  DOR_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DOR（32ビットデータ論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 42

5.10.9　32ビットデータ論理和(3デバイス)　DOR_3_M

指定された2つのBIN32ビットデータをビットごとに論理和演算します。

■関数定義

```
BOOL DOR_3_M (BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理和演算されるデータ（BIN32ビットデータ）
S2	IN	論理和演算するデータ（BIN32ビットデータ）
D	OUT	演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1の32ビットデータとX20～X3Fの32ビットデータの*)
(*論理和を行い，その結果をResultに出力します。                    *)
  DOR_3_M (X0, dwData1, K8X20, Result);
```

●対応するMELSEC命令

・DOR（32ビットデータ論理和）

5.10.10　ブロックデータ論理和　BKOR_M

指定された2つのデバイスからn点分の16ビットデータをビットごとに論理和演算します。

■関数定義

```
BOOL BKOR_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理和演算されるデータ（BIN16ビットデータ）
S2	IN	論理和演算するデータ（BIN16ビットデータ）
n	IN	演算するデータ数（BIN16ビットデータ）
D	OUT	演算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D10からD0に格納されている値の点数分のデータと    *)
(*D20からD0に格納されている値の点数分のデータの論理和を行い，その結果を *)
(*D100以降に格納します。                                          *)
  BKOR_M (X0, D10, D20, D0, D100)
```

●対応するMELSEC命令

・BKOR（ブロック論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 43

5.10.11　排他的論理和(2デバイス)　WXOR_M

指定された2つのBIN16ビットデータをビットごとに排他的論理和演算します。

■関数定義　　BOOL WXOR_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	排他的論理和演算するデータ（BIN16ビットデータ）
D	IN/OUT	排他的論理和演算されるデータ・演算結果（BIN16ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D10とD20の16ビットデータの排他的論理積を行い，　*)
(*その結果をD20に格納します。　*)
  WXOR_M (X0, D10, D20);
```

●対応するMELSEC命令

・WXOR（16ビットデータ排他的論理和）

5.10.12　排他的論理和(3デバイス)　WXOR_3_M

指定された2つのBIN16ビットデータをビットごとに排他的論理和演算します。

■関数定義　　BOOL WXOR_3_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	排他的論理和演算されるデータ（BIN16ビットデータ）
S2	IN	排他的論理和演算するデータ（BIN16ビットデータ）
D	OUT	演算結果（BIN16ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D10とD20の16ビットデータの排他的論理積を行い，　*)
(*その結果をD100に格納します。　*)
  WXOR_3_M (X0, D10, D20, D100);
```

●対応するMELSEC命令

・WXOR（16ビットデータ排他的論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 44

5.10.13　32ビットデータ排他的論理和(2デバイス)　DXOR_M

指定された2つのBIN32ビットデータをビットごとに排他的論理和演算します。

■関数定義

```
BOOL DXOR_M (BOOL EN, ANY32 S1, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	排他的論理和演算するデータ（BIN32ビットデータ）
D	IN/OUT	排他的論理和演算されるデータ・演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,dwData1とResultの32ビットデータの排他的論理和を行い,*)
(*その結果をResultに格納します。                                    *)
  DXOR_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DXOR（32ビットデータ排他的論理和）

5.10.14　32ビットデータ排他的論理和(3デバイス)　DXOR_3_M

指定された2つのBIN32ビットデータをビットごとに排他的論理和演算します。

■関数定義

```
BOOL DXOR_3_M (BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	排他的論理和演算されるデータ（BIN32ビットデータ）
S2	IN	排他的論理和演算するデータ（BIN32ビットデータ）
D	OUT	演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,dwData1とdwData2の32ビットデータの排他的論理和を行い,*)
(*その結果を Resultへ格納します。                                   *)
  DXOR_3_M (X0, dwData1, dwData2, Result);
```

●対応するMELSEC命令

・DXOR（32ビットデータ排他的論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 45

5.10.15　ブロックデータ排他的論理和　BKXOR_M

指定された2つのデバイスからn点分の16ビットデータをビットごとに排他的論理和演算します。

■関数定義

BOOL BKXOR_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理演算されるデータ（BIN16ビットデータ）
S2	IN	論理演算するデータ（BIN16ビットデータ）
n	IN	演算するデータ数（BIN16ビットデータ）
D	OUT	演算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D10からD0に格納されている値の点数分のデータと　*)
(*D20からD0に格納されている値の点数分のデータの排他的論理和を行い，その結果*)
(*をD100以降に格納します。　*)
  BKXOR_M (X0, D10, D20, D0, D100);
```

●対応するMELSEC命令

・BKXOR（ブロック排他的論理和）

5.10.16　否定排他的論理和(2デバイス)　WXNR_M

指定された2つのBIN16ビットデータをビットごとに否定排他的論理和演算します。

■関数定義

BOOL WXNR_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	否定排他的論理和演算するデータ（BIN16ビットデータ）
D	IN/OUT	否定排他的論理和演算されるデータ・演算結果（BIN16ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，X20～X2Fの16ビットデータとD10の16ビットデータの *)
(*否定排他的論理和を行い，D10に格納します。　*)
  WXNR_M (M0, K4X20, D10);
```

●対応するMELSEC命令

・WXNR（16ビットデータ否定排他的論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 46

5.10.17　否定排他的論理和(3デバイス)　WXNR_3_M

指定された2つのBIN16ビットデータをビットごとに否定排他的論理和演算します。

■関数定義

```
BOOL WXNR_3_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	否定排他的論理和演算されるデータ（BIN16ビットデータ）
S2	IN	否定排他的論理和演算するデータ（BIN16ビットデータ）
D	OUT	演算結果（BIN16ビットデータ）

備考）引数“S1”と“D”，“S2”と“D”に同じデバイスを指定できます。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，X20～X2Fの16ビットデータとD0の16ビットデータの　*)
(*否定排他的論理和を行い，D100に格納します。　*)
  WXNR_3_M (X0, K4X20, D0, D100);
```

●対応するMELSEC命令

・WXNR（16ビットデータ否定排他的論理和）

5.10.18　32ビットデータ否定排他的論理和(2デバイス)　DXNR_M

指定された2つのBIN32ビットデータをビットごとに否定排他的論理和演算します。

■関数定義

```
BOOL DXNR_M (BOOL EN, ANY32 S1, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	否定排他的論理和演算するデータ（BIN32ビットデータ）
D	IN/OUT	否定排他的論理和演算されるデータ・演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1の32ビットデータとResultの32ビットデータの *)
(*否定排他的論理和を行い，その結果をResultに格納します。　*)
  DXNR_M (X0, dwData1, Result);
```

●対応するMELSEC命令

・DXNR（32ビットデータ否定排他的論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 47

5.10.19　32ビットデータ否定排他的論理和(3デバイス)　DXNR_3_M

指定された2つのBIN32ビットデータをビットごとに否定排他的論理和演算します。

■関数定義

```
BOOL DXNR_3_M (BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	否定排他的論理和演算されるデータ（BIN32ビットデータ）
S2	IN	否定排他的論理和演算するデータ（BIN32ビットデータ）
D	OUT	演算結果（BIN32ビットデータ）

備考）ビットデバイスの場合，桁指定以上は“0（ゼロ）”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1の32ビットデータとdwData2の32ビットデータの *)
(*否定排他的論理和を行い，その結果をResultに格納します。                    *)
  DXNR_3_M (X0, dwData1, dwData2, Result);
```

●対応するMELSEC命令

・DXNR（32ビットデータ否定排他的論理和）

5.10.20　ブロックデータ否定排他的論理和　BKXNR_M

指定された2つのデバイスからn点分の16ビットデータをビットごとに否定排他的論理和演算します。

■関数定義

```
BOOL BKXNR_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	論理演算されるデータ（BIN16ビットデータ）
S2	IN	論理演算するデータ（BIN16ビットデータ）
n	IN	演算するデータ数（BIN16ビットデータ）
D	OUT	演算結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100からD0に格納されている値の点数分のデータと，*)
(*W100からD0に格納されている値の点数分のデータの否定排他的論理和を行い,その*)
(*結果をD200以降に格納します。                                              *)
  BKXNR_M (X0, D100, W100, D0, D200);
```

●対応するMELSEC命令

・BKXNR（ブロック否定排他的論理和）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 48

5.11　ローテーション

5.11.1　右ローテーション（キャリフラグ含まない）　ROR_M

指定されたBIN16ビットデータをキャリフラグを含めないで右へnビット回転します。

■関数定義

```
BOOL ROR_M (BOOL EN, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～15)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN16ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0のデータをキャリフラグを含めないで右へ3ビット *)
(*回転します。                                                    *)
  ROR_M (X0, K3, D0);
```

●対応するMELSEC命令

・ROR（16ビットデータの右ローテション）

5.11.2　右ローテーション（キャリフラグ含む）　RCR_M

指定されたBIN16ビットデータをキャリフラグを含め右へnビット回転します。

■関数定義

```
BOOL RCR_M (BOOL EN, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～15)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN16ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0のデータをキャリフラグを含めて右へ3ビット     *)
(*回転します。                                                    *)
  RCR_M (X0, K3, D0);
```

●対応するMELSEC命令

・RCR（16ビットデータの右ローテション）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 49

5.11.3　左ローテーション（キャリフラグ含まない）　ROL_M

指定されたBIN16ビットデータをキャリフラグを含めないでnビット左へ回転します。

■関数定義　**BOOL ROL_M (BOOL EN, ANY16 n, ANY16 D);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～15)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN16ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0のデータをキャリフラグを含めないで左へ3ビット *)
(*回転します。                                                *)
  ROL_M (X0, K3, D0);
```

●対応するMELSEC命令

・ROL（16ビットデータの左ローテション）

5.11.4　左ローテーション（キャリフラグ含む）　RCL_M

指定されたBIN16ビットデータをキャリフラグを含めnビット左へ回転します。

■関数定義　**BOOL RCL_M (BOOL EN, ANY16 n, ANY16 D);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～15)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN16ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D0のデータをキャリフラグを含めて左へ3ビット     *)
(*回転します。                                                *)
  RCL_M (X0, K3, D0);
```

●対応するMELSEC命令

・RCL（16ビットデータの左ローテション）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 50

5.11.5　32ビットデータ右ローテーション（キャリフラグ含まない）　DROR_M

指定されたBIN32ビットデータをキャリフラグを含めないで右へnビット回転します。

■関数定義　　**BOOL** DROR_M (BOOL EN, ANY16 n, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～31)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN32ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,dwData1の32ビットデータをキャリフラグを含まないで,*)
(*D0に格納されている値分のビット数右へ回転します。                    *)
  DROR_M (X0, D0, dwData1);
```

●対応するMELSEC命令

・DROR（32ビットデータの右ローテーション）

5.11.6　32ビットデータ右ローテーション（キャリフラグ含む）　DRCR_M

指定されたBIN32ビットデータをキャリフラグを含め右へnビット回転します。

■関数定義　　**BOOL** DRCR_M (BOOL EN, ANY16 n, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～31)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN32ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1の32ビットデータをキャリフラグを含め      *)
(*D0に格納されている値分のビット数右へ回転します。                    *)
  DRCR_M (X0, D0, dwData1);
```

●対応するMELSEC命令

・DRCR（32ビットデータの右ローテーション）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 51

5.11.7　32ビットデータ左ローテーション（キャリフラグ含まない）　DROL_M

指定されたBIN32ビットデータをキャリフラグを含めないで左へnビット回転します。

■関数定義

```
BOOL DROL_M (BOOL EN, ANY16 n, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～31)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN32ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1の32ビットデータをキャリフラグを含まないで *)
(*左へ4ビット回転します。                                          *)
  DROL_M (X0, K4, dwData1);
```

●対応するMELSEC命令

・DROL（32ビットデータの左ローテション）

5.11.8　32ビットデータ左ローテーション（キャリフラグ含む）　DRCL_M

指定されたBIN32ビットデータをキャリフラグを含め左へnビット回転します。

■関数定義

```
BOOL DRCL_M (BOOL EN, ANY16 n, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	回転する回数(0～31)（BIN16ビットデータ）
D	IN/OUT	回転するデータ・回転結果（BIN32ビットデータ）

備考）"D"にビットデバイスを指定した場合は，指定桁数のデータで回転します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dwData1の32ビットデータをキャリフラグを含めて　*)
(*左へ4ビット回転します。                                          *)
  DRCL_M (X0, K4, dwData1);
```

●対応するMELSEC命令

・DRCL（32ビットデータの左ローテション）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 52

5.12　シフト

5.12.1　nビット右シフト　SFR_M

指定されたBIN16ビットデータを右へnビットシフトします。

■関数定義　　**BOOL** SFR_M (BOOL EN, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	シフトする回数(0～15)（BIN16ビットデータ）
D	IN/OUT	シフトするデータ・シフト結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100のデータを右へ4ビットシフトします。          *)
  SFR_M (X0, K4, D100);
```

●対応するMELSEC命令

・SFR（16ビットデータのnビット右シフト）

5.12.2　nビット左シフト　SFL_M

指定されたBIN16ビットデータを左へnビットシフトします。

■関数定義　　**BOOL** SFL_M (BOOL EN, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	シフトする回数(0～15)（BIN16ビットデータ）
D	IN/OUT	シフトするデータ・シフト結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100のデータを左へ4ビットシフトします。          *)
  SFL_M (X0, K4, D100);
```

●対応するMELSEC命令

・SFL（16ビットデータのnビット左シフト）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 53

5.12.3　nビットデータ1ビット右シフト　BSFR_M

指定されたデバイスから，n点分のビットデータを右へ1ビットシフトします。

■関数定義　　**BOOL** BSFR_M (BOOL EN, ANY16 n, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	シフトするデバイスの数（BIN16ビットデータ）
D	IN/OUT	シフトするデータ・シフト結果（ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，M100～M104のデータを右へ1ビットシフトします。　*)
 BSFR_M (X0, K5, M100);
```

●対応するMELSEC命令

・BSFR（nビットデータの1ビット右シフト）

5.12.4　nビットデータ1ビット左シフト　BSFL_M

指定されたデバイスから，n点分のビットデータを左へ1ビットシフトします。

■関数定義　　**BOOL** BSFL_M (BOOL EN, ANY16 n, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	シフトするデバイスの数（BIN16ビットデータ）
D	IN/OUT	シフトするデータ・シフト結果（ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，M100～M104のデータを左へ1ビットシフトします。　*)
 BSFL_M (X0, K5, M100);
```

●対応するMELSEC命令

・BSFL（nビットデータの1ビット左シフト）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 54

5.12.5　1ワード右シフト　DSFR_M

指定されたデバイスから，n点分の16ビットデータを右へ1ワードシフトします。

■関数定義　　**BOOL** DSFR_M (BOOL EN, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	シフトするデバイスの数（BIN16ビットデータ）
D	IN/OUT	シフトするデータ・シフト結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100～D106のデータを右へ1ワードシフトします。　*)
 DSFR_M (X0,K7, D100);
```

●対応するMELSEC命令

・DSFR（nワードデータの1ワード右シフト）

5.12.6　1ワード左シフト　DSFL_M

指定されたデバイスから，n点分の16ビットデータを左へ1ワードシフトします。

■関数定義　　**BOOL** DSFL_M (BOOL EN, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	シフトするデバイスの数（BIN16ビットデータ）
D	OUT	シフトするデータ・シフト結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100～D106のデータを左へ1ワードシフトします。　*)
 DSFL_M (X0, K7, D100);
```

●対応するMELSEC命令

・DSFL（nワードデータの1ワード左シフト）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 55

5.13　ビット処理

5.13.1　ワードデバイスのビットセット　BSET_M

指定されたワードデバイスのnビット目をセットします。

■関数定義　　**BOOL** BSET_M (BOOL EN, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	ビットセットするビット番号（BIN16ビットデータ）
D	IN/OUT	ビットセットするデータ・ビットセット結果 （BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100の8ビット目をセットします。　　*)
  BSET_M (X0, K8, D100);
```

●対応するMELSEC命令

・BSET（ワードデバイスのビットセット）

5.13.2　ワードデバイスのビットリセット　BRST_M

指定されたワードデバイスのnビット目をリセットします。

■関数定義　　**BOOL** BRST_M (BOOL EN, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n	IN	ビットリセットするビット番号（BIN16ビットデータ）
D	IN/OUT	ビットリセットするデータ・ビットリセット結果 （BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100の8ビット目をリセットします。　　*)
  BRST_M (X0, K8, D100);
```

●対応するMELSEC命令

・BRST（ワードデバイスのリセット）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 56

5.13.3　ワードデバイスのビットテスト　TEST_MD

指定されたワードデバイスの指定位置のビット状態を指定ビットデバイスに書き込みます。

■関数定義

BOOL TEST_MD (BOOL EN, ANY16 S1, ANY16 S2, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	抽出するデータ（BIN16ビットデータ）
S2	IN	抽出するビットの位置（BIN16ビットデータ）
D	OUT	抽出データ（ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100の10ビット目の状態でM0をON・OFFします。　　*)
  TEST_MD (X0, D100, K10, M0);
```

●対応するMELSEC命令

・TEST（ビットセット）

5.13.4　32ビットデータのビットテスト　DTEST_MD

指定されたBIN32ビットデータの指定位置のビットを指定ビットデバイスに書き込みます。

■関数定義

BOOL DTEST_MD (BOOL EN, ANY32 S1, ANY16 S2, BOOL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	抽出するデータ（BIN32ビットデータ）
S2	IN	抽出するビットの位置（BIN16ビットデータ）
D	OUT	抽出データ（ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dDataの10ビット目を取り出しM0に書き込む。　　*)
  DTEST_MD (X0, dData, K10, M0);
```

●対応するMELSEC命令

・DTEST（ビットセット）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 57

5.13.5　ビットデバイス一括リセット　BKRST_M

指定されたビットデバイスからn点分をリセットします。

■関数定義　　**BOOL** BKRST_M (BOOL EN, BOOL S1, ANY16 n);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	リセットするデータの先頭（ビットデータ）
n	IN	リセットするビット数（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，M10からD100に格納されている点数分リセットします。*)
  BKRST_M (X0, M10, D100);
```

●対応するMELSEC命令

・BKRST（ビットデバイスの一括リセット）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 58

5.14　データ処理

5.14.1　データサーチ　SER_M

指定されたBIN16ビットデータを検索対象として，指定されたBIN16ビットデータからn点分を検索します。

■関数定義

```
BOOL SER_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16(2) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	検索するデータ（BIN16ビットデータ）		
S2	IN	検索されるデータ（BIN16ビットデータ）		
n	IN	検索するデータ数（BIN16ビットデータ）		
D	OUT	検索結果 (ARRAY [0..1] OF ANY16)	D[0]	一致した位置
			D[1]	一致数

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100を検索対象としてD200からD300点分検索します。*)
(*検索対象と一致した個数をD[1]に，D200から何点目かの相対値をD[0]に格納　*)
(*します。　*)
  SER_M (X0, D100, D200, D300, D);
```

●対応するMELSEC命令

・SER（16ビットデータサーチ）

5.14.2　32ビットデータサーチ　DSER_M

指定されたBIN32ビットデータを検索対象として，指定されたBIN32ビットデータから2n点分を検索します。

■関数定義

```
BOOL DSER_M (BOOL EN, ANY32 S1, ANY32 S2, ANY16 n, ANY16(2) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	検索するデータ（BIN32ビットデータ）		
S2	IN	検索されるデータ（BIN32ビットデータ）		
n	IN	検索するデータ数（BIN16ビットデータ）		
D	OUT	検索結果 (ARRAY [0..1] OF ANY16)	D[0]	一致した位置
			D[1]	一致数

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dData1，dData1+1を検索対象としてdData2から32　*)
(*ビット単位でD100に格納されている点数分検索します。検索対象と一致した　*)
(*個数をArrayResult [1]に，dData2から何点目かの相対値をArrayResult[0]に　*)
(*格納します。　*)
  DSER_M (X0, dData1, dData2, D100, ArrayResult);
```

●対応するMELSEC命令

・DSER（32ビットデータサーチ）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 59

5.14.3　ビットチェック　SUM_M

指定されたBIN16ビットデータの各ビットで1になっているビット数をカウントします。

■関数定義　　**BOOL SUM_M (BOOL EN, ANY16 S1, ANY16 D);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	カウントするデータ（BIN16ビットデータ）
D	OUT	カウント結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iDataの各ビットで1になっているビット数をResultに*)
(*格納。                                                              *)
  SUM_M (X0, iData, Result);
```

●対応するMELSEC命令

・SUM（16ビットデータのビットチェック）

5.14.4　32ビットデータビットチェック　DSUM_M

指定されたBIN32ビットデータの各ビットで1になっているビット数をカウントします。

■関数定義　　**BOOL DSUM_M (BOOL EN, ANY32 S1, ANY16 D);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	カウントするデータ（BIN32ビットデータ）
D	OUT	カウント結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dDataの各ビットで1になっているビット数をResultに*)
(*格納。                                                              *)
  DSUM_M (X0, dData, Result);
```

●対応するMELSEC命令

・DSUM（32ビットデータのビットチェック）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 60

5.14.5　デコード　DECO_M

指定されたデータの下位nビットをデコードします。

■関数定義　　**BOOL** DECO_M (BOOL EN, ANY_SIMPLE S1, ANY16 n, ANY_SIMPLE D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	デコードするデータ
n	IN	有効ビット長(1～8)　※0：無処理（BIN16ビットデータ）
D	OUT	デコード結果

備考）引数“S1”,“D”にDINT/REAL/STRING型は使用できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,D100の下位BitSizeビットをデコードし,デコード結果を*)
(*Resultから2^BitSizeビットに格納します。                                *)
  DECO_M (X0, D100, BitSize, Result);
```

●対応するMELSEC命令

・DECO（8→256ビットデコード）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5.14.6　エンコード　ENCO_M

指定されたデータから2^nビットのデータをエンコードします。

■関数定義　　**BOOL** ENCO_M (BOOL EN, ANY_SIMPLE S1, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	エンコードデータ
n	IN	有効ビット長(1～8)　※0：無処理(BIN16ビットデータ)
D	OUT	エンコード結果

備考）引数“S1”にDINT/REAL/STRING型は使用できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100から2^BitSizeビットをエンコードしResultに格納。*)
  ENCO_M (X0, D100, BitSize, Result);
```

●対応するMELSEC命令

・ENCO（256→8ビットエンコード）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 61

5.14.7　7セグメントデコード　SEG_M

指定されたデータの下位4ビット(0～F)を7セグメント表示データにデコードします。

■関数定義

BOOL SEG_M (BOOL EN, ANY16 S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	デコードするデータ
D	OUT	デコード結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100の下位4ビットを7セグメント表示データに　　*)
(*デコードしResultに格納。　　*)
  SEG_M (X0, D100, Result);
```

●対応するMELSEC命令

・SEG（7セグメントデコード）

5.14.8　16ビットデータの4ビット分離　DIS_M

指定されたBIN16ビットデータの下位n桁分のデータを分離し，指定されたデバイスからn点分の下位4ビットに格納します。

■関数定義

BOOL DIS_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	分離するデータ（BIN16ビットデータ）
n	IN	分離数(1～4)　※0：無処理（BIN16ビットデータ）
D	OUT	分離結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100の下位D200桁（1桁4ビット）分のデータを　　*)
(*ResultからD200点分の下位4ビットに格納します。　　*)
  DIS_M (X0, D100, D200, Result);
```

●対応するMELSEC命令

・DIS（16ビットデータの4ビット分離）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 62

5.14.9　16ビットデータの4ビット結合　UNI_M

指定されたデバイスからn点分のBIN16ビットデータの下位4ビットを指定されたデバイスに結合します。

■関数定義　　BOOL UNI_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	結合するデータ（BIN16ビットデータ）
n	IN	結合数(1～4)　※0：無処理（BIN16ビットデータ）
D	OUT	結合結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100から3点分の16ビットデータの下位4ビットを　*)
(*Resultに結合します。　*)
  UNI_M (X0, D100, K3, Result);
```

●対応するMELSEC命令

・UNI（4ビットデータの16ビット結合）

5.14.10　任意データのビット分離　NDIS_M

指定されたデバイス以降に格納されているデータの各ビットを，指定したビット分ずつ分離します。

■関数定義　　BOOL NDIS_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	分離するデータ（BIN16ビットデータ）
S2	IN	分離単位（分離するビット数）　(BIN16ビットデータ)
D	OUT	分離結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1以降に格納しているデータの各ビットを，　*)
(*iData2ずつ分離してResult以降に格納します。　*)
  NDIS_M (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・NDIS（任意ビットデータの分離）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 63

5.14.11 任意データのビット結合 NUNI_M

指定されたデバイス以降に格納されているデータの各ビットを，指定されたビット分ずつ結合します。

■関数定義

```
BOOL NUNI_M (BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	結合するデータ（BIN16ビットデータ）
S2	IN	結合単位（結合するビット数）（BIN16ビットデータ）
D	OUT	結合結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1以降に格納しているデータの各ビットを，      *)
(*iData2ずつ結合してResult以降に格納します。                              *)
  NUNI_M (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・NUNI（任意ビットデータの結合）

5.14.12 バイト単位データ分離 WTOB_MD

指定されたデバイス以降に格納されているBIN16ビットデータをnバイトに分離します。

■関数定義

```
BOOL WTOB_MD (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	バイト単位で分離するデータ（BIN16ビットデータ）
n	IN	分離するバイトデータの個数（BIN16ビットデータ）
D	OUT	分離結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1以降に格納している16ビットデータを，      *)
(*iData2バイトに分離してResult以降に格納します。                          *)
  WTOB_MD (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・WTOB（バイト単位データの分離）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 64

5.14.13　バイト単位データ結合　BTOW_MD

指定されたデバイス以降のn点分のBIN16ビットデータの下位8ビットをワード単位に結合します。

■関数定義

```
BOOL BTOW_MD (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	バイト単位で結合するデータ（BIN16ビットデータ）
n	IN	結合するバイトデータの個数（BIN16ビットデータ）
D	OUT	結合結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1以降のiData2ワード分の16ビットデータの　*)
(*下位8ビットを，ワード単位に結合して，Result以降に格納します。　*)
  BTOW_MD (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・BTOW（バイト単位データの結合）

5.14.14　データ最大値検索　MAX_M

指定されたデバイスからn点数分のBIN16ビットデータから最大値を検索します。

■関数定義

```
BOOL MAX_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	検索するデータの先頭（BIN16ビットデータ）
n	IN	検索するデータ数（BIN16ビットデータ）
D	OUT	最大値の検索結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1以降からiData2点の16ビットBINデータ　*)
(*から最大値を検索してResultに格納します。　*)
  MAX_M (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・MAX（16ビットデータ最大値検索）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 65

5.14.15　32ビットデータ最大値検索　DMAX_M

指定されたデバイスからn点数分のBIN32ビットデータから最大値を検索します。

■関数定義　**BOOL DMAX_M (BOOL EN, ANY32 S1, ANY16 n, ANY32 D);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	検索するデータの先頭（BIN32ビットデータ）
n	IN	検索するデータ数（BIN16ビットデータ）
D	OUT	最大値の検索結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dData以降のiData点の32ビットBINデータから　　*)
(*最大値を検索してResultに格納します。　　*)
  DMAX_M (X0, dData, iData, Result);
```

●対応するMELSEC命令

・DMAX（32ビットデータ最大値検索）

5.14.16　データ最小値検索　MIN_M

指定されたデバイスからn点数分のBIN16ビットデータから最小値を検索します。

■関数定義　**BOOL MIN_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 D);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	検索するデータの先頭（BIN16ビットデータ）
n	IN	検索するデータ数（BIN16ビットデータ）
D	OUT	最小値の検索結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1以降からiData2点の16ビットBINデータから　*)
(*最小値を検索してResultに格納します。　　*)
  MIN_M (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・MIN（16ビットデータ最小値検索）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 66

5.14.17　32ビットデータ最小値検索　DMIN_M

指定されたデバイスからn点数分のBIN32ビットデータから最小値を検索します。

■関数定義　**BOOL DMIN_M (BOOL EN, ANY32 S1, ANY16 n, ANY32 D);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	検索するデータの先頭（BIN32ビットデータ）
n	IN	検索するデータ数（BIN16ビットデータ）
D	OUT	最小値の検索結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dData以降からiData点の32ビットBINデータから　　*)
(*最小値を検索してResult，Result+1に格納します。　　*)
  DMIN_M (X0, dData, iData, Result);
```

●対応するMELSEC命令

・DMIN（32ビットデータ最小値検索）

5.14.18　データソート　SORT_M

指定されたデバイスからn点数分のBIN16ビットデータを昇順・降順にソートします。

■関数定義　**BOOL SORT_M (BOOL EN, ANY16 S1, ANY16 n, ANY16 S2, BOOL D1, ANY16 D2);**

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	ソートするデータの先頭（BIN16ビットデータ）
n	IN	ソートするデータ数（BIN16ビットデータ）
S2	IN	1回の実行で比較するデータ数（BIN16ビットデータ）
D1	OUT	ソート完了でONさせるビットデバイス（ビットデータ）
D2	OUT	システム使用デバイス（BIN16ビットデータ）

備考）ソート順はSM703のON・OFFで指定します。SM703 OFF時：昇順，SM703 ON時：降順

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1からiData2点分のBIN16ビットデータを　　*)
(*昇順・降順にソートします。　　*)
  SORT_M (X0, iData1, iData2, iData3, bData, iData4);
```

●対応するMELSEC命令

・SORT（16ビットデータソート）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 67

5.14.19　32ビットデータソート　DSORT_M

指定されたデバイスからn点数分のBIN32ビットデータを昇順・降順にソートします。

■関数定義　**BOOL** DSORT_M (BOOL EN, ANY32 S1, ANY16 n, ANY16 S2, BOOL D1, ANY16 D2);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	ソートするデータの先頭（BIN32ビットデータ）
n	IN	ソートするデータ数（BIN16ビットデータ）
S2	IN	1回の実行で比較するデータ数（BIN16ビットデータ）
D1	OUT	ソート完了でONさせるビットデバイス（ビットデータ）
D2	OUT	システム使用デバイス（BIN16ビットデータ）

備考）ソート順はSM703のON・OFFで指定します。SM703 OFF時：昇順，SM703 ON時：降順

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dDataからiData1点分のBIN32ビットデータを　　*)
(*昇順・降順にソートします。　　*)
  DSORT_M (X0, dData, iData1, iData2, bData, iData3);
```

●対応するMELSEC命令

・DSORT（32ビットデータソート）

5.14.20　合計値算出　WSUN_M

指定されたデバイスからn点数分のBIN16ビットデータを全て加算します。

■関数定義　**BOOL** WSUM_M (BOOL EN, ANY16 S1, ANY16 n, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	合計値を算出するデータ（BIN16ビットデータ）
n	IN	データ数（BIN16ビットデータ）
D	OUT	合計値格納先（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1からiData2点の16ビットBINデータを全て　　*)
(*加算しResultに格納します。　　*)
  WSUM_M (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・WSUM（16ビット合計値算出）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 68

5.14.21　32ビットデータ合計値算出　DWSUN_M

指定されたデバイスからn点数分のBIN32ビットデータを全て加算します。

■関数定義

```
BOOL DWSUM_M (BOOL EN, ANY32 S1, ANY16 n, ANY16(4) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	合計値を算出するデータ（BIN32ビットデータ）		
n	IN	データ数（BIN16ビットデータ）		
D	OUT	合計値格納先 (ARRAY[0..3] OF ANY16)	D[0]	上4桁
			D[1]	↓
			D[2]	
			D[3]	下4桁

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dDataからiData点の32ビットBINデータを全て　　*)
(*加算しResultに格納します。　　*)
  DWSUM_M (X0, dData, iData, Result);
```

●対応するMELSEC命令

・DWSUM（32ビット合計値算出）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 69

5.15　構造化

5.15.1　リフレッシュ　COM_M

インテリジェント機能ユニットのI/Oリフレッシュと一般データの処理をします。

■関数定義　　BOOL COM_M (BOOL EN);

引数名	IN/OUT	内容
EN	IN	実行条件 （常に有効を示す値TRUEまたは常時ONデバイスSM400のみ指定可能。）

戻り値	内容
BOOL	実行条件

●使用例

```
(*SM755 OFF時：インテリジェント機能ユニットの自動リフレッシュと一般　　*)
(*データの処理, SM755 ON時　：一般データの処理のみします。　　*)
  COM_M (TRUE);
```

●対応するMELSEC命令

・COM（リフレッシュ命令）

5 - 70

5.16　バッファメモリアクセス

5.16.1　インテリジェント機能ユニット1ワードデータリード　FROM_M

指定されたインテリジェント機能ユニット・特殊機能ユニット内のバッファメモリの指定されたアドレスから指定点数分のデータを読み出します。

■関数定義

```
BOOL FROM_M (BOOL EN, ANY16 n1, ANY16 n2, ANY16 n3, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n1	IN	インテリジェント機能ユニット・特殊機能ユニットの先頭入力番号*1（BIN16ビットデータ）
n2	IN	読み出すデータの先頭アドレス（BIN16ビットデータ）
n3	IN	読出しデータ数（BIN16ビットデータ）
D	OUT	読出しデータ（BIN16ビットデータ）

*1：先頭入出力番号を16進数4桁で表したときの上3桁で指定します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，入出力番号040～05Fに装着されたインテリジェント機能*)
(*ユニットのバッファメモリのアドレス10からデータをD0に1ワード読み出す。 *)
  FROM_M (X0, H4, K10, K1, D0);
```

●対応するMELSEC命令

・FROM（インテリジェント機能ユニットからの1ワードデータリード）

5.16.2　インテリジェント機能ユニット2ワードデータリード　DFRO_M

指定されたインテリジェント機能ユニット・特殊機能ユニット内バッファメモリの指定されたアドレスから指定点数×2のデータを読み出します。

■関数定義

```
BOOL DFRO_M (BOOL EN, ANY16 n1, ANY16 n2, ANY16 n3, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
n1	IN	インテリジェント機能ユニット・特殊機能ユニットの先頭入力番号*1（BIN16ビットデータ）
n2	IN	読み出すデータの先頭アドレス（BIN16ビットデータ）
n3	IN	読出しデータ数（BIN16ビットデータ）
D	OUT	読出しデータ（BIN32ビットデータ）

*1：先頭入出力番号を16進数4桁で表したときの上3桁で指定します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，入出力番号040～05Fに装着されたインテリジェント  *)
(*機能ユニットのバッファメモリのアドレス602,603からデータをDwResultへ     *)
(*2ワード読み出す。                                                       *)
  DFRO_M (X0, H4, K602, K1, DwResult);
```

●対応するMELSEC命令

・DFRO（インテリジェント機能ユニットからの2ワードデータリード）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 71

5.16.3　インテリジェント機能ユニット1ワードデータライト　TO_M

指定されたデバイスから,n3点のデータを指定されたインテリジェント機能ユニット・特殊機能ユニット内バッファメモリの指定されたアドレス以降に書き込みます。

■関数定義　　**BOOL** TO_M (BOOL EN, ANY16 S1, ANY16 n1, ANY16 n2, ANY16 n3);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	書込みデータ（BIN16ビットデータ）
n1	IN	インテリジェント機能ユニット・特殊機能ユニットの先頭入力番号（BIN16ビットデータ）
n2	IN	データを書き込む先頭アドレス（BIN16ビットデータ）
n3	IN	書込みデータ数（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，入出力番号040～05Fに装着されたインテリジェント　*)
(*機能ユニットのバッファメモリのアドレス0に3を書き込む。　　*)
  TO_M (X0, K3, H4, K0, K1);
```

●対応するMELSEC命令

・TO（インテリジェント機能ユニットへの1ワードデータライト）

5.16.4　インテリジェント機能ユニット2ワードデータライト　DTO_M

指定されたデバイスから，n3×2点のデータを指定されたインテリジェント機能ユニット・特殊機能ユニット内のバッファメモリの指定されたアドレス以降に書き込みます。

■関数定義　　**BOOL** DTO_M (BOOL EN, ANY32 S1, ANY16 n1, ANY16 n2, ANY16 n3);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	書込みデータ（BIN32ビットデータ）
n1	IN	インテリジェント機能ユニット・特殊機能ユニットの先頭入力番号（BIN16ビットデータ）
n2	IN	(n3×2)点のデータを書き込む先頭アドレス（BIN16ビットデータ）
n3	IN	書込みデータ数（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，入出力番号040～05Fに装着されたインテリジェント　*)
(*機能ユニットのバッファメモリのアドレス41,42に0を書き込む。　　*)
  DTO_M (X0, K0, H4, K41, K1);
```

●対応するMELSEC命令

・DTO（インテリジェント機能ユニットへの2ワードデータライト）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 72

5.17　文字列処理

5.17.1　BIN→10進アスキー変換　BINDA_S_MD

指定されたBIN16ビットデータの10進数表現の各桁の数値をそれぞれアスキーコードデータに変換します。

■関数定義　　**BOOL** BINDA_S_MD (BOOL EN, ANY16 S1, STRING(8) D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN16ビットデータ）
D	OUT	変換結果（10進アスキーコードデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,iDataに格納されているBINデータを10進数で表現したとき*)
(*の各位の数値をそれぞれアスキーコードに変換し, sDataに格納する。　　*)
  BINDA_S_MD (X0, iData, sData);
```

●対応するMELSEC命令

・BINDA（BIN16ビット→10進アスキー変換）

5.17.2　32ビットBIN→10進アスキー変換　DBINDA_S_MD

指定されたBIN32ビットデータの10進数表現の各桁の数値をそれぞれアスキーコードデータに変換します。

■関数定義　　**BOOL** DBINDA_S_MD (BOOL EN, ANY32 S1, STRING(12) D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN32ビットデータ）
D	OUT	変換結果（10進アスキーコードデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,dDataに格納されているBINデータを10進数で表現したとき*)
(*の各位の数値をそれぞれアスキーコードに変換し, sDataに格納する。　　*)
  DBINDA_S_MD (X0, dData, sData);
```

●対応するMELSEC命令

・DBINDA（BIN32ビット→10進アスキー変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 73

5.17.3　BIN→16進アスキー変換　BINHA_S_MD

指定されたBIN16ビットデータの16進数表現の各桁の数値をそれぞれアスキーコードデータに変換します。

■関数定義　**BOOL** BINHA_S_MD (BOOL EN, ANY16 S1, STRING(6) D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN16ビットデータ）
D	OUT	変換結果（16進アスキーコードデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,iDataに格納されているBINデータを16進数で表現したとき*)
(*の各位の数値をそれぞれアスキーコードに変換し, sDataに格納する。　　*)
  BINHA_S_MD (X0, iData, sData);
```

●対応するMELSEC命令

・BINHA（BIN16ビット→16進アスキー変換）

5.17.4　32ビットBIN→16進アスキー変換　DBINHA_S_MD

指定されたBIN32ビットデータの16進数表現の各桁の数値をそれぞれアスキーコードデータに変換します。

■関数定義　**BOOL** DBINHA_S_MD (BOOL EN, ANY32 S1, STRING(10) D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN32ビットデータ）
D	OUT	変換結果（16進アスキーコードデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,dDataに格納されているBINデータを16進数で表現したとき*)
(*の各位の数値をそれぞれアスキーコードに変換し, sDataに格納する。　　*)
  DBINHA_S_MD (X0, dData, sData);
```

●対応するMELSEC命令

・DBINHA（BIN32ビット→16進アスキー変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 74

5.17.5　BCD4桁→10進アスキー変換　BCDDA_S_MD

指定されたBCD4桁データの各桁の数値をそれぞれアスキーコードに変換します。

■関数定義　　**BOOL** BCDDA_S_MD (BOOL EN, ANY16 S1, STRING(6) D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BCD4桁データ）
D	OUT	変換結果（10進アスキーコードデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,iDataに格納されているBCDデータを10進数で表現したとき*)
(*の各位の数値をそれぞれアスキーコードに変換し, sDataに格納する。　　*)
  BCDDA_S_MD (X0, iData, sData);
```

●対応するMELSEC命令

・BCDDA（BCD4桁→10進アスキー変換）

5.17.6　BCD8桁→10進アスキー変換　DBCDDA_S_MD

指定されたBCD8桁データの各桁の数値をそれぞれアスキーコードに変換します。

■関数定義　　**BOOL** DBCDDA_S_MD (BOOL EN, ANY32 S1, STRING(10) D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BCD8桁データ）
D	OUT	変換結果（10進アスキーコードデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,dDataに格納されているBCDデータを10進数で表現したとき*)
(*の各位の数値をそれぞれアスキーコードに変換し, sDataに格納する。　　*)
  DBCDDA_S_MD (X0, dData, sData);
```

●対応するMELSEC命令

・DBCDDA（BCD8桁→10進アスキー変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 75

5.17.7　10進アスキー→BIN変換　DABIN_S_MD

指定された10進アスキーコードデータをBIN16ビットデータに変換します。

■関数定義　　**BOOL** DABIN_S_MD (BOOL EN, STRING(6) S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（10進アスキーコードデータ）
D	OUT	変換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataに格納されている10進アスキーデータを　　*)
(*BIN16ビットデータに変換し，iDataに格納する。　　*)
  DABIN_S_MD (X0, sData, iData);
```

●対応するMELSEC命令

・DABIN（10進アスキー→BIN16ビット変換）

5.17.8　10進アスキー→32ビットBIN変換　DDABIN_S_MD

指定された10進アスキーコードデータをBIN32ビットデータに変換します。

■関数定義　　**BOOL** DDABIN_S_MD (BOOL EN, STRING(11) S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（10進アスキーコードデータ）
D	OUT	変換結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataに格納されている10進アスキーデータを　　*)
(*BIN32ビットデータに変換し，dDataに格納する。　　*)
  DDABIN_S_MD (X0, sData, dData);
```

●対応するMELSEC命令

・DDABIN（10進アスキー→32ビット変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 76

5.17.9　16進アスキー→BIN変換　HABIN_S_MD

指定された16進アスキーコードデータをBIN16ビットデータに変換します。

■関数定義　　BOOL HABIN_S_MD (BOOL EN, STRING(4) S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（16進アスキーデータ）
D	OUT	変換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,sDataに格納されている16進アスキーデータをBIN16ビット*)
(*データに変換し，iDataに格納する。                                *)
  HABIN_S_MD (X0, sData, iData);
```

●対応するMELSEC命令

・HABIN（16進アスキー→BIN16ビット変換）

5.17.10　16進アスキー→32ビットBIN変換　DHABIN_S_MD

指定された16進アスキーコードデータをBIN32ビットデータに変換します。

■関数定義　　BOOL DHABIN_S_MD (BOOL EN, STRING(8) S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（16進アスキーデータ）
D	OUT	変換結果（BIN32ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,sDataに格納されている16進アスキーデータをBIN32ビット*)
(*データに変換し，dDataに格納する。                                *)
  DHABIN_S_MD (X0, sData, dData);
```

●対応するMELSEC命令

・DHABIN（16進アスキー→32ビットBIN変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 77

5.17.11　10進アスキー→BCD4桁変換　DABCD_S_MD

指定された10進アスキーコードデータをBCD4桁データに変換します。

■関数定義　　**BOOL** DABCD_S_MD (BOOL EN, STRING(4) S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（10進アスキーコードデータ）
D	OUT	変換結果（BCD4桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataに格納されている10進アスキーデータを　　*)
(*BCD4桁データに変換し，iDataに格納する。　　*)
  DABCD_S_MD ( X0, sData, iData );
```

●対応するMELSEC命令

・DABCD（10進アスキー→BCD4桁変換）

5.17.12　10進アスキー→BCD8桁変換　DDABCD_S_MD

指定された10進アスキーコードデータをBCD8桁データに変換します。

■関数定義　　**BOOL** DDABCD_S_MD (BOOL EN, STRING(8) S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（10進アスキーコードデータ）
D	OUT	変換結果（BCD8桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataに格納されている10進アスキーデータを　　*)
(*BCD8桁データに変換し，dDataに格納する。　　*)
  DDABCD_S_MD (X0, sData, dData);
```

●対応するMELSEC命令

・DDABCD（10進アスキー→BCD8桁変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 78

5.17.13　デバイスのコメントデータ読出し　COMRD_S_MD

指定されたデバイスのコメントをアスキーコードデータで読み出します。

■関数定義　　BOOL COMRD_S_MD (BOOL EN, ANY_SIMPLE S1, STRING(32) D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	コメントを読み出すデータ
D	OUT	コメント読み出し結果（アスキーコードデータ）

備考）引数“S1”にDINT/REAL/STRING型は使用できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，D100に設定されているコメントを読み出し，sDataに *)
(*アスキーコードで格納する。                                          *)
  COMRD_S_MD (X0, D100, sData);
```

●対応するMELSEC命令

・COMRD（デバイスのコメントデータ読出し）

5.17.14　文字列の長さ検出　LEN_S_MD

指定された文字列の長さを求めます。

■関数定義　　BOOL LEN_S_MD (BOOL EN, STRING S1, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	文字列長を検出するデータ（文字列データ）
D	OUT	検出結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定した文字列の長さを検出しiDataに格納する。*)
  LEN_S_MD (X0, sData, iData);
```

●対応するMELSEC命令

・LEN（文字列の長さ検出）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 79

5.17.15　BIN→文字列変換　STR_S_MD

指定されたBIN16ビットデータの指定された位置に小数点を付加して文字列に変換します。

■関数定義

```
BOOL STR_S_MD (BOOL EN, ANY32 S1, ANY16 S2, STRING(9) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	変換する数値の桁数（BIN32ビットデータ）	S1	全桁数（2～8桁）
			S1+1	少数部桁数(0～5桁)
S2	IN	変換するデータ（BIN16ビットデータ）		
D	OUT	変換結果（文字列データ）		

備考）“S1”にビットデバイスの桁指定は指定できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iDataで指定したBIN16ビットデータをdDataで指定した *)
(*位置に小数点を付加して文字列に変換し，sDataに格納する。              *)
  STR_S_MD (X0, dData, iData, sData);
```

●対応するMELSEC命令

・STR（BIN16ビット→文字列変換）

5.17.16　32ビットBIN→文字列変換　DSTR_S_MD

指定されたBIN32ビットデータの指定された位置に小数点を付加して文字列に変換します。

■関数定義

```
BOOL DSTR_S_MD (BOOL EN, ANY32 S1, ANY32 S2, STRING(14) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	変換する数値の桁数（BIN32ビットデータ）	S1	全桁数（2～8桁）
			S1+1	少数部桁数(0～5桁)
S2	IN	変換するデータ（BIN32ビットデータ）		
D	OUT	変換結果（文字列データ）		

備考）“S1”にビットデバイスの桁指定は指定できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dData1で指定したBIN32ビットデータをdData2で指定した*)
(*位置に小数点を付加して文字列に変換し，sDataに格納する。              *)
  DSTR_S_MD (X0, dData1, dData2, sData);
```

●対応するMELSEC命令

・DSTR（BIN32ビット→文字列変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 80

5.17.17　文字列→BIN変換　VAL_S_MD

指定された文字列をBIN16ビットデータに変換し，その桁数とBIN16ビットデータを取得します。

■関数定義

```
BOOL VAL_S_MD (BOOL EN, STRING(8) S1, ANY32 D1, ANY16 D2);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（文字列データ） 備考）S1で指定する文字列のうち小数部となる文字数は0～5文字。 ただし，（全桁数-3）以下となるように指定すること。
D1	OUT	変換結果（桁数）（BIN32ビットデータ）
D2	OUT	変換結果（BIN16ビットデータ）

備考）“D1”にビットデバイスの桁指定は指定できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定した文字列をBIN16ビットデータに変換し，*)
(*桁数をdDataに，BINデータをiDataに格納する。                    *)
  VAL_S_MD (X0, sData, dData, iData);
```

●対応するMELSEC命令

・VAL（文字列→BIN16ビット変換）

5.17.18　文字列→32ビットBIN変換　DVAL_S_MD

指定された文字列をBIN32ビットデータに変換し，その桁数とBIN32ビットデータを取得します。
取得結果を指定されたデバイスに格納します。

■関数定義

```
BOOL DVAL_S_MD (BOOL EN, STRING(13) S1, ANY32 D1, ANY32 D2);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（文字列データ） 備考）S1で指定する文字列のうち小数部となる文字数は0～5文字。 ただし，（全桁数-3）以下となるように指定すること。
D1	OUT	変換結果（桁数）（BIN32ビットデータ）
D2	OUT	変換結果（BIN32ビットデータ）

備考）“D1”にビットデバイスの桁指定は指定できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定した文字列をBIN32ビットデータに変換し，*)
(*桁数をdData1に，BINデータをdData2に格納する。                  *)
  DVAL_S_MD (X0, sData, dData1, dData2);
```

●対応するMELSEC命令

・DVAL（文字列→BIN32ビット変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 81

5.17.19　浮動小数点→文字列変換　ESTR_M

指定された実数データを指定された表示指示にしたがって文字列に変換します。

■関数定義

BOOL ESTR_M (BOOL EN, REAL S1, ANY16(3) S2, STRING(24) D);

引数名	IN/OUT	内容	
EN	IN	実行条件（TRUE時のみ関数を実行します）	
S1	IN	変換するデータ（実数データ）	
S2	IN	変換する数値の表示指定（ARRAY［0..2］OF ANY16）	
		S2[0]	表示形式（0：小数点形式，1：指数形式）
		S2[1]	全桁数（2～24桁） 少数部桁数が“0”のとき……桁数（最大：24）≧ 2 少数部桁数が“0”以外のとき…桁数（最大：24）≧（少数部桁数+3）
		S2[2]	小数部桁数（0～7桁）
D	OUT	変換結果（文字列データ）	

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した実数データをArrayDataで　　*)
(*指定した表示指示にしたがって文字列に変換し，sDataに格納する。　　*)
  ESTR_M (X0, rData, ArrayData, sData);
```

●対応するMELSEC命令

・ESTR（浮動小数点→文字列変換）

5.17.20　文字列→浮動小数点変換　EVAL_M

指定された文字列データを実数データに変換します。

■関数定義

BOOL EVAL_M (BOOL EN, STRING(24) S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（文字列データ）
D	OUT	変換結果（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定した文字列を実数データに変換し，　　*)
(*rDataに格納する。　　*)
  EVAL_M (X0, sData, rData);
```

●対応するMELSEC命令

・EVAL（文字列→浮動小数点変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 82

5.17.21　BIN→アスキー変換　ASC_S_MD

指定されたBIN16ビットデータを16進数扱いで指定された文字数のアスキーデータに変換します。

■関数定義

```
BOOL ASC_S_MD (BOOL EN, ANY16 S1, ANY16 n, STRING D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BIN16ビットデータ）
n	IN	格納する文字数（BIN16ビットデータ）
D	OUT	変換結果（アスキーデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1で指定したBIN16ビットデータを16進数扱いで *)
(*アスキーに変換し，sDataで指定されたデバイス番号以降iData2で指定した　*)
(*文字数分の範囲に格納する。　*)
  ASC_S_MD (X0, iData1, iData2, sData);
```

●対応するMELSEC命令

・ASC（BIN16進データ→アスキー変換）

5.17.22　アスキー→BIN変換　HEX_S_MD

指定された文字数分に格納されている16進アスキーデータをBIN16ビットデータに変換します。

■関数定義

```
BOOL HEX_S_MD (BOOL EN, STRING S1, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（16進アスキーデータ）
n	IN	変換する文字数（BIN16ビットデータ）
D	OUT	変換結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定したデバイス番号以降iData1で指定した *)
(*文字数分に格納されている16進数アスキーデータをBIN値に変換してiData2に　*)
(*格納する。　*)
  HEX_S_MD (X0, sData, iData1, iData2);
```

●対応するMELSEC命令

・HEX（アスキー→BIN16進変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 83

5.17.23　文字列右側からの取出し　RIGHT_M

指定された文字列データの右（文字列の最終）からn文字分のデータを取得します。

■関数定義

```
BOOL RIGHT_M (BOOL EN, STRING S1, ANY16 n, STRING D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	取得するデータ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）
D	OUT	取得結果（n文字分の文字列データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定した文字列の右（文字列の最終）からiData*)
(*文字分のデータをResultに格納する。                                    *)
  RIGHT_M (X0, sData, iData, Result);
```

●対応するMELSEC命令

・RIGHT（文字列の右側からの取出し）

5.17.24　文字列左側からの取出し　LEFT_M

指定された文字列データの左（文字列の先頭）からn文字分のデータを取得します。

■関数定義

```
BOOL LEFT_M (BOOL EN, STRING S1, ANY16 n, STRING D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	取得するデータ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）
D	OUT	取得結果（n文字分の文字列データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定した文字列の左（文字列の先頭）からiData*)
(*文字分のデータをResultに格納する。                                    *)
  LEFT_M (X0, sData, iData, Result);
```

●対応するMELSEC命令

・LEFT（文字列の左側からの取出し）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 84

5.17.25　文字列中の任意取出し　MIDR_M

指定された文字列データのS2[0]番目から，S2[1]文字数分のデータを取得します。

■関数定義

BOOL MIDR_M (BOOL EN, STRING S1, ANY16(2) S2, STRING D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	取得するデータ（文字列データ）		
S2	IN	先頭文字の位置および取得する文字数の格納先(ARRAY [0..1] OF ANY16)	S2[0]	先頭文字の位置
			S2[1]	取得する文字数
D	OUT	取得結果（文字列データ）		

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sDataで指定した文字列の左（文字列の先頭）          *)
(*からStrArray [0]で指定された位置からStrArray [1]で指定された文字数分  *)
(*のデータをResultに格納する。                                          *)
  MIDR_M (X0, sData, StrArray, Result);
```

●対応するMELSEC命令

・MIDR（文字列中の任意取出し）

5.17.26　文字列中の任意置換　MIDW_M

指定された文字列データのS2[0]からS2[1]で指定された文字数分のデータを格納します。

■関数定義

BOOL MIDW_M (BOOL EN, STRING S1, ANY16(2) S2, STRING D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	取得するデータ（文字列データ）		
S2	IN	先頭文字の位置および取得する文字数の格納先(ARRAY [0..1] OF ANY16)	S2[0]	置換先の先頭文字の位置
			S2[1]	取得する文字数
D	IN/OUT	置換するデータ ・ 置換結果（文字列データ）		

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sData1で指定した文字列の左（文字列の先頭）          *)
(*からStrArray [1]で指定された文字分のデータをsData2に格納されている  *)
(*文字列データの左からStrArray [0]で指定された位置以降に格納する。      *)
  MIDW_M (X0, sData1, StrArray, sData2);
```

●対応するMELSEC命令

・MIDW（文字列中の任意置換え）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 85

5.17.27 文字列サーチ INSTR_M

指定された文字列データの左のn文字目から指定された文字列データを検索します。

■関数定義

```
BOOL INSTR_M (BOOL EN, STRING S1, STRING S2, ANY16 n, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	検索するデータ（文字列データ）
S2	IN	検索されるデータ（文字列データ）
n	IN	検索開始位置（左のn文字目から） （BIN16ビットデータ）
D	OUT	検索結果（S2で指定された文字列データの先頭から何文字目か）（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，sData2で指定した文字列の左（文字列の先頭）の      *)
(*iData文字目からsData1で指定した文字列を検索し，検索結果をResultに格納 *)
(*する。                                                              *)
  INSTR_M (X0, sData1, sData2, iData, Result);
```

●対応するMELSEC命令

・INSTR（文字列サーチ）

5.17.28 浮動小数点→BCD分解 EMOD_M

指定された実数データを指定された少数部桁に基づいてBCD型浮動小数点フォーマットに分解します。

■関数定義

```
BOOL EMOD_M (BOOL EN, REAL S1, ANY16 S2, ANY16(5) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	分解するデータ（実数データ）		
S2	IN	少数部桁データ（BIN16ビットデータ）		
D	OUT	BCD分解したデータの格納先 (ARRAY[0..4] OF ANY16)	D[0]	符号（正：0，負：1）
			D[1]	BCD7桁
			D[2]	
			D[3]	指数部符号（正：0，負：1）
			D[4]	BCD指数

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した実数データをiDataで指定した      *)
(*少数部桁に基づいてBCD型浮動小数点フォーマットに分解し,Resultに格納する。*)
  EMOD_M (X0, rData, iData, Result);
```

●対応するMELSEC命令

・EMOD（浮動小数点データ→BCD分解）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 86

5.17.29　BCDフォーマットデータ→浮動小数点　EREXP_M

指定されたBCD型浮動小数点フォーマットデータを指定された少数部桁に基づいて実数データに変換します。

■関数定義

```
BOOL EREXP_M (BOOL EN, ANY16 S1, ANY16 S2, REAL D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ（BCD型浮動小数点フォーマットデータ）
S2	IN	少数部桁データ（BIN16ビットデータ）
D	OUT	変換結果（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData1で指定したBCD型浮動小数点フォーマットデータを*)
(*iData2で指定した少数部桁に基づいて実数データに変換し，Resultに格納する。*)
  EREXP_M (X0, iData1, iData2, Result);
```

●対応するMELSEC命令

・EREXP（BCDフォーマットデータ→浮動小数点）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 87

5.18　特殊関数

5.18.1　浮動小数点SIN演算　SIN_E_MD

指定された角度のSIN（正弦）値を演算します。

■関数定義　　**BOOL** SIN_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	SIN（正弦）演算する角度データ（実数データ） 備考）指定する角度は，ラジアン単位（角度×π／180）で設定します。
D	OUT	演算結果（SIN値）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した角度のSIN値を計算し，Resultに　　*)
(*格納する。　　*)
  SIN_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・SIN（浮動小数点SIN演算（単精度））

5.18.2　浮動小数点COS演算　COS_E_MD

指定された角度のCOS（余弦）値を演算します。

■関数定義　　**BOOL** COS_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	COS（余弦）演算する角度データ（実数データ） 備考）指定する角度は，ラジアン単位（角度×π／180）で設定します。
D	OUT	演算結果（COS値）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した角度のCOS値を計算し，Resultに　　*)
(*格納する。　　*)
  COS_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・COS（浮動小数点COS演算（単精度））

5.18.3　浮動小数点TAN演算　TAN_E_MD

指定された角度のTAN（正接）値を演算します。

■関数定義　　**BOOL** TAN_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	TAN（正接）演算する角度データ（実数データ） 備考）指定する角度は，ラジアン単位（角度×π／180）で設定します。
D	OUT	演算結果（TAN値）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した角度のTAN値を計算し，Resultに　*)
(*格納する。　*)
  TAN_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・TAN（浮動小数点TAN演算（単精度））

5.18.4　浮動小数点SIN^{-1}演算　ASIN_E_MD

指定されたSIN値のSIN^{-1}（逆正弦）演算を行います。

■関数定義　　**BOOL** ASIN_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	演算するデータ　SIN値(-1.0～1.0)（実数データ）
D	OUT	演算結果（ラジアン単位の角度データ）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定したSIN値から角度を演算しResultに　*)
(*格納する。　*)
  ASIN_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・ASIN（浮動小数点SIN^{-1}演算（単精度））

5 - 89

5.18.5　浮動小数点COS^{-1}演算　ACOS_E_MD

指定したCOS値のCOS^{-1}（逆余弦）演算を行います。

■関数定義　　**BOOL** ACOS_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	演算するデータ　COS値(-1.0～1.0)　（実数データ）
D	OUT	演算結果（ラジアン単位の角度データ）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定したCOS値から角度を演算しResultに　*)
(*格納する。　*)
  ACOS_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・ACOS（浮動小数点COS^{-1}演算（単精度））

5.18.6　浮動小数点TAN^{-1}演算　ATAN_E_MD

指定されたTAN値のTAN^{-1}（逆正接）演算を行います。

■関数定義　　**BOOL** ATAN_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	演算するデータ　TAN値（実数データ）
D	OUT	演算結果（ラジアン単位の角度データ）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定したTAN値から角度を演算しResultに　*)
(*格納する。　*)
  ATAN_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・ATAN（浮動小数点TAN^{-1}演算（単精度））

5 - 90

5.18.7　浮動小数点角度→ラジアン　RAD_E_MD

指定された角度の大きさの単位を度単位からラジアン単位に変換します。

■関数定義　　**BOOL** RAD_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ　度単位の角度データ（実数データ）
D	OUT	変換結果（ラジアン単位）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した度単位の角度データをラジアン単位 *)
(*に変換し，Resultに格納する。                                          *)
  RAD_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・RAD（浮動小数点角度→ラジアン（単精度））

5.18.8　浮動小数点ラジアン→角度変換　DEG_E_MD

指定された角度の大きさの単位をラジアン単位から度単位に変換します。

■関数定義　　**BOOL** DEG_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変換するデータ　ラジアン値データ（実数データ）
D	OUT	変換結果（度単位）（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，角度の大きさの単位をrDataで指定したラジアン単位から*)
(*度単位に変換し，Resultに格納する。                                    *)
  DEG_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・DEG（浮動小数点ラジアン→角度（単精度））

5 - 91

5.18.9　浮動小数点平方根　SQR_E_MD

指定した値の平方根を演算します。

■関数定義　　BOOL SQR_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	演算するデータ（正の数のみ指定可能）　（実数データ）
D	OUT	演算結果（実数データ）

備考）"S1"で指定する値は，正の数のみです。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した値の平方根を演算し，Resultに格納する。*)
  SQR_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・SQR（浮動小数点平方根（単精度））

5.18.10　浮動小数点指数演算　EXP_E_MD

指定された値のeを底とした自然指数を演算します。

■関数定義　　BOOL EXP_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	演算する指数部データ（実数データ）
D	OUT	演算結果（e^{S1}）（実数データ）

備考）底（e）を“2.71828”として演算します。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataを指数とした自然指数演算を行い，Resultに     *)
(*格納する。                                                              *)
  EXP_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・EXP（浮動小数点指数演算（単精度））

5 - 92

5.18.11　浮動小数点自然対数演算　LOG_E_MD

指定した値のeを底としたときの対数（自然対数）を演算します。

■関数定義　　**BOOL** LOG_E_MD (BOOL EN, REAL S1, REAL D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	演算するデータ（正の数のみ指定可能）　（実数データ）
D	OUT	演算結果($\log_e$S1)（実数データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，rDataで指定した値のeを底としたときの　　*)
(*対数（自然対数）を演算し，Resultに格納する。　　*)
  LOG_E_MD (X0, rData, Result);
```

●対応するMELSEC命令

・LOG（浮動小数点自然対数演算（単精度））

5.18.12　乱数発生　RND_M

0～32767の乱数を発生します。

■関数定義　　**BOOL** RND_M (BOOL EN, ANY16 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
D	OUT	乱数発生結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，0～32767未満の乱数を発生し，Resultに格納する。　*)
  RND_M (X0, Result);
```

●対応するMELSEC命令

・RND（乱数発生）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 93

5.18.13　系列変更　SRND_M

指定された16ビットBINデータの内容にしたがって乱数系列を変更します。

■関数定義　　**BOOL** SRND_M (BOOL EN, ANY16 S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	乱数系列変更結果（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iDataで指定されているデバイスに格納されている　*)
(*16ビットBINデータの内容にしたがって乱数系列を変更する。　*)
  SRND_M (X0, iData);
```

●対応するMELSEC命令

・SRND（系列変更）

5.18.14　BCD4桁平方根　BSQR_MD

指定されたBCD4桁データの平方根を演算します。

■関数定義　　**BOOL** BSQR_MD (BOOL EN, ANY16 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	演算するBCD4桁データ（BIN16ビットデータ）
D	OUT	演算結果（BIN32ビットデータ）

備考）“D”にビットデバイスの桁指定は指定できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iDataで指定した値の平方根を演算し，dDataに格納する。*)
  BSQR_MD (X0, iData, dData);
```

●対応するMELSEC命令

・BSQR（BCD4桁平方根）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 94

5.18.15 BCD8桁平方根 BDSQR_MD

指定されたBCD8桁データの平方根を演算します。

■関数定義 **BOOL** BDSQR_MD (BOOL EN, ANY32 S1, ANY32 D);

引数名	IN/OUT	内容
EN	IN	実行条件 (TRUE時のみ関数を実行します)
S1	IN	演算するBCD8桁データ (BIN32ビットデータ)
D	OUT	演算結果 (BIN32ビットデータ)

備考) "D"にビットデバイスの桁指定は指定できません。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると,dDataで指定した値の平方根を演算し,Resultに格納する。*)
  BDSQR_MD (X0, dData, Result);
```

●対応するMELSEC命令

・BDSQR (BCD8桁平方根)

5.18.16 BCD型SIN演算 BSIN_MD

指定された角度のBCD4桁データをSIN (正弦) 演算します。

■関数定義 **BOOL** BSIN_MD (BOOL EN, ANY16 S1, ANY16(3) D);

引数名	IN/OUT	内容		
EN	IN	実行条件 (TRUE時のみ関数を実行します)		
S1	IN	演算するデータ (BCD4桁データ)		
D	OUT	演算結果 (ARRAY [0..2] OF ANY16)	D[0]	符号 (正 : 0, 負 : 1)
			D[1]	整数部 (BCD4桁データ)
			D[2]	小数部 (BCD4桁データ)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, iDataで指定した角度のSIN値を演算し,          *)
(*ArrayData [0]に演算結果の符号をArrayData [1]に演算結果の整数部,    *)
(*ArrayData [2]に小数部を格納する。                                  *)
  BSIN_MD (X0, iData, ArrayData);
```

●対応するMELSEC命令

・BSIN (BCD型SIN演算)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 95

5.18.17　BCD型COS演算　　BCOS_MD

指定された角度のBCD4桁データをCOS（余弦）演算します。

■関数定義

BOOL BCOS_MD (BOOL EN, ANY16 S1, ANY16(3) D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	演算するデータ（BCD4桁データ）		
D	OUT	演算結果	D[0]	符号（正：0，負：1）
		(ARRAY [0..2] OF ANY16)	D[1]	整数部（BCD4桁データ）
			D[2]	小数部（BCD4桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iDataで指定した角度のCOS値を演算しArrayData [0]  *)
(*に演算結果符号をArrayData [1]に演算結果の整数部，ArrayData [2]に小数部を *)
(*格納する。                                                              *)
  BCOS_MD (X0, iData, ArrayData);
```

●対応するMELSEC命令

・BCOS（BCD型COS演算）

5.18.18　BCD型TAN演算　　BTAN_MD

指定された角度のBCD4桁データをTAN（正接）演算します。

■関数定義

BOOL BTAN_MD (BOOL EN, ANY16 S1, ANY16(3) D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	演算するデータ（BCD4桁データ）		
D	OUT	演算結果	D[0]	符号（正：0，負：1）
		(ARRAY [0..2] OF ANY16)	D[1]	整数部（BCD4桁データ）
			D[2]	小数部（BCD4桁データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iDataで指定した角度のTAN値を演算し，             *)
(*ArrayData [0]に演算結果の符号をArrayData [1]に演算結果の整数部，       *)
(*ArrayData [2]に小数部を格納する。                                       *)
  BTAN_MD (X0, iData, ArrayData);
```

●対応するMELSEC命令

・BTAN（BCD型TAN演算）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 96

5.18.19 BCD型SIN^{-1}演算 BASIN_MD

指定されたBCD値のSIN^{-1}（逆正弦）値を演算します。

■関数定義

BOOL BASIN_MD (BOOL EN, ANY16(3) S1, ANY16 D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	演算するデータ (ARRAY [0..2] OF ANY16)	S[0]	符号（正：0，負：1）
			S[1]	整数部（BCD4桁データ）
			S[2]	小数部（BCD4桁データ）
D	OUT	演算結果（デバイスの先頭番号）（BCD4桁データ）		

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，BasinArrayDataで指定した値のSIN-1値を演算し，    *)
(*Resultに格納する。                                                    *)
  BASIN_MD (X0, BasinArrayData , Result);
```

●対応するMELSEC命令

・BASIN（BCD型SIN^{-1}演算）

5.18.20 BCD型COS^{-1}演算 BACOS_MD

指定されたBCD値のCOS^{-1}（逆余弦）値を演算します。

■関数定義

BOOL BACOS_MD (BOOL EN, ANY16(3) S1, ANY16 D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	COS^{-1}（逆余弦）演算するデータ (ARRAY [0..2] OF ANY16)	S[0]	符号（正：0，負：1）
			S[1]	整数部（BCD4桁データ）
			S[2]	小数部（BCD4桁データ）
D	OUT	演算結果（デバイスの先頭番号）（BCD4桁データ）		

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，BacosArrayDataで指定した値のCOS-1値を演算し，    *)
(*Resultに格納する。                                                    *)
  BACOS_MD (X0, BacosArrayData, Result);
```

●対応するMELSEC命令

・BACOS（BCD型COS^{-1}演算）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 97

5.18.21　BCD型TAN^{-1}演算　BATAN_MD

指定されたBCD値のTAN^{-1}（逆正接）値を演算します。

■関数定義

```
BOOL BATAN_MD (BOOL EN, ANY16(3) S1, ANY16 D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	演算するデータの格納されているデバイスの先頭番号（ARRAY［0..2］OF ANY16）	S[0]	符号（正：0，負：1）
			S[1]	整数部（BCD4桁データ）
			S[2]	小数部（BCD4桁データ）
D	OUT	演算結果（BCD4桁データ）		

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，BatanArrayDataで指定した値のTAN-1値を演算し，　　*)
(*Resultに格納する。　　*)
  BATAN_MD (X0, BatanArrayData, Result);
```

●対応するMELSEC命令

・BATAN（BCD型TAN^{-1}演算）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 98

5.19　データ制御

5.19.1　上下限リミット制御　LIMIT_MD

指定されたBIN16ビットデータが上下限リミット値の範囲内か否かにより，出力値を制御します。

■関数定義

```
BOOL LIMIT_MD (BOOL EN, ANY16 S1, ANY16 S2, ANY16 S3, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	下限リミット値（BIN16ビットデータ）
S2	IN	上限リミット値（BIN16ビットデータ）
S3	IN	入力値（BIN16ビットデータ）
D	OUT	出力値（BIN16ビットデータ）

備考）出力値は次に示すように制御されます。
S1（下限リミット値）＞S3（入力値）のとき・・・・・・・・S1（下限リミット値）→ D（出力値）
S2（上限リミット値）＜S3（入力値）のとき・・・・・・・S2（上限リミット値）→ D（出力値）
S1（下限リミット値）≦S3（入力値）≦ S2（上限リミット値）のとき
・・・・・・S3（入力値）→ D（出力値）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData3で指定した入力値が，iData1，iData2で指定した*)
(*上下限リミット値の範囲内か否かにより，Resultに出力値を格納する。　　*)
  LIMIT_MD (X0, iData1, iData2, iData3, Result);
```

●対応するMELSEC命令

・LIMIT（16ビット上下限リミット制御）

5.19.2　32ビットデータ上下限リミット制御　DLIMIT_MD

指定したBIN32ビットデータが上下限リミット値の範囲内か否かにより，出力値を制御します。

■関数定義

```
BOOL DLIMIT_MD (BOOL EN, ANY32 S1, ANY32 S2, ANY32 S3, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	下限リミット値（BIN32ビットデータ）
S2	IN	上限リミット値（BIN32ビットデータ）
S3	IN	入力値（BIN32ビットデータ）
D	OUT	出力値（BIN32ビットデータ）

備考）出力値は次に示すように制御されます。
S1（下限リミット値）＞S3（入力値）のとき・・・・・・・・S1（下限リミット値）→ D（出力値）
S2（上限リミット値）＜S3（入力値）のとき・・・・・・・・S2（上限リミット値）→ D（出力値）
S1（下限リミット値）≦S3（入力値）≦ S2（上限リミット値）のとき
・・・・・・S3（入力値）→ D（出力値）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dData3で指定した入力値が，dData1，dData2で指定した*)
(*上下限リミット値の範囲内か否かにより，Resultに出力値を格納する。　　*)
  DLIMIT_MD (X0, dData1, dData2, dData3, Result);
```

●対応するMELSEC命令

・DLIMIT（32ビット上下限リミット制御）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 99

5.19.3　不感帯制御　BAND_MD

指定されたBIN16ビットデータが指定した不感帯の上下限範囲内か否かにより，出力値を制御します。

■関数定義

```
BOOL BAND_MD (BOOL EN, ANY16 S1, ANY16 S2, ANY16 S3, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	不感帯の下限値データ（BIN16ビットデータ）
S2	IN	不感帯の上限値データ（BIN16ビットデータ）
S3	IN	入力値（BIN16ビットデータ）
D	OUT	出力値（BIN16ビットデータ）

備考）出力値は次に示すように制御されます。

S1（下限値）＞S3（入力値）のとき‥‥‥‥　S3（入力値）－S1（下限値）→ D（出力値）
S2（上限値）＜S3（入力値）のとき‥‥‥‥　S3（入力値）－S2（上限値）→ D（出力値）
S1（下限値）≦S3（入力値）≦ S2（上限値）のとき‥‥‥‥‥‥‥‥‥‥‥‥‥‥　0 → D（出力値）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData3で指定した入力値が，iData1，iData2で指定した*)
(*不感帯の上下限範囲内か否かにより，Resultに出力値を格納する。　　*)
  BAND_MD (X0, iData1, iData2, iData3, Result);
```

●対応するMELSEC命令

・BAND（16ビット不感帯制御）

5.19.4　32ビットデータ不感帯制御　DBAND_MD

指定されたBIN32ビットデータが，指定した不感帯の上下限範囲内か否かにより，出力値を制御します。

■関数定義

```
BOOL DBAND_MD (BOOL EN, ANY32 S1, ANY32 S2, ANY32 S3, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	不感帯の下限値データ（BIN32ビットデータ）
S2	IN	不感帯の上限値データ（BIN32ビットデータ）
S3	IN	入力値（BIN32ビットデータ）
D	OUT	出力値（BIN32ビットデータ）

備考）出力値は次に示すように制御されます。

S1（下限値）＞S3（入力値）のとき‥‥‥‥　S3（入力値）－S1（下限値）→ D（出力値）
S2（上限値）＜S3（入力値）のとき‥‥‥‥　S3（入力値）－S2（上限値）→ D（出力値）
S1（下限値）≦S3（入力値）≦ S2（上限値）のとき‥‥‥‥‥‥‥‥‥‥‥‥‥‥　0 → D（出力値）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData3で指定した入力値が，iData1，iData2で指定した*)
(*不感帯の上下限範囲内か否かにより，Resultに出力値を格納する。　　*)
  DBAND_MD (X0, dData1, dData2, dData3, Result);
```

●対応するMELSEC命令

・DBAND（32ビット不感帯制御）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 100

5.19.5　ビットゾーン制御　ZONE_MD

指定されたBIN16ビットデータにバイアス値を付加して，出力値をゾーン制御します。

■関数定義

```
BOOL ZONE_MD (BOOL EN, ANY16 S1, ANY16 S2, ANY16 S3, ANY16 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	入力値に加算する負のバイアス値（BIN16ビットデータ）
S2	IN	入力値に加算する正のバイアス値（BIN16ビットデータ）
S3	IN	入力値（IN16ビットデータ）
D	OUT	出力値（BIN16ビットデータ）

備考）出力値は次に示すように制御されます。

S3（入力値）＜0のとき・・・・・・・・・・・・S3（入力値）＋S1（負のバイアス値）→ D（出力値）
S3（入力値）＝0のとき・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・0 → D（出力値）
S3（入力値）＞0のとき・・・・・・・・・・・・S3（入力値）＋S1（正のバイアス値）→ D（出力値）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData3で指定した入力値に，iData1またはiData2で　*)
(*指定したバイアス値を付加して，Resultに格納する。　*)
  ZONE_MD (X0, iData1, iData2, iData3, Result);
```

●対応するMELSEC命令

・ZONE（16ビットゾーン制御）

5.19.6　32ビットデータビットゾーン制御　DZONE_MD

指定されたBIN32ビットデータに指定したバイアス値を付加して，出力値をゾーン制御します。

■関数定義

```
BOOL DZONE_MD (BOOL EN, ANY32 S1, ANY32 S2, ANY32 S3, ANY32 D);
```

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	入力値に加算する負のバイアス値（BIN32ビットデータ）
S2	IN	入力値に加算する正のバイアス値（BIN32ビットデータ）
S3	IN	入力値（BIN32ビットデータ）
D	OUT	出力値（BIN32ビットデータ）

備考）出力値は次に示すように制御されます。

S3（入力値）＜0のとき・・・・・・・・・・・・S3（入力値）＋S1（負のバイアス値）→ D（出力値）
S3（入力値）＝0のとき・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・0 → D（出力値）
S3（入力値）＞0のとき・・・・・・・・・・・・S3（入力値）＋S1（正のバイアス値）→ D（出力値）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，iData3で指定した入力値に，iData1またはiData2で　*)
(*指定したバイアス値を付加して，Resultに格納する。　*)
  DZONE_MD (X0, dData1, dData2, dData3, Result);
```

●対応するMELSEC命令

・DZONE（32ビットゾーン制御）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 101

5.19.7　ファイルレジスタのブロックNo.切換え　RSET_MD

プログラム中で使用するファイルレジスタのブロックNo.を指定されたブロックNo.に変更します。

■関数定義　　**BOOL** RSET_MD (BOOL EN, ANY16 S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変更するブロックNo.データ（BIN16ビットデータ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，プログラム中で使用するファイルレジスタのブロック*)
(*No.をiDataで指定されたデバイスに格納されているブロックNo.に変更する。 *)
  RSET_MD (X0, iData);
```

●対応するMELSEC命令

・RSET（ファイルレジスタのブロックNo.切換え）

5.19.8　ファイルレジスタ用ファイルのセット　QDRSET_M

プログラム中で使用するファイルレジスタのファイル名を指定されたファイル名に変更します。

■関数定義　　**BOOL** QDRSET_M (BOOL EN, STRING S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変更するファイルレジスタの“ドライブNo.：ファイル名”（文字列データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，ドライブNo.1のファイルレジスタのファイル名を     *)
(*"ABS.QDR"に変更する。                                                  *)
  QDRSET_M (X0, "1:ABC");
```

●対応するMELSEC命令

・QDRSET（ファイルレジスタ用ファイルのセット）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 102

5.19.9　コメント用ファイルのセット　QCDSET_M

プログラム中で使用するコメントファイルのファイル名を指定されたファイル名に変更します。

■関数定義　　BOOL QDRSET_M (BOOL EN, STRING S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	変更するコメントファイルの“ドライブNo.：ファイル名”（文字列データ）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，ドライブNo.3のコメントファイルのファイル名を　　*)
(*"DEF.QCD"に変更する。　　*)
  QCDSET_M (X0, "3:DEF");
```

●対応するMELSEC命令

・QCDSET（コメント用ファイルのセット）

5 - 103

5.20 時計

5.20.1 時計データの読出し DATERD_MD

QCPU/LCPUの時計素子より“年, 月, 日, 時, 分, 秒, 曜日”を読み出します。指定した格納先にBIN値で格納します。

■関数定義

BOOL DATERD_MD (BOOL EN, ANY16(7) D);

引数名	IN/OUT	内容		
EN	IN	実行条件 (TRUE時のみ関数を実行します)		
D	OUT	読み出した時計データ (ARRAY [0..6] OF ANY16)	D[0]	年 (西暦 : 1980～2079)
			D[1]	月 (1～12)
			D[2]	日 (1～31)
			D[3]	時 (0～23)
			D[4]	分 (0～59)
			D[5]	秒 (0～59)
			D[6]	曜日 (0～6)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, QCPU/LCPUの時計素子より“年, 月, 日, 時, 分,    *)
(*秒,曜日”を読み出し,TimeDataで指定されたデバイスにBIN値で格納する。    *)
  DATERD_MD (X0, TimeData);
```

●対応するMELSEC命令

・DATERD (時計データの読出し)

5.20.2 時計データの書込み DATEWR_MD

時計データ“年, 月, 日, 時, 分, 秒, 曜日”をQCPU/LCPUの時計素子に書き込みます。

■関数定義

BOOL DATEWR_MD (BOOL EN, ANY16(7) S);

引数名	IN/OUT	内容		
EN	IN	実行条件 (TRUE時のみ関数を実行します)		
S	IN	書き込む時計データ (ARRAY [0..6] OF ANY16)	S[0]	年 (西暦 : 1980～2079)
			S[1]	月 (1～12)
			S[2]	日 (1～31)
			S[3]	時 (0～23)
			S[4]	分 (0～59)
			S[5]	秒 (0～59)
			S[6]	曜日 (0～6)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, TimeDataに格納している時計データをQCPU/LCPUの    *)
(*時計素子に書き込む。                                                    *)
  DATEWR_MD (X0, TimeData);
```

●対応するMELSEC命令

・DATEWR (時計データの書込み)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 104

5.20.3 時計データの加算 DATEPLUS_M

指定された時刻データに指定された時間データを加算します。

■関数定義

```
BOOL DATEPLUS_M (BOOL EN, ANY16(3) S1, ANY16(3) S2, ANY16(3) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件 (TRUE時のみ関数を実行します)		
S1	IN	加算される時刻データ (ARRAY[0..2] OF ANY16)	S1[0]	時(0～23)
			S1[1]	分(0～59)
			S1[2]	秒(0～59)
S2	IN	加算する時間データ (ARRAY[0..2] OF ANY16)	S2[0]	時(0～23)
			S2[1]	分(0～59)
			S2[2]	秒(0～59)
D	OUT	加算結果時刻データ (ARRAY[0..2] OF ANY16)	D[0]	時(0～23)
			D[1]	分(0～59)
			D[2]	秒(0～59)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, TimeData1で指定された時刻データにTimeData2で指定*)
(*された時間データを加算し, 加算結果をResultに格納する。                *)
  DATEPLUS_M (X0, TimeData1, TimeData2, Result);
```

●対応するMELSEC命令

・DATE+ (時計データの加算)

5.20.4 時計データの減算 DATEMINUS_M

指定された時刻データから指定された時間データを減算します。

■関数定義

```
BOOL DATEMINUS_M (BOOL EN, ANY16(3) S1, ANY16(3) S2, ANY16(3) D);
```

引数名	IN/OUT	内容		
EN	IN	実行条件 (TRUE時のみ関数を実行します)		
S1	IN	減算される時刻データ (ARRAY[0..2] OF ANY16)	S1[0]	時(0～23)
			S1[1]	分(0～59)
			S1[2]	秒(0～59)
S2	IN	減算する時間データ (ARRAY[0..2] OF ANY16)	S2[0]	時(0～23)
			S2[1]	分(0～59)
			S2[2]	秒(0～59)
D	OUT	減算結果時刻データ (ARRAY[0..2] OF ANY16)	D[0]	時(0～23)
			D[1]	分(0～59)
			D[2]	秒(0～59)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると, TimeData1で指定された時刻データにTimeData2で指定*)
(*された時間データを減算し, 減算結果をResultに格納する。                *)
  DATEMINUS_M (X0, TimeData1, TimeData2, Result);
```

●対応するMELSEC命令

・DATE- (時計データの減算)

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 105

5.20.5　時計データフォーマット変換（時，分，秒→秒）　SECOND_M

指定された時間データを秒に換算します。

■関数定義　**BOOL** SECOND_M (BOOL EN, ANY16(3) S, ANY32 D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S	IN	換算する時計データ (ARRAY[0..2] OF ANY16)	S[0]	時(0～23)
			S[1]	分(0～59)
			S[2]	秒(0～59)
D	OUT	換算結果時計データ（秒）　(BIN32ビットデータ)		

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，TimeDataで指定された時間データを秒に換算して　*)
(*Resultに格納する。　*)
  SECOND_M (X0, TimeData, Result);
```

●対応するMELSEC命令

・SECOND（時計データのフォーマット変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5.20.6　時計データフォーマット変換（秒→時，分，秒）　HOUR_M

指定された秒のデータを時，分，秒に換算します。

■関数定義　BOOL HOUR_M (BOOL EN, ANY32 S1, ANY16(3) D);

引数名	IN/OUT	内容		
EN	IN	実行条件（TRUE時のみ関数を実行します）		
S1	IN	換算する時計データ（秒）　(BIN32ビットデータ)		
D	OUT	換算結果の時計データ (ARRAY[0..2] OF ANY16)	D[0]	時(0～23)
			D[1]	分(0～59)
			D[2]	秒(0～59)

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，dDataで指定された秒のデータを時，分，秒に換算して*)
(*TimeDataに格納する。　*)
  HOUR_M (X0, dData, TimeData);
```

●対応するMELSEC命令

・HOUR（時計データのフォーマット変換）

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

5 - 106

5.21　プログラム制御

5.21.1　プログラム待機　PSTOP_M

指定されたファイル名のプログラムを待機状態にします。

■関数定義　　**BOOL** PSTOP_M (BOOL EN, STRING S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	待機状態にするプログラムのファイル名 （文字列データ）

備考）待機状態にできるのは，プログラムメモリ（ドライブ番号：0）に格納されている
プログラムのみ。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，ファイル名が"ABC"のプログラムを待機状態にする。　*)
  PSTOP_M (X0, “ABC”);
```

●対応するMELSEC命令

・PSTOP（プログラム待機命令）

5.21.2　プログラム出力OFF待機　POFF_M

指定されたファイル名のプログラムを非実行にして待機状態にします。

■関数定義　　**BOOL** POFF_M (BOOL EN, STRING S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	非実行にして待機状態にするプログラムのファイル名 （文字列データ）

備考）非実行にして待機状態にできるのは，プログラムメモリ（ドライブ番号：0）に格納されているプログラムのみ。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，ファイル名が“ABC”のプログラムを非実行にして　*)
(*待機状態にする。　*)
  POFF_M (X0, sData);
```

●対応するMELSEC命令

・POFF（プログラム出力OFF待機命令）

5 - 107

5.21.3　プログラムスキャン実行登録　PSCAN_M

指定されたファイル名のプログラムをスキャン実行状態にします。

■関数定義　　BOOL PSCAN_M (BOOL EN, STRING S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	スキャン実行状態にするプログラムのファイル名 （文字列データ）

備考）スキャン実行状態にできるのは，プログラムメモリ（ドライブ番号：0）に格納されているプログラムのみ。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，ファイル名が“ABC”のプログラムをスキャン　　*)
(*実行状態にする。　　*)
  PSCAN_M (X0, sData);
```

●対応するMELSEC命令

・PSCAN（プログラムスキャン実行登録命令）

5.21.4　プログラム低速実行登録　PLOW_M

指定されたファイル名のプログラムを低速実行状態にします。

■関数定義　　BOOL PLOW_M (BOOL EN, STRING S1);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）
S1	IN	低速実行状態にするプログラムのファイル名（文字列データ）

備考）低速実行状態にできるのは，プログラムメモリ（ドライブ番号:0）に格納されているプログラムのみ。

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，ファイル名が“ABC”のプログラムを低速実行状態にする。*)
  PLOW_M (X0, “ABC”);
```

●対応するMELSEC命令

・PLOW（プログラム低速実行登録命令）

5.22　その他

5.22.1　WDTリセット　WDT_M

シーケンスプログラム中でウォッチドッグタイマのリセットを行います。

■関数定義　　**BOOL** WDT_M (BOOL EN);

引数名	IN/OUT	内容
EN	IN	実行条件（TRUE時のみ関数を実行します）

戻り値	内容
BOOL	実行条件

●使用例

```
(*実行条件X0がONすると，シーケンスプログラム中でウォッチドッグタイマの　*)
(*リセットする。　*)
  WDT_M (X0);
```

●対応するMELSEC命令

・WDT（ウォッチドッグタイマリセット）

5 - 109

６　ＩＥＣ関数

関数の見方

本書では，IEC関数の関数定義・引数・戻り値・使用例を記載しています。
IEC関数はMELSEC共通命令の組み合わせで作られています。IEC関数の使用可能デバイス，関数実行時に発生するエラー，使用可能CPUタイプは『MELSEC-Q/L プログラミングマニュアル（共通命令編）』を参照してください。参照項は，本書『**●使用例**　表内の使用命令』欄に記載されている項となります。

6.1.6　倍精度整数型(DINT)→実数型(REAL)変換　DINT_TO_REAL
DINT_TO_REAL_E

倍精度整数型(DINT)のデータを実数型(REAL)のデータに変換します。 ⟶ ①

■関数定義

REAL DINT_TO_REAL (DINT S1);
②　③　④　⑤

●引数 ⟶ ⑥

引数名	IN/OUT	内容
S1	IN	変換するデータ（BIN32ビッドデータ）

●戻り値 ⟶ ⑦

戻り値名	内容
REAL	変換結果（実数データ）

●使用例 ⟶ ⑧

引数型	STプログラム	変換結果		使用命令
DINT	r_data1 := DINT_TO_REAL(di_data1); ⑨　⑩	LD DFLT	SM400 di_data1 r_data1	LD,DFLT ⑪

① 関数の機能を示します。
② 関数のデータ型を示します。
③ 関数名を示します。
④ 引数のデータ型を示します。（STRING型の表記はSTRING（文字数）です。文字数6の場合，STRING(6)となります。）
⑤ 引数名を示します。
⑥ 関数で使用する引数の一覧表（引数名・IN/OUT・内容）を示します。
⑦ 関数で使用する戻り値の一覧表（戻り値名・内容）を示します。
⑧ 関数の使用例を示します。（実デバイス・ラベルを使用した例を示します。）
⑨ 本例はREAL型（実数型）のラベルを使用した例です。
⑩ 本例はDINT型（ダブルワード型）のラベルを使用した例です。
⑪ 関数に対応するQCPU(Qモード)/LCPU　MELSEC共通命令を示します。

6 - 1

『MELSEC-Q/L プログラミングマニュアル（共通命令編）』MELSEC命令と本書IEC関数との対応を下記に示します。

MELSEC-Q/L プログラミングマニュアル（共通命令編）『MELSEC命令』

6.3.3 BIN16 ビット /32 ビットデータ→浮動小数点変換（単精度）→①(FLT(P),DFLT(P))

Ver. Basic　High performance　Process　Redundant　Universal　LCPU →②

ベーシックモデル QCPU：シリアル No. の上 5 桁が 04122 以降

□は，FLT/DFLTを示す。

Ⓢ：32 ビット浮動小数点データに変換する整数データまたは，整数データが格納されている先頭デバイス番号（BIN16/32 ビット）

Ⓓ：変換した 32 ビット浮動小数点データを格納する先頭デバイス番号（実数）

設定データ	内部デバイス ビット	内部デバイス ワード	R, ZR	J□¥□ ビット	J□¥□ ワード	U□¥G□	Zn	定数 K, H	その他
Ⓢ	○	○		○		○	○	○	—
Ⓓ	—	○		—		○	○*1	—	—

→③

＊1：ユニバーサルモデル QCPU，LCPU で使用できます。

本書『MELSEC関数』

●使用例

引数型	STプログラム	変換結果		使用命令
DINT	r_data1 := DINT_TO_REAL(di_data1);	LD	SM400	LD,DFLT ↓ ④
		DFLT	di_data1	
			r_data1	

① MELSEC命令参照先
② 使用可能CPUタイプ
　命令を使用可能なCPUタイプを示します。
③ 使用可能デバイス
④ 参照するMELSEC共通命令

6 - 2

6.1　型変換機能

6.1.1　ブール型(BOOL)→倍精度整数型(DINT)変換　BOOL_TO_DINT
BOOL_TO_DINT_E

指定されたブール型(BOOL)のデータを倍精度整数型(DINT)のデータに変換します。

■関数定義　DINT BOOL_TO_DINT(BOOL S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（ビットデータ）

●戻り値

戻り値名	内容
DINT	変換結果（BIN32ビットデータ）

備考）戻り値の最下位ビットに変換するデータ（ビットデータ）を格納します。

●使用例

引数型	STプログラム	変換結果	使用命令
BOOL	di_data1 := BOOL_TO_DINT(b_data1);	LD　b_data1 DMOV　K1 　　di_data1 LDI　b_data1 DMOV　K0 　　di_data1	LD, DMOV, LDI

■関数定義　BOOL BOOL_TO_DINT_E(BOOL EN, BOOL S1, DINT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（ビットデータ）
D1	OUT	変換結果（BIN32ビットデータ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，bDataのブール型データを倍精度整数型(DINT)に変換し，   *)
(*  Resultに格納する。                                                          *)
    M0 := BOOL_TO_DINT_E( X0, bData, Result ) ;
```


6 - 3

6.1.2　ブール型(BOOL)→整数型(INT)変換　BOOL_TO_INT
BOOL_TO_INT_E

ブール型(BOOL)のデータを整数型(INT)のデータに変換します。

■関数定義　INT BOOL_TO_INT (BOOL S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（ビットデータ）

●戻り値

戻り値名	内容
INT	変換結果（BIN16ビットデータ）

備考）戻り値の最下位ビットに変換するデータ（ビットデータ）を格納します。

●使用例

引数型	STプログラム	変換結果	使用命令
INT	D50 := BOOL_TO_INT(M100);	LD M100 MOV K1 D50 LDI M100 MOV K0 D50	LD, MOV, LDI

■関数定義　BOOL BOOL_TO_INT_E(BOOL EN, BOOL S1, INT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（ビットデータ）
D1	OUT	変換結果（BIN16ビットデータ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，bDataのブール型(BOOL)を整数型(INT)に変換し *)
(*  Resultに格納します。                                              *)
 M0 := BOOL_TO_INT_E( X0, bData, Result ) ;
```


6 - 4

6.1.3　ブール型(BOOL)→文字列型(STRING)変換　BOOL_TO_STR
BOOL_TO_STR_E

ブール型(BOOL)のデータを文字列型(STRING)のデータに変換します。

■関数定義　STRING(2) BOOL_TO_STR (BOOL S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（ビットデータ）

●戻り値

戻り値名	内容
STRING(2)	変換結果（文字列データ）

備考）変換するデータ（ビットデータ）が0の場合，戻り値は"0"になります。
変換するデータ（ビットデータ）が1の場合，戻り値は"1"になります。

●使用例

引数型	STプログラム	変換結果	使用命令
BOOL	s_ary1 := BOOL_TO_STR(b_data1);	LD　b_data1 MOV　K49 　　s_ary1 LDI　b_data1 MOV　K48 　　s_ary1	LD, MOV, LDI

■関数定義　BOOL BOOL_TO_STR_E(BOOL EN, BOOL S1, STRING(2) D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（ビットデータ）
D1	OUT	変換結果（文字列データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，bDataのブール型(BOOL)のデータを文字列型に変換し，  *)
(*  Resultに格納する。                                                          *)
    M0 := BOOL_TO_STR _E( X0, bData, Result ) ;
```


6 - 5

6.1.4　倍精度整数型(DINT)→ブール型(BOOL)変換　DINT_TO_BOOL
DINT_TO_BOOL_E

倍精度整数型(DINT)のデータをブール型(BOOL)のデータに変換します。

■関数定義　BOOL DINT_TO_BOOL (DINT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN32ビットデータ)

●戻り値

戻り値名	内容
BOOL	変換結果（ビットデータ)

備考）変換するデータ（BIN32ビットデータ）が0の場合，戻り値は"0"になります。
変換するデータ（BIN32ビットデータ）が0以外の場合，戻り値は"1"になります。

●使用例

引数型	STプログラム	変換結果	使用命令
DINT	M100 := DINT_TO_BOOL(di_data1);	LDD<>　di_data1 K0 OUT　M100	LDD<>, OUT

■関数定義　BOOL DINT_TO_BOOL _E(BOOL EN, DINT S1, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN32ビットデータ)
D1	OUT	変換結果（ビットデータ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，dDataの倍精度整数型(DINT)データをブール型(BOOL)*)
(*  に変換し，Resultに格納します。                                          *)
    M0 := DINT_TO_BOOL_E( X0, dData, Result ) ;
```


6 - 6

6.1.5　倍精度整数型(DINT)→整数型(INT)変換　DINT_TO_INT
DINT_TO_INT_E

倍精度整数型(DINT)のデータを整数型(INT)のデータに変換します。

■関数定義　INT DINT_TO_INT (DINT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN32ビットデータ)

●戻り値

戻り値名	内容
INT	変換結果 (BIN16ビットデータ)

備考)戻り値に変換するデータ(BIN32ビットデータ)の下位16ビットを格納します。
上位16ビットは切捨てとなります。

●使用例

引数型	STプログラム	変換結果		使用命令
DINT	i_data1 := DINT_TO_INT(di_data1);	LD MOV	SM400 di_data1 i_data1	LD,MOV

■関数定義　BOOL DINT_TO_INT_E(BOOL EN, DINT S1, INT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN32ビットデータ)
D1	OUT	変換結果 (BIN16ビットデータ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, dDataの倍精度整数型(DINT)のデータを整数型(INT) *)
(*  のデータに変換し, Resultに格納します。                                *)
    M0 := DINT_TO_INT_E( X0, dData, Result ) ;
```

6.1.6　倍精度整数型(DINT)→実数型(REAL)変換　DINT_TO_REAL
DINT_TO_REAL_E

倍精度整数型(DINT)のデータを実数型(REAL)のデータに変換します。

■関数定義　REAL DINT_TO_REAL (DINT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN32ビットデータ)

●戻り値

戻り値名	内容
REAL	変換結果 (実数データ)

●使用例

引数型	STプログラム	変換結果		使用命令
DINT	r_data1 := DINT_TO_REAL(di_data1);	LD DFLT	SM400 di_data1 r_data1	LD,DFLT

■関数定義　BOOL DINT_TO_REAL_E(BOOL EN, DINT S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN32ビットデータ)
D1	OUT	変換結果 (実数データ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, dDataの倍精度整数型(DINT)データを実数型(REAL)  *)
(*  データに変換し, Resultに格納します。                                  *)
    M0 := DINT_TO_REAL_E( X0, dData, Result ) ;
```


6 - 8

6.1.7　倍精度整数型(DINT)→文字列型(STRING)変換　DINT_TO_STR
DINT_TO_STR_E

倍精度整数型(DINT)のデータを文字列型(STRING)のデータに変換します。

■関数定義　STRING(12) DINT_TO_STR (DINT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN32ビットデータ)

●戻り値

戻り値名	内容
STRING(12)	変換結果 (文字列データ)

備考）本関数はベーシックモデルQCPUでは使用できません。

●使用例

引数型	STプログラム	変換結果	使用命令
DINT	s_ary1 := DINT_TO_STR(K65535);	LD　SM400 DBINDA　K65535 s_ary1	LD,DBINDA

■関数定義　BOOL DINT_TO_STR _E(BOOL EN, DINT S1, STRING(12) D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN32ビットデータ)
D1	OUT	変換結果 (文字列データ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, dDataの倍精度整数型(DINT)データを文字列型      *)
(*  データに変換し, Resultに格納します。                                  *)
    M0 := DINT_TO_STR_E( X0, dData, Result ) ;
```


6 - 9

6.1.8　整数型(INT)→ブール型(BOOL)変換　　INT_TO_BOOL
INT_TO_BOOL_E

整数型(INT)のデータをブール型(BOOL)のデータに変換します。

■関数定義　　BOOL INT_TO_BOOL (INT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN16ビットデータ)

●戻り値

戻り値名	内容
BOOL	変換結果（ビットデータ)

備考）変換するデータ（BIN16ビットデータ）が0の場合，戻り値は"0"になります。
変換するデータ（BIN16ビットデータ）が0以外の場合，戻り値は"1"になります。

●使用例

引数型	STプログラム	変換結果	使用命令
INT	b_data1 := INT_TO_BOOL(i_data1);	LD<>　i_data1 K0 OUT　b_data1	LD<>,OUT

■関数定義　　BOOL INT_TO_BOOL _E(BOOL EN, INT S1, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN16ビットデータ)
D1	OUT	変換結果（ビットデータ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，iDataの整数型(INT)データをブール型(BOOL)データ *)
(*  に変換し，Resultに格納します。                                          *)
    M0 := INT_TO_BOOL_E( X0, iData, Result ) ;
```


6 - 10

6.1.9　整数型(INT)→倍精度整数型(DINT)変換　　INT_TO_DINT
INT_TO_DINT_E

整数型(INT)のデータを倍精度整数型(DINT)のデータに変換します。

■関数定義　　DINT INT_TO_DINT (INT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN16ビットデータ)

●戻り値

戻り値名	内容
DINT	変換結果 (BIN32ビットデータ)

●使用例

引数型	STプログラム	変換結果		使用命令
INT	di_data1 := INT_TO_DINT(D500);	LD DBL	SM400 D500 di_data1	LD,DBL

■関数定義　　BOOL INT_TO_DINT _E(BOOL EN, INT S1, DINT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN16ビットデータ)
D1	OUT	変換結果 (BIN32ビットデータ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, iDataの整数型(INT)データを倍精度整数型(DINT)  *)
(*  データに変換し, Resultに格納します。                                  *)
    M0 := INT_TO_DINT_E( X0, iData, Result ) ;
```

6.1.10　整数型(INT)→実数型(REAL)変換　　INT_TO_REAL
INT_TO_REAL_E

整数型(INT)のデータを実数型(REAL)のデータに変換します。

■関数定義　　REAL INT_TO_REAL (INT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN16ビットデータ)

●戻り値

戻り値名	内容
REAL	変換結果 (実数データ)

●使用例

引数型	STプログラム	変換結果	使用命令
INT	w_Real1:= INT_TO_REAL(D0);	LD　SM400 FLT　D0 w_Real1	LD,FLT

■関数定義　　BOOL INT_TO_REAL_E(BOOL EN, INT S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN16ビットデータ)
D1	OUT	変換結果 (実数データ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, iDataの整数型(INT)データを実数型(REAL)データに *)
(*  変換し, Resultに格納します。                                          *)
    M0 := INT_TO_REAL_E( X0, iData, Result ) ;
```


6 - 12

6.1.11　整数型(INT)→文字列型(STRING)変換　　INT_TO_STR
INT_TO_STR_E

整数型(INT)のデータを文字列型(STRING)のデータに変換します。

■関数定義　　STRING(8) INT_TO_STR (INT S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (BIN16ビットデータ)

●戻り値

戻り値名	内容
STRING(8)	変換結果 (文字列データ)

備考）本関数はベーシックモデルQCPUでは使用できません。

●使用例

引数型	STプログラム	変換結果	使用命令
INT	w_Str1 := INT_TO_STR(D0);	LD　SM400 BINDA D0 w_Str1	LD,BINDA

■関数定義　　BOOL INT_TO_STR _E(BOOL EN, INT S1, STRING(8) D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (BIN16ビットデータ)
D1	OUT	変換結果 (文字列データ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, iDataの整数型(INT)データを文字列型データに変換   *)
(*  し, Resultに格納します。                                                *)
     M0 := INT_TO_STR_E( X0, iData, Result ) ;
```

6.1.12　実数型(REAL)→倍精度整数型(DINT)変換　　REAL_TO_DINT
REAL_TO_DINT_E

指定された実数型(REAL)のデータを倍精度整数型(DINT)のデータに変換します。

■関数定義　　DINT REAL_TO_DINT(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (実数データ)

●戻り値

戻り値名	内容
DINT	変換結果 (BIN32ビットデータ)

●使用例

引数型	STプログラム	変換結果		使用命令
REAL	w_DWord1:= REAL_TO_DINT(w_Real 1);	LD DINT	SM400 w_Real1 w_DWord1	LD,DINT

■関数定義　　BOOL REAL_TO_DINT_E(BOOL EN, REAL S1, DINT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (実数データ)
D1	OUT	変換結果 (BIN32ビットデータ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, rDataの実数型(REAL)データを倍精度整数型(DINT)  *)
(*  データに変換し, Resultに格納します。                                  *)
     M0 := REAL_TO_DINT_E( X0, rData, Result ) ;
```


6 - 14

6.1.13　実数型(REAL)→整数型(INT)変換　　REAL_TO_INT
REAL_TO_INT_E

実数型(REAL)のデータを整数型(INT)のデータに変換します。

■関数定義　　INT REAL_TO_INT (REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ (実数データ)

●戻り値

戻り値名	内容
INT	変換結果 (BIN16ビットデータ)

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	w_Word1:= REAL_TO_INT(w_Real1);	LD　SM400 INT　w_Real1 　　w_Word1	LD,INT

■関数定義　　BOOL REAL_TO_INT_E(BOOL EN, REAL S1, INT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ (実数データ)
D1	OUT	変換結果 (BIN16ビットデータ)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, rDataの実数型(REAL)データを整数型(INT)データに *)
(*  変換し, Resultに格納します。                                          *)
    M0 := REAL_TO_INT_E( X0, rData, Result ) ;
```


6 - 15

6.1.14　実数型(REAL)→文字列型(STRING)変換　　REAL_TO_STR
REAL_TO_STR_E

実数型(REAL)のデータを文字列型のデータに変換します。

■関数定義　　STRING(14) REAL_TO_STR (REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（実数データ）

●戻り値

戻り値名	内容
STRING(14)	変換結果（文字列データ）

注）ESTR命令の表示形式は指数形式，全桁数は13，小数部桁数は5となります。

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	w_Str1:= REAL_TO_STR(w_Real1);	LD　SM400 MOV　K1 D10237 MOV　K13 D10238 MOV　K5 D10239 ESTR　w_Real1 D10237 　　w_Str1	LD,MOV,ESTR

■関数定義　　BOOL REAL_TO_STR _E(BOOL EN, REAL S1, STRING(14) D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（実数データ）
D1	OUT	変換結果（文字列データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*　実行条件X0がONすると，rDataの実数型(REAL)データを文字列型データに変換し，*)
(*　Resultに格納します。                                                    *)
    M0 := REAL_TO_STR_E( X0, rData, Result ) ;
```


6 - 16

6.1.15　文字列型(STRING)→ブール型(BOOL)変換　　STR_TO_BOOL
STR_TO_BOOL_E

文字列型(STRING)のデータをブール型(BOOL)のデータに変換します。

■関数定義　　BOOL STR_TO_BOOL (STRING(2) S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（文字列データ）

●戻り値

戻り値名	内容
BOOL	変換結果（ビットデータ）

備考）変換するデータ（文字列データ）が0の場合，戻り値は"0"になります。
変換するデータ（文字列データ）が0以外の場合，戻り値は"1"になります。

●使用例

引数型	STプログラム	変換結果	使用命令
STRING	w_Bit1:= STR_TO_BOOL(w_Str1);	LD<>　w_Str1 K48 OUT　w_Bit1	LD<>,OUT

■関数定義　　BOOL STR_TO_BOOL_E(BOOL EN, STRING(2) S1, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（文字列データ）
D1	OUT	変換結果（ビットデータ

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataの文字列型データをブール型データに変換し，*)
(*  Resultに格納します。                                                    *)
    M0 := STR_TO_BOOL _E( X0, sData, Result ) ;
```

6.1.16　文字列型(STRING)→倍精度整数型(DINT)変換　STR_TO_DINT
STR_TO_DINT_E

文字列型(STRING)データを倍精度整数型(DINT)データに変換します。

■関数定義　DINT STR_TO_DINT (STRING(12) S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（文字列データ）

●戻り値

戻り値名	内容
DINT	変換結果（BIN32ビットデータ）

備考） 本関数はベーシックモデルQCPUでは使用できません。

●使用例

引数型	STプログラム	変換結果	使用命令
STRING	w_DWord1:= STR_TO_DINT("123");	LD　SM400 DDABIN　"123" 　　w_DWord1	LD,DDABIN

■関数定義　BOOL STR_TO_DINT _E(BOOL EN, STRING(12) S1, DINT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（文字列データ）
D1	OUT	変換結果（BIN32ビットデータ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataの文字列型データを倍精度整数型(DINT)      *)
(*  データに変換し，Resultに格納します。                                  *)
     M0 := STR_TO_DINT_E( X0, sData, Result ) ;
```


6 - 18

6.1.17　文字列型(STRING)→整数型(INT)変換　　STR_TO_INT
STR_TO_INT_E

文字列型(STRING)のデータを整数型(INT)のデータに変換します。

■関数定義　　INT STR_TO_INT (STRING(6) S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（文字列データ）

●戻り値

戻り値名	内容
INT	変換結果（BIN16ビットデータ）

備考）本関数はベーシックモデルQCPUでは使用できません。

●使用例

引数型	STプログラム	変換結果	使用命令
STRING	w_Word1:= STR_TO_INT(w_Str1);	LD　SM400 DABIN w_Str1 w_Word1	LD,DABIN

■関数定義　　BOOL STR_TO_INT _E(BOOL EN, STRING(6) S1, INT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（文字列データ）
D1	OUT	変換結果（BIN16ビットデータ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataの文字列型データを整数型(INT)データに変換し，*)
(*  Resultに格納します。                                                    *)
    M0 := STR_TO_INT_E( X0, sData, Result ) ;
```


6 - 19

6.1.18　文字列型(STRING)→実数型(REAL)変換　　STR_TO_REAL
STR_TO_REAL_E

文字列型(STRING)データを実数型(REAL)データに変換します。

■関数定義　　REAL STR_TO_REAL (STRING(24) S1);

●引数

引数名	IN/OUT	内容
S1	IN	変換するデータ（文字列データ）

●戻り値

戻り値名	内容
REAL	変換結果 （実数データ）

●使用例

引数型	STプログラム	変換結果	使用命令
STRING	w_Real1:= STR_TO_REAL(w_Str1);	LD　SM400 EVAL　w_Str1 w_Real1	LD,EVAL

■関数定義　　BOOL STR_TO_REAL _E(BOOL EN, STRING(24) S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	変換するデータ（文字列データ）
D1	OUT	変換結果（実数データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataの文字列型データを実数型(REAL)データに変換し，*)
(*  Resultに格納します。                                                    *)
    M0 := STR_TO_REAL_E( X0, sData, Result ) ;
```


6 - 20

6.2　数値機能(一般関数)

6.2.1　絶対値　　ABS
ABS_E

指定されたデータの絶対値を演算します。

■関数定義　　ANY_NUM ABS (ANY_NUM S1);

●引数

引数名	IN/OUT	内容
S1	IN	絶対値を求めるデータ

●戻り値

戻り値名	内容
ANY_NUM	絶対値演算結果

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	r_data1 := ABS(r_data2); 	LD　SM400 EMOV　r_data2 　　r_data1 LDE<　r_data2 E0 E*　E-1 　　r_data2 　　r_data1	LD,EMOV, LDE<, E*
INT	D0 := ABS(D1); 	LD　SM400 MOV　D1　D0 LD<　D1　K0 NEG　D0	LD,MOV, LD<, NEG
DINT	di_data1 := ABS(di_data2); 	LD　SM400 DMOV　di_data2 　　di_data1 LDD<　di_data2 　　K0 DCML　di_data2 　　di_data1 D+　K1 　　di_data1	LD,DMOV, LDD<, DCML D+

■関数定義　　BOOL ABS_E(BOOL EN, ANY_NUM S1, ANY_NUM D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	絶対値を求めるデータ
D1	OUT	絶対値演算結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，iDataに格納されているデータの絶対値を求め，      *)
(*  Resultに格納します。                                                    *)
    M0 := ABS_E( X0, iData, Result ) ;
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 21

6.2.2　平方根　　SQRT
SQRT_E

指定されたデータの平方根を演算します。

■関数定義　　REAL SQRT (REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	平方根を求めるデータ（実数データ）

●戻り値

戻り値名	内容
REAL	平方根演算結果（実数データ）

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	r_data1 := SQRT(r_data2);	LD　SM400 SQR　r_data2 　　r_data1	LD,SQR

■関数定義　　BOOL SQRT_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	平方根を求めるデータ（実数データ）
D1	OUT	平方根演算結果（実数データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されているデータの平方根を求め，      *)
(*  Resultに格納します。                                                    *)
    M0 := SQRT_E( X0, rData, Result ) ;
```


6 - 22

6.3　数値機能(対数関数)

6.3.1　自然対数　　LN
LN_E

指定されたデータの自然対数を演算します。

■関数定義　　REAL LN(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	自然対数を求めるデータ（実数データ）

●戻り値

戻り値名	内容
REAL	自然対数演算結果（実数データ）

●使用例

引数型	STプログラム	変換結果		使用命令
REAL	r_data1 := LN(1.23456);	LD LOG	SM400 E1.23456 r_data1	LD,LOG

■関数定義　　BOOL LN_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	自然対数を求めるデータ（実数データ）
D1	OUT	自然対数演算結果（実数データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されているデータの自然対数を求め，  *)
(*  Resultに格納します。                                                  *)
    M0 := LN_E( X0, rData, Result ) ;
```


6 - 23

6.3.2　自然指数　　EXP
EXP_E

指定されたデータの自然指数を演算します。

■関数定義　　REAL EXP(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	自然指数を求めるデータ（実数データ）

●戻り値

戻り値名	内容
REAL	自然指数演算結果（実数データ）

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	r_data1 := EXP(r_data2);	LD　SM400 EXP　r_data2 r_data1	LD,EXP

■関数定義　　BOOL EXP_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	自然指数を求めるデータ（実数データ）
D1	OUT	自然指数演算結果（実数データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されているデータの自然指数を求め，  *)
(*  Resultに格納します。                                                  *)
    M0 := EXP_E( X0, rData, Result ) ;
```


6 - 24

6.4　数値機能(三角関数)

6.4.1　浮動小数点SIN演算　　SIN
SIN_E

指定された角度のSIN（正弦）値を演算します。

■関数定義　　REAL SIN(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	SIN（正弦）演算する角度データ（実数データ）

備考）指定する角度は，ラジアン単位（角度×π／180）で設定します。

●戻り値

戻り値名	内容
REAL	SIN演算結果（実数データ）

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	r_data1 := SIN(1.23456);	LD　SM400 SIN　E1.23456 　　r_data1	LD,SIN

■関数定義　　BOOL SIN_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	SIN（正弦）演算する角度データ（実数データ） **備考**）指定する角度は，ラジアン単位（角度×π／180）で設定します。
D1	OUT	SIN演算結果（実数データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されている角度データのSIN値を計算し，  *)
(*  結果をResultに格納します。                                                *)
    M0 := SIN_E( X0, rData, Result ) ;
```


6 - 25

6.4.2　浮動小数点COS演算　　COS
COS_E

指定された角度のCOS（余弦）値を演算します。

■関数定義　　REAL COS(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	COS（余弦）演算する角度データ（実数データ）

備考）指定する角度は，ラジアン単位（角度×π／180）で設定します。

●戻り値

戻り値名	内容
REAL	COS演算結果（実数データ）

●使用例

引数型	STプログラム	変換結果		使用命令
REAL	w_Real1 := COS(w_Real2);	LD COS	SM400 w_Real2 w_Real1	LD,COS

■関数定義　　BOOL COS_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	COS（余弦）演算する角度データ（実数データ） **備考**）指定する角度は，ラジアン単位（角度×π／180）で設定します。
D1	OUT	COS演算結果（実数データ）

備考）指定する角度は，ラジアン単位（角度×π／180）で設定します。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されている角度データのCOS値を計算し，*)
(*  結果をResultに格納します。                                          *)
    M0 := COS_E( X0, rData, Result ) ;
```


6 - 26

6.4.3　浮動小数点TAN演算　　TAN
TAN_E

指定された角度のTAN（正接）値を演算します。

■関数定義　　REAL TAN(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	TAN（正接）演算する角度データ（実数データ）

備考）指定する角度は，ラジアン単位（角度×π／180）で設定します。

●戻り値

戻り値名	内容
REAL	TAN演算結果（実数データ）

●使用例

引数型	STプログラム	変換結果		使用命令
REAL	w_Real1 := TAN(w_Real2);	LD TAN	SM400 w_Real2 w_Real1	LD,TAN

■関数定義　　BOOL TAN_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	TAN（正接）演算する角度データ（実数データ） **備考**）指定する角度は，ラジアン単位（角度×π／180）で設定します。
D1	OUT	TAN演算結果（実数データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されている角度のTAN値を計算し，結果を*)
(*  Resultに格納します。                                                    *)
    M0 := TAN_E( X0, rData, Result ) ;
```


6 - 27

6.4.4　浮動小数点SIN^{-1}演算　　ASIN
ASIN_E

指定されたSIN値のSIN^{-1}(逆正弦)を演算します。

■関数定義　　REAL ASIN(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	SIN^{-1}(逆正弦)演算するSIN値(-1.0～1.0)　(実数データ)

●戻り値

戻り値名	内容
REAL	SIN^{-1}演算結果 (実数データ)

備考) 本関数はベーシックモデルQCPUでは使用できません。
演算結果はラジアン単位の角度データです。

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	w_Real1 := ASIN(w_Real2);	LD　SM400 ASIN　w_Real2 w_Real1	LD,ASIN

■関数定義　　BOOL ASIN_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	SIN^{-1}(逆正弦)演算するSIN値(-1.0～1.0)　(実数データ)
D1	OUT	SIN^{-1}演算結果 (実数データ)

備考) 演算結果はラジアン単位の角度データです。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されているSIN値から角度を演算し，    *)
(*  結果をResultに格納します。                                              *)
    M0 := ASIN_E( X0, rData, Result ) ;
```

6.4.5　浮動小数点COS^{-1}演算　　ACOS
ACOS_E

指定したCOS値のCOS^{-1}(逆余弦)を演算します。

■関数定義　　REAL ACOS(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	COS^{-1}(逆余弦)演算するCOS値(-1.0～1.0)　(実数データ)

●戻り値

戻り値名	内容
REAL	COS^{-1}演算結果 (実数データ)

備考）本関数はベーシックモデルQCPUでは使用できません。
　　演算結果はラジアン単位の角度データです。

●使用例

引数型	STプログラム	変換結果		使用命令
REAL	w_Real1 := ACOS(w_Real2);	LD ACOS	SM400 w_Real2 w_Real1	LD,ACOS

■関数定義　　BOOL ACOS_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	COS^{-1}(逆余弦)演算するCOS値(-1.0～1.0)　(実数データ)
D1	OUT	COS^{-1}演算結果 (実数データ)

備考）演算結果はラジアン単位の角度データです。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されているCOS値から角度を演算し，    *)
(*  結果をResultに格納します。                                              *)
    M0 := ACOS_E( X0, rData, Result ) ;
```


6 - 29

6.4.6　浮動小数点TAN^{-1}演算　　ATAN
ATAN_E

指定されたTAN値のTAN^{-1}(逆正接)を演算します。

■関数定義　　REAL ATAN(REAL S1);

●引数

引数名	IN/OUT	内容
S1	IN	TAN^{-1}(逆正接)演算するTAN値（実数データ）

●戻り値

戻り値名	内容
REAL	TAN^{-1}演算結果　（実数データ）

備考） 本関数はベーシックモデルQCPUでは使用できません。
演算結果はラジアン単位の角度データです。

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	w_Real1 := ATAN(w_Real2);	LD　SM400 ATAN　w_Real2 w_Real1	LD,ATAN

■関数定義　　BOOL ATAN_E(BOOL EN, REAL S1, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	TAN^{-1}(逆正接)演算するTAN値（実数データ）
D1	OUT	TAN^{-1}演算結果（実数データ）

備考） 演算結果はラジアン単位の角度データです。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されているTAN値から角度を演算し，    *)
(*  結果をResultに格納します。                                            *)
    M0 := ATAN_E( X0, rData, Result ) ;
```


6 - 30

6.5　算術演算機能

6.5.1　加算　　ADD_E

指定された複数データを加算します。

■関数定義　　BOOL ADD_E(BOOL EN, ANY_NUM S1, ANY_NUM S2,.....,ANY_NUM Sn, ANY_NUM D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	加算するデータ
D1	OUT	加算演算結果

●戻り値

戻り値名	内容
BOOL	実行条件（ビットデータ）

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	b_result := ADD_E(b_select, r_data1, r_data2, r_data3);	LD　b_select E+　r_data1 　　r_data2 　　r_data3 LD　b_select OUT　b_result	LD, E+, OUT
INT	b_result := ADD_E(b_select, D10, D20, D30, D40);	LD　b_select +　D10　D20 　　D40 +　D30　D40 LD　b_select OUT　b_result	LD, +, OUT
DINT	b_result := ADD_E(b_select, di_data1, di_data2, di_data3);	LD　b_select D+　di_data1 　　di_data2 　　di_data3 LD　b_select OUT　b_result	LD,D+,OUT

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 31

6.5.2　乗算　　MUL_E

指定された複数データを乗算します。

■関数定義　　BOOL MUL_E(BOOL EN, ANY_NUM S1, ANY_NUM S2,.....,ANY_NUM Sn, ANY_NUM D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	乗算するデータ
D1	OUT	乗算演算結果

●戻り値

戻り値名	内容
BOOL	実行条件（ビットデータ）

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	b_result := MUL_E(b_select, r_data1, r_data2, r_data3);	LD　b_select E*　r_data1 　　r_data2 　　r_data3 LD　b_select OUT　b_result	LD, E*, OUT
INT	b_result := MUL_E(b_select, D10, D20, D30, D40);	LD　b_select *　D10　D20 　　D10238 *　D10238　D30 　　D10236 MOV　D10236　D40 LD　b_select OUT　b_result	LD, *, MOV, OUT
DINT	b_result := MUL_E(b_select, di_data1, di_data2, di_data3);	LD　b_select D*　di_data1 　　di_data2 　　D10236 DMOV　D10236 　　di_data3 LD　b_select OUT　b_result	LD, D*, DMOV, OUT

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 32

6.5.3　減算　　SUB_E

指定されたデータ同士を減算します。

■関数定義　　BOOL SUB_E(BOOL EN, ANY_NUM S1, ANY_NUM S2, ANY_NUM D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	減算されるデータ
S2	IN	減算するデータ
D1	OUT	減算演算結果

●戻り値

戻り値名	内容
BOOL	実行条件（ビットデータ）

●使用例

引数型	STプログラム	変換結果		使用命令
REAL	b_result := SUB_E(b_select, r_data1, r_data2, r_data3);	LD E- LD OUT	b_select r_data1 r_data2 r_data3 b_select b_result	LD, E-, OUT
INT	b_result := SUB_E(b_select, 32767, D100, i_data1);	LD - LD OUT	b_select K32767 D100 i_data1 b_select b_result	LD, -, OUT
DINT	b_result := SUB_E(b_select, di_data1, di_data2, di_data3);	LD D- LD OUT	b_select di_data1 di_data2 di_data3 b_select b_result	LD, D-, OUT

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 33

6.5.4 除算　DIV_E

指定されたデータ同士を除算します。

■関数定義　BOOL DIV_E(BOOL EN, ANY_NUM S1, ANY_NUM S2, ANY_NUM D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	除算されるデータ
S2	IN	除算するデータ
D1	OUT	除算演算結果

●戻り値

戻り値名	内容
BOOL	実行条件（ビットデータ）

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	b_result := DIV_E(b_select, r_data1, r_data2, r_data3);	LD b_select E/ r_data1 r_data2 r_data3 LD b_select OUT b_result	LD, E/, OUT
INT	b_result := DIV_E(b_select, D10, D20, D30);	LD b_select / D10 D20 D10238 MOV D10238 D30 LD b_select OUT b_result	LD, /, MOV, OUT
DINT	b_result := DIV_E(b_select, di_data1, di_data2, di_data3);	LD b_select D/ di_data1 di_data2 D10236 DMOV D10236 di_data3 LD b_select OUT b_result	LD, D/, DMOV, OUT

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 34

6.5.5　剰余　　MOD
MOD_E

指定されたデータ同士を除算し，その剰余を演算します。

■関数定義

BOOL MOD_E(BOOL EN, ANY_INT S1, ANY_INT S2, ANY_INT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	除算されるデータ
S2	IN	除算するデータ
D1	OUT	剰余演算結果)

●戻り値

戻り値名	内容
BOOL	実行条件（ビットデータ）

●使用例

引数型	STプログラム	変換結果	使用命令
INT	B100 := MOD_E(M1, D10, D20, D30);	LD　M1 /　D10　D20 　D10238 MOV　D10239　D30 LD　M1 OUT　B100	LD, /, MOV, OUT
DINT	b_result := MOD_E(b_select, di_data1, di_data2, di_data3);	LD　b_select D/　di_data1 　di_data2 　D10236 DMOV　D10238 　di_data3 LD　b_select OUT　b_result	LD, D/, DMOV, OUT

※MODは演算子としてのみ使用可能です。

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 35

6.5.6　指数　　EXPT
EXPT_E

指定された底とするデータと指数とするデータにより指数を演算します。

■関数定義　　REAL EXPT (REAL S1, ANY_NUM S2);

●引数

引数名	IN/OUT	内容
S1	IN	底とするデータ
S2	IN	指数とするデータ

●戻り値

戻り値名	内容
REAL	演算結果（実数データ）

●使用例

引数型	STプログラム	変換結果		使用命令
REAL	r_data1 := EXPT(r_data2, r_data3);	LD LOG E* EXP 	SM400 r_data2 r_data1 r_data1 r_data3 r_data1 r_data1 r_data1	LD, LOG, E*, EXP
INT	r_data1 := EXPT(1.123, k32767);	LD LOG FLT E* EXP 	SM400 E1.123 r_data1 K32767 D10238 r_data1 D10238 r_data1 r_data1 r_data1	LD, LOG, FLT, E*, EXP
DINT	r_data1 := EXPT(r_data2, di_data1);	LD LOG DFLT E* EXP 	SM400 r_data2 r_data1 di_data1 D10238 r_data1 D10238 r_data1 r_data1 r_data1	LD,LOG, DFLT, E*, EXP

6 - 36

■関数定義　　BOOL EXPT_E(BOOL EN, REAL S1, ANY_NUM S2, REAL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	底とするデータ
S2	IN	指数とするデータ
D1	OUT	演算結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，rDataに格納されているデータをiDataに格納されて *)
(*  いるデータで指数演算した結果をResultに格納します。                    *)
    M0 := EXPT_E( X0, rData, iData, Result ) ;
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 37

6.5.7　代入　　MOVE
MOVE_E

指定されたデータを指定された格納先に代入します。

■関数定義　　ANY MOVE (ANY S1);

●引数

引数名	IN/OUT	内容
S1	IN	代入するデータ

●戻り値

戻り値名	内容
ANY	代入結果データ

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	W_Real1:= MOVE(W_Real2);	LD　SM400 EMOV　w_Real2 w_Real1	LD,EMOV
INT	D1 :=MOVE(D0);	LD　SM400 MOV　D0　D1	LD,MOV
DINT	w_DWord1:= MOVE(2147483647);	LD　SM400 DMOV　K2147483647 w_DWord1	LD,DMOV
BOOL	w_Bit1:= MOVE(w_Bit2);	LD　SM400 MPS AND　w_Bit2 SET　w_Bit1 MRD ANI　w_Bit2 RST　w_Bit1 MPP OUT　M8191	LD,MPS,AND,SET,MRD, ANI,RST,MPP,OUT
STRING	w_Str1 := MOVE("ABCDEFG");	LD　SM400 $MOV　"ABCDEFG" w_Str1	LD,$MOV

■関数定義　　BOOL MOVE_E(BOOL EN, ANY S1, ANY D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	代入するデータ
D1	OUT	代入結果データ

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*　実行条件X0がONすると，iDataに格納されているデータをResultに格納します。*)
    M0 := MOVE_E( X0, iData, Result ) ;
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 38

6.6 ビットシフト機能

6.6.1 ビット左シフト　SHL
SHL_E

指定データをｎビット左へシフトします。

■関数定義　ANY_BIT SHL (ANY_BIT S1, ANY_BIT n);

●引数

引数名	IN/OUT	内容
S1	IN	シフトするデータ
n	IN	シフトするビット数 **備考**）シフトするビット数は定数のみ指定可能です。

●戻り値

戻り値名	内容
ANY_BIT	シフトされたデータ **備考**）最下位からｎビット分は0になります。

●使用例

引数型	STプログラム	変換結果	使用命令
INT	D0 := SHL(D1,1);	LD　SM400 MOV　D1 D0 SFL　D0 K1	LD,MOV,SFL

■関数定義　BOOL SHL_E(BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	シフトするデータ
n	IN	シフトするビット数 **備考**）シフトするビット数は定数のみ指定可能です。
D1	OUT	シフトされたデータ **備考**）最下位からｎビット分は0になります。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，D0に格納されているデータを2ビット左シフトして，*)
(*  その結果をD100に格納します。                                          *)
    M0:=SHL_E( X0, D0, 2, D100 );
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 39

6.6.2　ビット右シフト　　SHR
SHR_E

指定データを右へｎビットシフトします。

■関数定義　　ANY_BIT SHR (ANY_BIT S1, ANY_BIT n);

●引数

引数名	IN/OUT	内容
S1	IN	シフトするデータ
n	IN	シフトするビット数 **備考**）シフトするビット数は定数のみ指定可能です。

●戻り値

戻り値名	内容
ANY_BIT	シフトされたデータ **備考**）最上位からｎビット分は0になります。

●使用例

引数型	STプログラム	変換結果	使用命令
INT	D0 := SHR(D1,1);	LD　SM400 MOV　D1 D0 SFR　D0 K1	LD,MOV,SFR

■関数定義　　BOOL SHR_E(BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	シフトするデータ
n	IN	シフトするビット数 **備考**）シフトするビット数は定数のみ指定可能です。
D1	OUT	シフトされたデータ **備考**）最上位からｎビット分は0になります。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，D0に格納されているデータを右へ2ビットシフトして，*)
(*  その結果をD100に格納します。                                          *)
    M0:=SHR_E(X0, D0, 2, D100);
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 40

6.6.3　右ローテーション　　ROR
ROR_E

右側へｎビット円を描いて回転します。

■関数定義　　ANY_BIT ROR (ANY_BIT S1, ANY_BIT n);

●引数

引数名	IN/OUT	内容
S1	IN	回転するデータ
n	IN	回転するビット数 **備考**）回転するビット数は定数のみ指定可能です。

●戻り値

戻り値名	内容
ANY_BIT	回転結果データ

●使用例

引数型	STプログラム	変換結果	使用命令
INT	D0 := ROR(D1,1);	LD　SM400 MOV　D1 D0 ROR　D0 K1	LD,MOV,ROR

■関数定義　　BOOL ROR_E(BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	回転するデータ
n	IN	回転するビット数 **備考**）回転するビット数は定数のみ指定可能です。
D1	OUT	回転結果データ

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，D0に格納されているデータを右へ1ビット回転して  *)
(*  D100に格納します。                                                    *)
    M0:=ROR_E( X0, D0, 1, D100 );
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 41

6.6.4　左ローテーション　　ROL
ROL_E

左側へｎビット円を描いて回転します。

■関数定義　　ANY_BIT ROL (ANY_BIT S1, ANY_BIT n);

●引数

引数名	IN/OUT	内容
S1	IN	回転するデータ
n	IN	回転するビット数 **備考**）回転するビット数は定数のみ指定可能です。

●戻り値

戻り値名	内容
ANY_BIT	回転結果データ

●使用例

引数型	STプログラム	変換結果	使用命令
INT	D0 := ROL(D1,1);	LD　SM400 MOV　D1 D0 ROL　D0 K1	LD,MOV,ROL

■関数定義　　BOOL ROL_E(BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	回転するデータ
n	IN	回転するビット数 **備考**）回転するビット数は定数のみ指定可能です。
D1	OUT	回転結果データ

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，D0に格納されているデータを左へ1ビット回転して  *)
(*  D100に格納します。                                                    *)
    M0:=ROL_E( X0, D0, 1, D100 );
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 42

6.7　ビット型ブール機能

6.7.1　論理積　　AND_E

指定された複数データの論理積を演算します。

■関数定義　　BOOL　AND_E(BOOL EN, ANY_BIT S1, ANY_BIT S2,...., ANY_BIT Sn, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	論理積演算するデータ
D1	OUT	論理積演算結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
BOOL	b_result := AND_E(b_select, b_data1, b_data2, b_data3, b_data4);	LD　b_data1 AND　b_data2 AND　b_data3 OUT　M8191 LD　b_select AND　M8191 SET　b_data4 LD　b_select ANI　M8191 RST　b_data4 LD　b_select OUT　b_result	LD,AND,OUT,SET,ANI, RST
ワードデバイス	b_result := AND_E(b_select, d0, d1, d2, d3);	LD　b_select WAND　D0　D1 D10239 WAND　D10239　D2 D3 LD　b_select OUT　b_result	LD, WAND, OUT

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 43

6.7.2　論理和　　OR_E

指定された複数データの論理和を演算します。

■関数定義　　BOOL OR_E(BOOL EN, ANY_BIT S1, ANY_BIT S2,...., ANY_BIT Sn, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	論理和演算するデータ
D1	OUT	論理和演算結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
BOOL	b_result := OR_E(TRUE, b_data1, b_data2, b_data3);	LD　b_data1 OR　b_data2 OUT　M8191 LD　SM400 AND　M8191 SET　b_data3 LD　SM400 ANI　M8191 RST　b_data3 LD　SM400 OUT　b_result	LD,OR,OUT,AND,SET, ANI,RST
ワードデバイス	B1 := OR_E(TRUE, D0, D1, D2);	LD　SM400 WOR　D0 D1 D2 LD　SM400 OUT　B1	LD, WOR, OUT

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 44

6.7.3　排他的論理和　　XOR_E

指定された複数データの排他的論理和を演算します。

■関数定義　　BOOL XOR_E(BOOL EN, ANY_BIT S1, ANY_BIT S2,...., ANY_BIT Sn, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	排他的論理和演算するデータ
D1	OUT	排他的論理和演算結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
BOOL	b_result := XOR_E(b_select, b_data1, b_data2, b_data3);	LD　b_data1 ANI　b_data2 LDI　b_data1 AND　b_data2 ORB OUT　M8191 LD　b_select AND　M8191 SET　b_data3 LD　b_select ANI　M8191 RST　b_data3 LD　b_select OUT　b_result	LD,ANI,LDI,AND,ORB, OUT,SET,RST
ワードデバイス	b_result := XOR_E(TRUE, d0z2, d1z3, d2z4);	LD　SM400 WXOR　D0Z2　D1Z3 　　D2Z4 LD　SM400 OUT　b_result	LD,WXOR, OUT

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 45

6.7.4　論理否定　　NOT
NOT_E

指定されたデータの論理否定を演算します。

■関数定義　　ANY_BIT NOT(ANY_BIT S1);

●引数

引数名	IN/OUT	内容
S1	IN	論理否定演算するデータ

●戻り値

戻り値名	内容
ANY_BIT	論理否定演算結果

●使用例

引数型	STプログラム	変換結果	使用命令
BOOL	b_result := NOT(b_data1);	LDI　b_data1 OUT　b_result	LDI, OUT
ワードデバイス	d0z2 := NOT(d1z3);	LD　SM400 CML　D1Z3 D0Z2	LD, CML

■関数定義　　BOOL NOT_E(BOOL EN, ANY_BIT S1, ANY_BIT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	論理否定演算するデータ
D1	OUT	論理否定演算結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，D0に格納されているデータの論理否定を求め，D100 *)
(*  に格納する。                                                          *)
    M0:=NOT_E( X0, D0, D100 );
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 46

6.8　選択機能

6.8.1　バイナリの選択　　SEL
SEL_E

選択条件により指定された２つのデータから１つのデータを選択します。

■関数定義　　ANY SEL(BOOL S1, ANY S2, ANY S3);

●引数

引数名	IN/OUT	内容
S1	IN	選択条件
S2	IN	S1がFALSEの場合に選択されるデータ
S3	IN	S1がTRUEの場合に選択されるデータ

●戻り値

戻り値名	内容
ANY	選択結果 S1がFALSEの場合・・・戻り値＝S2 S1がTRUEの場合　・・・戻り値＝S3

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	r_data1 := SEL(b_select, r_data2, r_data3);	LDI　　b_select EMOV　　r_data2 　　r_data1 LD　　b_select EMOV　　r_data3 　　r_data1	LDI, EMOV, LD,
INT	D1 := SEL(X1, D2, D3);	LDI　X1 MOV　D2 D1 LD　X1 MOV　D3 D1	LDI, MOV, LD
DINT	K8X100 := SEL(X1, K8X10, K2147483647);	LDI　　X1 DMOV　　K8X10 K8X100 LD　　X1 DMOV　　K2147483647 　　K8X100	LDI, DMOV, LD
BOOL	b_result := SEL(b_select, b_data1, b_data2);	LDI　b_select MPS AND　b_data1 SET　b_result MPP ANI　b_data1 RST　b_result LD　b_select MPS AND　b_data2 SET　b_result MPP ANI　b_data2 RST　b_result	LDI, MPS,AND, SET, MPP, ANI, RST,LD
STRING	s_result := SEL(b_select, s_ary1, s_ary2);	LDI　　b_select $MOV　　s_ary1 　　s_result LD　　b_select $MOV　　s_ary2 　　s_result	LDI, $MOV,LD

6 - 47

■関数定義　　BOOL SEL_E(BOOL EN, BOOL S1, ANY S2, ANY S3, ANY D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	選択条件
S2	IN	S1がFALSEの場合に選択されるデータ
S3	IN	S1がTRUEの場合に選択されるデータ
D1	OUT	選択結果 S1がFALSEの場合・・・戻り値＝S2 S1がTRUEの場合 ・・・戻り値＝S3

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，bDataのビットデータがFALSEの場合はiData1に      *)
(*  格納されているデータをbDataのビットデータがTRUEの場合はiData2に       *)
(*  格納されているデータをResultに格納します。                            *)
    M0 := SEL_E( X0, bData, iData1, iData2, Result ) ;
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 48

6.8.2　最大値　　MAX
MAX_E

指定されたデータの中から最大値を検索します。

■関数定義　　ANY_SIMPLE MAX(ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn);

●引数

引数名	IN/OUT	内容
S1～Sn	IN	検索対象データ

●戻り値

戻り値名	内容
ANY_SIMPLE	検索結果

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	w_Real4 := MAX(w_Real1,w_Real2,w_Real3);	LD SM400 EMOV w_Real1 w_Real4 LDE< w_Real4 w_Real2 EMOV w_Real2 w_Real4 LDE< w_Real4 w_Real3 EMOV w_Real3 w_Real4	LD,EMOV,LDE<
INT	D0 := MAX(D1,D2,D3);	LD SM400 MOV D1 D0 LD< D0 D2 MOV D2 D0 LD< D0 D3 MOV D3 D0	LD,MOV,LD<
DINT	w_DWord4 := MAX(-2147483648,0,2147483647);	LD SM400 DMOV K2147483647 w_DWord4	LD,DMOV
BOOL	w_Bit4 := MAX(w_Bit1,w_Bit2,w_Bit3);	LD w_Bit1 OR w_Bit2 OR w_Bit3 OUT w_Bit4	LD,OR,OUT
STRING	w_Str4 := MAX("ABC","DEF","GHI");	LD SM400 $MOV "ABC" w_Str4 LD$< w_Str4 "DEF" $MOV "DEF" w_Str4 LD$< w_Str4 "GHI" $MOV "GHI" w_Str4	LD,MOV,LD<

6 - 49

■関数定義

BOOL MAX_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn, ANY_SIMPLE D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	検索対象データ
D1	OUT	検索結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, iData1とiData2とiData3に格納されているデータから*)
(*  最大値となるデータを検索し, Resultに格納します。                    *)
    M0 := MAX_E( X0, iData1, iData2, iData3, Result ) ;
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 50

6.8.3　最小値　　MIN
MIN_E

指定されたデータの中から最小値を検索します。

■関数定義　　ANY_SIMPLE MIN(ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn);

●引数

引数名	IN/OUT	内容
S1～Sn	IN	検索対象データ

●戻り値

戻り値名	内容
ANY_SIMPLE	検索結果

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	Real4:= MIN(Real1,Real2,Real3);	LD　SM400 EMOV　Real1 Real4 LDE>　Real4 Real2 EMOV　Real2 Real4 LDE>　Real4 Real3 EMOV　Real3 Real4	LD,EMOV,LDE>
INT	Int4:= MIN(Int1,Int2,Int3);	LD　SM400 MOV　Int1 Int4 LD>　Int4 Int2 MOV　Int2 Int4 LD>　Int4 Int3 MOV　Int3 Int4	LD,MOV,LD>
DINT	Dint4:= MIN(Dint1,Dint2,Dint3);	LD　SM400 DMOV　Dint1 Dint4 LDD>　Dint4 Dint2 DMOV　Dint2 Dint4 LDD>　Dint4 Dint3 DMOV　Dint3 Dint4	LD,DMOV,LDD>
BOOL	bBit4:= MIN(bBit1,bBit2,bBit3);	LD　bBit1 AND　bBit2 AND　bBit3 OUT　bBit4	LD,AND,OUT
STRING	Str4:= MIN(Str1,Str2,Str3);	LD　SM400 $MOV　Str1 Str4 LD$>　Str4 Str2 $MOV　Str2 Str4 LD$>　Str4 Str3 $MOV　Str3 Str4	LD,MOV,LD>

6 - 51

■関数定義

BOOL MIN_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn, ANY_SIMPLE D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	検索対象データ
D1	OUT	検索結果

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, iData1とiData2とiData3に格納されているデータから*)
(*  最小のものをResultに格納します。                                        *)
    M0 := MIN_E( X0, iData1, iData2, iData3, Result ) ;
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 52

6.8.4 リミッタ　LIMIT
LIMIT_E

指定されたデータが上下限リミット値(最小・最大出力限界値)の範囲内か否かにより出力値を制御します。

■関数定義　ANY_SIMPLE LIMIT(ANY_SIMPLE MIN, ANY_SIMPLE S1, ANY_SIMPLE MAX);

●引数

引数名	IN/OUT	内容
MIN	IN	最小出力限界値
S1	IN	入力値
MAX	IN	最大出力限界値

●戻り値

戻り値名	内容
ANY_SIMPLE	出力値 MIN(下限値) ＞ S1(入力値) の場合　・・・戻り値＝MIN(下限値) MAX(上限値) ＜ S1(入力値) の場合　・・・戻り値＝MAX(上限値) MIN(下限値) ≦ S1(入力値) ≦ MAX(上限値)の場合 ・・・戻り値＝S1(入力値)

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	Real4:= LIMIT(Real1,Real2, Real3);	LDE>= Real2 Real1 ANDE<= Real2 Real3 EMOV Real2 Real4 LDE< Real2 Real1 EMOV Real1 Real4 LDE> Real2 Real3 EMOV Real3 Real4	LDE>=,ANDE<=,EMOV, LDE<,LDE>
INT	Int4:= LIMIT(Int1,Int2,In t3);	LD SM400 LIMIT Int1 Int3 Int2 Int4	LD,LIMIT
DINT	Dint4:= LIMIT(Dint1,Dint2, Dint3);	LD SM400 DLIMIT Dint1 Dint3 Dint2 Dint4	LD,DLIMIT
BOOL	bBit4:= LIMIT(bBit1,bBit2,b Bit3);	LD bBit2 OR bBit1 AND bBit3 OUT bBit4	LD,OR,AND,OUT
STRING	Str4:= LIMIT(Str1,Str2,Str 3);	LD$>= Str2 Str1 AND$<= Str2 Str3 $MOV Str2 Str4 LD$< Str2 Str1 $MOV Str1 Str4 LD$> Str2 Str3 $MOV Str3 Str4	LD$>=,AND$<=,MOV, LD<,LD$>

6 - 53

■関数定義

BOOL LIMIT_E(BOOL EN, ANY_SIMPLE MIN, ANY_SIMPLE S1, ANY_SIMPLE MAX, ANY_SIMPLE D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
MIN	IN	最小出力限界値
S1	IN	入力値
MAX	IN	最大出力限界値
D1	OUT	出力値 MIN(下限値) ＞ S1(入力値) の場合・・・D1＝MIN(下限値) MAX(上限値) ＜ S1(入力値) の場合・・・D1＝MAX(上限値) MIN(下限値) ≦ S1(入力値) ≦ MAX(上限値)の場合 ・・・D1＝S1(入力値)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, iData2のデータが最小値であるiData1のデータ *)
(*  より小さければiData1の値を最大値であるiData3のデータより大きけれ *)
(*  ばiData3の値をそうでなければiData2の値をResultに格納します。     *)
 M0 := LIMIT_E( X0, iData1, iData2, iData3, Result ) ;
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 54

6.8.5　マルチプレクサ　　MUX
MUX_E

指定された選択条件により，指定されたデータから一つを選択します。

■関数定義　　ANY MUX (INT n, ANY S1, ANY S2,....,ANY Sn);

●引数

引数名	IN/OUT	内容
n	IN	選択条件
S1～Sn	IN	選択対象データ

●戻り値

戻り値名	内容
ANY	選択結果 n＝1の場合　　戻り値＝S1 n＝2の場合　　戻り値＝S2 ：　　： n＝nの場合　　戻り値＝Sn

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	Real4 := MUX(Int1, Real1,Real2, Real3);	LD=　Int1 K1 EMOV　Real1 Real4 LD=　Int1 K2 EMOV　Real2 Real4 LD=　Int1 K3 EMOV　Real3 Real4	LD=,EMOV
INT	Int4:= MUX(wCon1 , Int1 , Int2, Int3);	LD=　wCon1 K1 MOV　Int1 Int4 LD=　wCon1 K2 MOV　Int2 Int4 LD=　wCon1 K3 MOV　Int3 Int4	LD=,MOV
DINT	Dint4:= MUX(D0, Dint1,Dint2,Dint3);	LD=　D0 K1 DMOV　Dint1 Dint4 LD=　D0 K2 DMOV　Dint2 Dint4 LD=　D0 K3 DMOV　Dint3 Dint4	LD=,DMOV
BOOL	bBit4:= MUX(3,bBit1,bBit2,b Bit3);	LD=　K3 K1 MPS AND　bBit1 SET　bBit4 MPP ANI　bBit1 RST　bBit4 LD=　K3 K2 MPS AND　bBit2 SET　bBit4 MPP ANI　bBit2 RST　bBit4 LD=　K3 K3 MPS AND　bBit3 SET　bBit4 MPP ANI　bBit3 RST　bBit4	LD=,MPS,AND,SET, MPP,ANI,RST

6 - 55

■関数定義　　BOOL MUX_E(BOOL EN, INT n, ANY S1, ANY S2,....,ANY Sn, ANY D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
n	IN	選択条件
S1～Sn	IN	選択対象データ
D1	OUT	選択結果 n＝1の場合　D1＝S1 n＝2の場合　D1＝S2 ：　： n＝nの場合　D1＝Sn

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると, iData1のデータから判断して, iData2とiData3と      *)
(*  iData4とiData5に格納されているデータの内の一つをResultに格納します。 *)
    M0 := MUX_E( X0, iData1, iData2, iData3, iData4, iData5, Result );
```

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 56

6.9　比較機能

6.9.1　右辺より大きい(＞)　　GT_E

指定されたすべてのデータにおいて，＞(より大きい)の関係が成立しているか否かを取得します。

■関数定義　　BOOL GT_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	比較対象データ
D1	OUT	比較結果

備考）D1　＝　(S1＞S2) & (S2＞S3)&・・・& (Sn-1＞Sn)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	GT_E(M0 , Real1, Real2, Real3, bBit1);	LDE>　Real1 Real2 ANDE>　Real2 Real3 OUT　M8191 LD　M0 AND　M8191 SET　bBit1 LD　M0 ANI　M8191 RST　bBit1	LDE>,ANDE>,OUT, LD,AND,SET,ANI, RST
INT	GT_E(M0 , Int1, Int2, Int3, bBit1);	LD>　Int1 Int2 AND>　Int2 Int3 OUT　M8191 LD　M0 AND　M8191 SET　bBit1 LD　M0 ANI　M8191 RST　bBit1	LD>,AND>,OUT,LD, AND,SET,ANI,RST
DINT	GT_E(M0 , Dint1, Dint2 , Dint3, bBit1);	LDD>　Dint1 Dint2 ANDD>　Dint2 Dint3 OUT　M8191 LD　M0 AND　M8191 SET　bBit1 LD　M0 ANI　M8191 RST　bBit1	LDD>,ANDD>,OUT LD,AND,SET,ANI, RST

6 - 57

引数型	STプログラム	変換結果	使用命令
BOOL	GT_E(M0 , M100, M101, M102, M103, bBit1);	LD M100 ANI M101 LD M101 ANI M102 ANB LD M102 ANI M103 ANB OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD,ANI,ANB,OUT, AND,SET,RST
STRING	GT_E(M0 , Str1, Str2 , Str3, bBit1);	LD$> Str1 Str2 AND$> Str2 Str3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD$>,AND$>,OUT LD,AND,SET,ANI RST

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 58

6.9.2　右辺より大きい，または等しい(>=)　　GE_E

指定されたすべてのデータにおいて，≧(より大きい，または等しい)の関係が成立しているか否かを取得します。

■関数定義　　BOOL GE_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	比較対象データ
D1	OUT	比較結果

備考）D1　＝　(S1≧S2) & (S2≧S3)&・・・& (Sn-1≧Sn)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	GE_E(M0 , Real1, Real2, Real3, bBit1);	LDE>= Real1 Real2 ANDE>= Real2 Real3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LDE>=,ANDE>=, OUT,LD,AND,SET, ANI,RST
INT	GE_E(M0 , Int1, Int2, Int3, bBit1);	LD>= Int1 Int2 AND>= Int2 Int3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD>=,AND>=,OUT LD,AND,SET,ANI, RST
DINT	GE_E(M0 , Dint1, Dint2 , Dint3, bBit1);	LDD>= Dint1 Dint2 ANDD>= Dint2 Dint3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LDD>=,ANDD>=, OUT,LD,AND,SET ANI,RST

6 - 59

引数型	STプログラム	変換結果	使用命令
BOOL	GE_E(M0 , M100, M101, M102, M103, bBit1);	LD M100 ORI M101 LD M101 ORI M102 ANB LD M102 ORI M103 ANB OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD,ORI,ANB,OUT AND,SET,ANI,RST
STRING	GE_E(M0 , Str1, Str2 , Str3, bBit1);	LD$>= Str1 Str2 AND$>= Str2 Str3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD$>=,AND$>=, OUT,LD,AND,SET, LD,ANI,RST

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 60

6.9.3　等しい(=)　EQ_E

指定されたすべてのデータにおいて，＝(等しい)の関係が成立しているか否かを取得します。

■関数定義　BOOL EQ_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	比較対象データ
D1	OUT	比較結果

備考）D1　＝　(S1＝S2) & (S2＝S3)&・・・&(Sn-1＝Sn)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	b_result := EQ_E(b_select, r_data1, r_data2, r_data3, b_data1);	LDE= r_data1 r_data2 ANDE= r_data2 r_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST data1 LD b_select OUT b_result	LDE=,ANDE=,OUT LD,AND,SET,ANI, RST
INT	B100 := EQ_E(M20, D10, D20, D30, M200);	LD= D10 D20 AND= D20 D30 OUT M8191 LD M20 AND M8191 SET M200 LD M20 ANI M8191 RST M200 LD M20 OUT B100	LD=,AND=,OUT LD,AND,SET,ANI RST
DINT	b_result := EQ_E(b_select, di_data1, di_data2, di_data3, b_data1);	LDD= di_data1 di_data2 ANDD= di_data2 di_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDD=,ANDD=,OUT, LD,AND,SET,ANI, RST

6 - 61

引数型	STプログラム	変換結果	使用命令
BOOL	b_result := EQ_E(b_select, X10, X11, X12, M20);	LD X10 AND X11 LDI X10 ANI X11 ORB LD X11 AND X12 LDI X11 ANI X12 ORB ANB OUT M8191 LD b_select AND M8191 SET M20 LD b_select ANI M8191 RST M20 LD b_select OUT b_result	LD,AND,LDI,ANI, ORB,ANB,SET,RST
STRING	b_result := EQ_E(b_select, s_ary1, s_ary2, b_data1);	LD$= s_ary1 s_ary2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LD$=,OUT,LD,AND, SET,ANI,RST

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 62

6.9.4　右辺より小さい，または等しい(<=)　　LE_E

指定されたすべてのデータにおいて，≦(より小さい，または等しい)の関係が成立しているか否かを取得します。

■関数定義

BOOL LE_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	比較対象データ
D1	OUT	比較結果

備考）D1　＝　(S1≦S2) & (S2≦S3)&・・・&(Sn-1≦Sn)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	b_result := LE_E(b_select, r_data1, r_data2, r_data3, b_data1);	LDE<= r_data1 r_data2 ANDE<= r_data2 r_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDE<=,ANDE<=, OUT,LD,AND,SET, ANI,RST
INT	B100 := LE_E(M20, D10, D20, D30, M200);	LD<= D10 D20 AND<= D20 D30 OUT M8191 LD M20 AND M8191 SET M200 LD M20 ANI M8191 RST M200 LD M20 OUT B100	LD<=,AND<=,OUT LD,AND,SET,ANI, RST

6 - 63

引数型	STプログラム	変換結果	使用命令
DINT	b_result := LE_E(b_select, di_data1, di_data2, di_data3, b_data1);	LDD<= di_data1 di_data2 ANDD<= di_data2 di_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDD<=,ANDD<=, OUT,LD,AND,SET, ANI,RST
BOOL	b_result := LE_E(b_select, X10, X11, X12, M20);	LDI X10 OR X11 LDI X11 OR X12 ANB OUT M8191 LD b_select AND M8191 SET M20 LD b_select ANI M8191 RST M20 LD b_select OUT b_result	LDI,OR,ANB,OUT, AND,SET,ANI,RST
STRING	b_result := LE_E(b_select, s_ary1, s_ary2, b_data1);	LD$<= s_ary1 s_ary2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LD$<=,OUT,LD, AND,SET,ANI,RST

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 64

6.9.5　右辺より小さい(＜)　　LT_E

指定されたすべてのデータにおいて，＜(より小)の関係が成立しているか否かを取得します。

■関数定義　　BOOL LT_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,...., ANY_SIMPLE Sn, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	比較対象データ
D1	OUT	比較結果

備考）D1　＝　(S1＜S2) & (S2＜S3)&・・・&(Sn-1＜Sn)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	b_result := LT_E(b_select, r_data1, r_data2, r_data3, b_data1);	LDE<　r_data1 　　r_data2 ANDE<　r_data2 　　r_data3 OUT　M8191 LD　b_select AND　M8191 SET　b_data1 LD　b_select ANI　M8191 RST　b_data1 LD　b_select OUT　b_result	LDE<,ANDE<,OUT, LD,AND,SET,ANI, RST
INT	B100 := LT_E(M20, D10, D20, D30, M200);	LD<　D10 D20 AND<　D20 D30 OUT　M8191 LD　M20 AND　M8191 SET　M200 LD　M20 ANI　M8191 RST　M200 LD　M20 OUT　B100	LD<,AND<,OUT, LD,SET,ANI,RST

6 - 65

引数型	STプログラム	変換結果		使用命令
DINT	b_result := LT_E(b_select, di_data1, di_data2, di_data3, b_data1);	LDD< ANDD< OUT LD AND SET LD ANI RST LD OUT	di_data1 di_data2 di_data2 di_data3 M8191 b_select M8191 b_data1 b_select M8191 b_data1 b_select b_result	LDD<,ANDD<,OUT, LD,AND,SET,ANI, RST
BOOL	b_result := LT_E(b_select, X10, X11, X12, M20);	LDI AND LDI AND ANB OUT LD AND SET LD ANI RST LD OUT	X10 X11 X11 X12 M8191 b_select M8191 M20 b_select M8191 M20 b_select b_result	LDI,AND,ANB,OUT, LD,SET,ANI,RST
STRING	b_result := LT_E(b_select, s_ary1, s_ary2, b_data1);	LD$< OUT LD AND SET LD ANI RST LD OUT	s_ary1 s_ary2 M8191 b_select M8191 b_data1 b_select M8191 b_data1 b_select b_result	LD$<,OUT,LD,AND, SET,ANI,RST

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 66

6.9.6　等しくない(◇)　　NE_E

指定された2つのデータにおいて，≠(等しくない)関係が成立しているか否かを取得します。

■関数定義　　BOOL NE_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2, BOOL D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	比較対象データ
S2	IN	比較対象データ
D1	OUT	比較結果

備考）D1=(S1≠S2)

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

引数型	STプログラム	変換結果	使用命令
REAL	b_result := NE_E(b_select, r_data1, r_data2, b_data1);	LDE<>　r_data1 　　r_data2 OUT　M8191 LD　b_select AND　M8191 SET　b_data1 LD　b_select ANI　M8191 RST　b_data1 LD　b_select OUT　b_result	LDE<>,OUT,LD, AND,SET,ANI,RST
INT	B100 := NE_E(M20, D10, D20, M200);	LD<>　D10 D20 OUT　M8191 LD　M20 AND　M8191 SET　M200 LD　M20 ANI　M8191 RST　M200 LD　M20 OUT　B100	LD<>,OUT,LD,AND, SET,ANI,RST

6 - 67

引数型	STプログラム	変換結果	使用命令
DINT	b_result := NE_E(b_select, di_data1, di_data2, b_data1);	LDD<> di_data1 di_data2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDD<>,OUT,LD,AND, SET,ANI,RST
BOOL	b_result := NE_E(b_select, X10, X11, M20);	LD X10 ANI X11 LDI X10 AND X11 ORB OUT M8191 LD b_select AND M8191 SET M20 LD b_select ANI M8191 RST M20 LD b_select OUT b_result	LD,ANI,LDI,AND, ORB,OUT,SET,RST
STRING	b_result := NE_E(b_select, s_ary1, s_ary2, b_data1);	LD$<> s_ary1 s_ary2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LD$<>,OUT,LD,AND, SET,ANI,RST

使用できるデータ型は『3.2.2 ANY型について』を参照してください。

6 - 68

6.10　文字列機能

6.10.1　文字列長取得　　LEN
LEN_E

指定された文字列データの文字列長を取得します。

■関数定義　　INT LEN (STRING S1);

●引数

引数名	IN/OUT	内容
S1	IN	文字列長を取得するデータ（文字列データ）

●戻り値

戻り値名	内容
INT	文字列長結果（BIN16ビットデータ）

備考）本関数はベーシックモデルQCPUでは使用できません。

●使用例

引数型	STプログラム	変換結果	使用命令
STRING	i_data1 := LEN(s_ary1);	LD　SM400 LEN　s_ary1 i_data1	LD,LEN

■関数定義　　BOOL LEN_E(BOOL EN, STRING S1, INT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	文字列長を取得するデータ（文字列データ）
D1	OUT	文字列長結果（BIN16ビットデータ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataに格納されている文字列の長さを取得して    *)
(*  Resultに格納します。                                                  *)
    M0 := LEN_E( X0, sData, Result ) ;
```


6 - 69

6.10.2　文字列の開始位置から取得　LEFT
LEFT_E

指定された文字列の左(文字列の先頭)から指定されたn文字分の文字列を取得します。

■関数定義　STRING LEFT (STRING S1, INT n);

●引数

引数名	IN/OUT	内容
S1	IN	取得するデータ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）

●戻り値

戻り値名	内容
STRING	取得結果（文字列データ）

備考）本関数はベーシックモデルQCPUでは使用できません。
取得した文字列データを格納するデータ領域は，n+1文字分を確保してください。

●使用例

引数型	STプログラム	変換結果		使用命令
STRING	s_ary1 := LEFT(s_ary2, i_data1);	LD LEFT	SM400 s_ary2 s_ary1 i_data1	LD,LEFT

■関数定義　BOOL LEFT _E(BOOL EN, STRING S1, INT n, STRING D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	取得するデータ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）
D1	OUT	取得結果（文字列データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataに格納されている文字列データの左から，      *)
(*  iDataで指定された文字数分の文字列を取得してResult に格納します。      *)
    M0 := LEFT_E( X0, sData, iData, Result ) ;
```


6 - 70

6.10.3　文字列の終端から取得　　RIGHT
RIGHT_E

指定された文字列の右(文字列の最終)から指定されたn文字分の文字列を取得します。

■関数定義　　STRING RIGHT (STRING S1, INT n);

●引数

引数名	IN/OUT	内容
S1	IN	取得する文字列データ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）

●戻り値

戻り値名	内容
STRING	取得結果（文字列データ）

備考）本関数はベーシックモデルQCPUでは使用できません。
取得した文字列データを格納するデータ領域は，n+1文字分を確保してください。

●使用例

引数型	STプログラム	変換結果		使用命令
STRING	s_ary1 := RIGHT(s_ary2, i_data1);	LD RIGHT	SM400 s_ary2 s_ary1 i_data1	LD,RIGHT

■関数定義　　BOOL RIGHT _E(BOOL EN, STRING S1, INT n, STRING D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	取得する文字列データ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）
D1	OUT	取得結果（文字列データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataに格納されている文字列の右からiDataで      *)
(*  指定された文字数分の文字列を取得してResult に格納します。              *)
    M0 := RIGHT_E( X0, sData, iData, Result ) ;
```


6 - 71

6.10.4　文字列の指定位置から取得　　MID
MID_E

指定された文字列データの左（文字列の先頭）から，指定された位置より指定されたn文字分の文字列データを取得します。

■関数定義

```
STRING MID( STRING S1, INT n, INT POS );
```

●引数

引数名	IN/OUT	内容
S1	IN	取得する文字列データ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）
POS	IN	取得するデータの先頭位置（BIN16ビットデータ）

●戻り値

戻り値名	内容
STRING	取得結果（文字列データ）

備考）本関数はベーシックモデルQCPUでは使用できません。
取得した文字列データを格納するデータ領域は，n+1文字分を確保してください。

●使用例

引数型	STプログラム	変換結果	使用命令
STRING	s_ary1 := MID(s_ary2, i_data1, i_data2);	LD　SM400 MOV　i_data1 　　D10239 MOV　i_data2 　　D10238 MIDR　s_ary2 　　s_ary1 　　D10238	LD,MOV,MIDR

■関数定義

```
BOOL MID_E( BOOL EN, STRING S1, INT n, INT POS, STRING D1 );
```

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	取得する文字列データ（文字列データ）
n	IN	取得する文字数（BIN16ビットデータ）
POS	IN	取得するデータの先頭位置（BIN16ビットデータ）
D1	OUT	取得結果（文字列データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataに格納されている文字列の先頭からiData2番目*)
(*  の位置からiData1に格納されている文字数分取得してResultに格納します   *)
    M0 := MID_E( X0, sData, iData1, iData2, Result ) ;
```


6 - 72

6.10.5　文字列の連結　　CONCAT
CONCAT_E

指定された文字列データすべてを連結します。

■関数定義　　STRING CONCAT(STRING S1, STRING S2,....,STRING Sn);

●引数

引数名	IN/OUT	内容
S1～Sn	IN	連結するデータ（文字列データ）

●戻り値

戻り値名	内容
STRING	連結結果（文字列データ）

備考）本関数はベーシックモデルQCPUでは使用できません。
連結した文字列データを格納するデータ領域は，連結した文字数+1文字分を確保してください。

●使用例

引数型	STプログラム	変換結果		使用命令
STRING	s_result := CONCAT(s_ary1, s_ary2, s_ary3);	LD $MOV $+ $+	SM400 s_ary1 s_result s_ary2 s_result s_ary3 s_result	LD,$MOV,$+

■関数定義　　BOOL CONCAT_E(BOOL EN, STRING S1, STRING S2,....,STRING Sn, STRING D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1～Sn	IN	連結するデータ（文字列データ）
D1	OUT	連結結果（文字列データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sData1, sData2に格納されている文字列データを   *)
(*  連結させResult に格納します。                                        *)
    M0 := CONCAT_E( X0, sData1, sData2, Result ) ;
```


6 - 73

6.10.6　指定位置への文字列挿入　　INSERT
INSERT_E

指定された文字列データの指定位置以降へ文字列データを挿入します。

■関数定義　　STRING INSERT(STRING S1, STRING S2, INT POS);

●引数

引数名	IN/OUT	内容
S1	IN	挿入されるデータ（文字列データ）
S2	IN	挿入データ　　（文字列データ）
POS	IN	挿入位置（BIN16ビットデータ）

●戻り値

戻り値名	内容
STRING	挿入結果（文字列データ）

備考）本関数はベーシックモデルQCPUでは使用できません。
挿入後の文字列データを格納するデータ領域は，挿入後の文字数+1文字分を確保してください。

●使用例

引数型	STプログラム	変換結果		使用命令
STRING	w_Str3 := INSERT(w_Str1,w_Str2,w_Word1);	LD	SM400	LD,$+,AND<>,MOV,-,MIDW,LEN
		$+	w_Str2	
			w_Str1	
			w_Str3	
		AND<>	w_Word1 K1	
		MOV	K1 D10238	
		-	w_Word1 K1	
			D10239	
		MIDW	w_Str1	
			w_Str3	
			D10238	
		MOV	w_Word1	
			D10238	
		LEN	w_Str2	
			D10239	
		MIDW	w_Str2	
			w_Str3	
			D10238	

■関数定義　　BOOL INSERT_E(BOOL EN, STRING S1, STRING S2, INT POS, STRING D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	挿入されるデータ（文字列データ）
S2	IN	挿入データ（文字列データ）
POS	IN	挿入位置（BIN16ビットデータ）
D1	OUT	挿入結果（文字列データ）

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sData1の文字列データの先頭からiData番目の位置  *)
(*  にsData2の文字列データを挿入してResult に格納します。                *)
    M0 := INSERT_E( X0, sData1, sData2, iData, Result ) ;
```


6 - 74

6.10.7　文字列の指定位置からの削除　DELETE
DELETE_E

指定された文字列の指定位置からn文字分の文字列を削除します。

■関数定義　STRING DELETE(STRING S1, INT n, INT POS);

●引数

引数名	IN/OUT	内容
S1	IN	削除されるデータ（文字列データ）
n	IN	削除文字数（BIN16ビットデータ）
POS	IN	削除位置（BIN16ビットデータ）

●戻り値

戻り値名	内容
STRING	削除結果（文字列データ）

備考）本関数はベーシックモデルQCPUでは使用できません。
削除後の文字列データを格納するデータ領域は，削除後の文字数+1文字分を確保してください。
削除位置POSが0の場合，削除されるデータS1の後ろ(右側)からｎ文字分の文字列を削除します。

●使用例

引数型	STプログラム	変換結果	使用命令
STRING	w_Str2 := DELETE(w_Str1,w_Word1, w_Word2);	LD　SM400 LEN　w_Str1 　　D10238 -　　w_Word1 D10238 RIGHT　w_Str1 　　w_Str2 　　D10238 -　　w_Word2 K1 　　D10239 AND<>　10239 K0 MOV　K1 D10238 MIDW　w_Str1 　　w_Str2 　　D10238	LD,LEN,-,RIGHT, AND<>,MOV,MIDW

■関数定義　BOOL DELETE_E(BOOL EN, STRING S1, INT n, INT POS, STRING D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	削除されるデータ（文字列データ）
n	IN	削除文字数（BIN16ビットデータ）
POS	IN	削除位置（BIN16ビットデータ）
D1	OUT	削除結果（文字列データ）

備考）削除後の文字列データを格納するデータ領域は，削除後の文字数+1文字分を確保してください。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sDataの文字列データの先頭よりiData2番目から     *)
(*  iData1で指定された文字数分の文字列を削除してResult に格納します。     *)
    M0 := DELETE_E( X0, sData, iData1, iData2, Result ) ;
```


6 - 75

6.10.8　文字列の指定位置からの置換　　REPLACE
REPLACE_E

指定された文字列データの指定位置からn文字分の文字列データを指定された文字列に置き換えます。

■関数定義　　STRING REPLACE(STRING S1, STRING S2, INT n, INT POS);

●引数

引数名	IN/OUT	内容
S1	IN	置換されるデータ（文字列データ）
S2	IN	置換するデータ（文字列データ）
n	IN	置換文字数（BIN16ビットデータ）
POS	IN	置換開始位置（BIN16ビットデータ）

●戻り値

戻り値名	内容
STRING	置換結果（文字列データ）

備考）本関数はベーシックモデルQCPUでは使用できません。
置換後の文字列データを格納するデータ領域は，置換後の文字数+1文字分を確保してください。

●使用例

引数型	STプログラム	変換結果		使用命令
STRING	w_Str3 := REPLACE(w_Str1,w_Str2, w_Word1, w_Word2);	LD $MOV MOV MOV MIDW 	SM400 w_Str1 w_Str3 w_Word1 D10239 w_Word2 D10238 w_Str2 w_Str3 D10238	LD,$MOV,MOV, MIDW

■関数定義　　BOOL REPLACE_E(BOOL EN, STRING S1, STRING S2, INT n, INT POS, STRING D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	置換されるデータ（文字列データ）
S2	IN	置換するデータ（文字列データ）
n	IN	置換文字数（BIN16ビットデータ）
POS	IN	置換開始位置（BIN16ビットデータ）
D1	OUT	置換結果（文字列データ）

備考）置換後の文字列データを格納するデータ領域は，置換後の文字数+1文字分を確保してください。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sData1の文字列データの先頭よりiData2番目から，  *)
(*  Data1で指定された文字数分の文字列データとsData2の文字列データとを    *)
(*  置き換えてResultに格納します。                                        *)
    M0 := REPLACE_E( X0, sData1, sData2, iData1, iData2, Result ) ;
```


6 - 76

6.10.9　文字列の指定位置からの検索　　FIND
FIND_E

指定された文字列から指定文字列を検索します。

■関数定義　　INT FIND(STRING S1, STRING S2);

●引数

引数名	IN/OUT	内容
S1	IN	検索されるデータ（文字列データ）
S2	IN	検索するデータ（文字列データ

●戻り値

戻り値名	内容
INT	最初に検索された検索位置（BIN16ビットデータ）

備考） 本関数はベーシックモデルQCPUでは使用できません。
検索できなかった場合，戻り値は0になります。

●使用例

引数型	STプログラム	変換結果		使用命令
STRING	w_Word1:=	LD	SM400	LD,INSTR
	FIND(w_Str1,w_Str2);	INSTR	w_Str2	
			w_Str1	
			w_Word1 K1	

■関数定義　　BOOL FIND_E(BOOL EN, STRING S1, STRING S2, INT D1);

●引数

引数名	IN/OUT	内容
EN	IN	実行条件(TRUEの時のみ関数を実行します)
S1	IN	検索されるデータ（文字列データ）
S2	IN	検索するデータ（文字列データ）
D1	OUT	最初に検索された検索位置（BIN16ビットデータ）

備考） 検索できなかった場合，戻り値は0になります。

●戻り値

戻り値名	内容
BOOL	実行条件

●使用例

```
(*  実行条件X0がONすると，sData1の文字列データからsData2の文字列データを *)
(*  検索して，最初に検索された検索位置をResult に格納します。           *)
    M0 := FIND_E( X0, sData1, sData2, Result ) ;
```


6 - 77

7　エラー一覧

作成したSTプログラムの変換時に発生するエラーについて説明します。
STプログラムをCPUユニットに書き込んだ時に発生する実行エラーについては『MELSEC-Q/L プログラミングマニュアル（共通命令編）』『QCPUユーザーズマニュアル（ハードウェア設計・保守点検編）』『MELSEC-L CPUユニットユーザーズマニュアル（ハードウェア設計・保守点検編）』を参照ください。

●変換エラーが発生すると

プログラム中のエラーについてエラーダイアログを表示します。
1つのプログラムに対して発生するエラーの最大数は1000個までです。1000個を超えるエラーについては，エラー一覧に表示されません。

●変換エラー表示について

1つのプログラム文で複数のエラーを表示する場合や，1つのエラーで複数のメッセージを表示する場合があります。

●変換エラー一覧(エラーメッセージ・原因・処置)

No.	エラーメッセージ	原因	処置
1	解析できない文字列が存在します。(C1009)	解析できない文字列が存在する。 解析できない文字列には以下の例があります。 例1：2## 書式が間違っている 例2：STRING型 ： STRV1を定義 STRV1 := $"abc"; $記号を使用する 例3：D0 := !10; !記号を使用する 例4：J25|¥K4X0 := 5; |記号を使用する	正しい文字列に変更してください。
2	解析できない演算子が存在します。(C1010)	解析できない演算子が存在する。 例1：Y0 : = M0 => M1;	正しい演算子に変更してください。
3	実数定数が間違っています。(C1013)	実数定数の記述が不正です。 不正な記述には以下の例があります。 例1：REAL型 ： RealV1を定義 RealV1 := 1.; 実数定数の書式が間違っている 例2：RealV1 := 0.1E; 実数定数の書式が間違っている	正しい実数定数の記述に変更してください。
4	デバイスの記述が間違っています。(C1014)	デバイスの記述が不正です。 不正な記述には以下の例があります。 例1：D0.10 := TRUE; ワードデバイスのビットNo 指定が間違っている 例2：D0@ := 0;	正しいデバイスの記述に変更してください。

7 - 1

No.	エラーメッセージ	原因	処置
5	デバイスの記述が間違っています。(C1017)	デバイスの記述が不正です。 不正な記述には以下の例があります。 例1：D0 := %MMW0.10; 使用できない書式で記述した	正しいデバイスの記述に変更してください。
6	コメントの記述が間違っています。(C1018)	コメントの記述が不正です。 ″(*″ ″*)″の形式になっていない。 不正な記述には以下の例があります。 例1：(* * 括弧が不足している 例2：(*(* 括弧と*の形式が間違っている 例3：(* *) ″*″と″)″の間にスペースが存在する 例4：(*aaaaa) *が不足している	正しいコメントの記述に変更してください。
7	文字列定数の記述が間違っています。(C1019)	文字列型定数の記述が不正です。 不正な記述には以下の例があります。 例1：STRING型 ： STRV1を定義 STRV1 := ″″″; 例2：STRV1 := ″; ″が不足している 例3：STRV1 := ″'文字″; 文字列″″内に'が存在する 例4：STRV1 := ″$″; エスケープシーケンスの使用方法に誤りがある	正しい文字列型定数の記述に変更してください。
8	定数の記述が間違っています。(C1020～C1023)	サポートしていないデータ型を使用した。もしくは間違った定数の記述をした。 不正な記述には以下の例があります。 例1：W_TMP := TIME#1100_0101; 例2：W_TMP := T#0; 例3：W_TMP := ２#0; ″２″が全角 例4：W_TMP := DT#1900-01-01, 00:00:00; 例5：W_TMP := D#1994-06; 例6：W_TMP := TOD#09:30:61;	使用されたデータ型はサポートしておりません。正しいデータ型に変更してください。定数を正しい記述へ変更してください。
9	未定義変数です。(*1)(C1028) (*1には変数名が入ります。)	定義されていない変数を使用した。 未定義変数使用例には以下があります。 例1：I_TEST :=1 ; ラベル設定せずにラベルを使用する 例2：D0 := HAAH; 16進数でA～F以外を使用する 例3：D0 := 1234 ; 式の中で全角スペースを使用する	使用する変数を定義してください。

7 - 2

No.	エラーメッセージ	原因	処置
10	配列の要素指定の方法に誤りがあります。(C1033)	配列の要素指定の方法が間違っている。 例1：ワード型配列ラベル：W_ARY W_ARY[0,1] := 1; 定義した配列と違う書式で記述した	正しい配列の記述に変更してください。
11	未定義関数です。(*1)(C1049) (*1には関数名が入ります。)	定義されていない関数を使用した。 未定義関数使用例には以下があります。 例1：実数型ラベル ： RE_1 M0 := OS_E_MD(TRUE,E1.0,RE_1);	正しい関数名の記述に変更してください。
12	変数名またはデバイス名が長すぎます。(C1077)	変数名が16文字を超えている。 または，デバイス名が長すぎる。 エラーとなるプログラム例には以下があります。 例1：abcde678901234567 := D10; 例2：D0 := D000000･･･ 00000000000001; デバイス名が長すぎる	定義されている変数名を使用してください。 正しいデバイスの記述に変更してください。
13	*1番目の引数に定数以外を使用しています。(C2021) (*1には引数エラー箇所が入ります。)	定数を指定すべき引数に定数以外を使用した。 エラーとなるプログラム例には以下があります。 例1：M1 := ROL(M0,X0); 2番目の引数に定数以外を使用した 例2：D100 := SHL(D0,D1); 2番目の引数に定数以外を使用した	指定された引数に定数を使用してください。
14	文法が間違っています。 (C2054)	間違った文法を記述した。 文法が不正となる例には以下があります。 例1：D0 ： 0; 代入文で"="を記述していない 例2：FOR ARY[0] := 0 TO D10 BY D20 DO D100 := D100+1; END_FOR; 反復変数に配列要素を指定した 例3：FOR　STR.W_TMP := 0 TO D10 BY D20 DO D100 := D100+1; END_FOR; 反復変数に構造体要素を指定した 例4：D0 := 1++++++++2; +演算子の使用方法に誤りがある 例5：ワード型配列：IntAry1 D0 := IntAry1[[0; 配列の記述方法に誤りがある 例6：CASE D0 OF 1 :D0 := K5; ELSE D1 := K5; END_CASE; "１"が全角文字	正しい文法に変更してください。

7 - 3

No.	エラーメッセージ	原因	処置

15	"*1"が不足しています。
	(C8006)
	(*1には
	END FOR
	;
	END WHILE
	END FOR
	END_REPEAT
	END_CASE
	END_IF
	が入ります。)
		ステートメントの最後尾に";"が記述されていない。	ステートメントの最後尾に";"を記述してください。

		FOR構文に"END FOR"が記述されていない。	FOR構文に"END FOR"を記述してください。
		WHILE構文に"END WHILE"が記述されていない。	WHILE構文に"END WHILE"を記述してください。
		REPEAT構文に"END_REPEAT"が記述されていない。	REPEAT構文に"END_REPEAT"を記述してください。
		CASE条件文に"END_CASE"が記述されてない。	CASE条件文に"END_CASE"を記述してください。
		IF条件文に"END_IF"が記述されていない。	IF条件文に"END_IF"を記述してください。
16	ループ文の外にEXIT構文が使用されています。(C8009)	ループ構文の外に"EXIT"構文が記述されている。	ループ構文の中に"EXIT"構文を記述してください。
17	定数の記述が間違っています。(C8010)	サポートしていないデータ型を使用した。 例1：タイマラベル：wTime wTime := T#1111111111111111 1s;	使用されたデータ型はサポートしておりません。正しいデータ型に変更してください。
18	未定義のFBが呼び出されました。(C8011)	定義されていないFBを呼び出した。 未定義FB使用例には以下があります。 例1：FB_1(); 未定義のFB呼出しを行う 例2：ワード型ラベル ： W_TMP W_TMP(); FB以外の変数を記述する	使用するFBを定義してください。
19	入力/入出力変数"*1"に値指定されていません。(C8012) (*1には入力・入出力変数名が入ります。)	FBの入力・入出力変数に値が指定されていない。 上記エラーとなる例には以下があります。 例1：入出力変数 ： IO_TEST1 流用FB名 ： FB1 FB1(); 例2：入出力変数 ： IO_TEST1 流用FB名 ： FB1 FB1(IO_TEST); 入力変数に値を代入しない	FBの入力・入出力変数に値を指定してください。
20	引数"*1"の型が不一致です。 (C8013) (*1には引数名が入ります。)	FB呼出しの引数と指定した値または変数の型が不一致です。 上記エラーとなる例には以下があります。 例1：入力変数（ワード型） ：IN1 流用FB名 ： FB1 FB1(IN1 := TRUE); ワード型入力変数にビット型定数を指定する 例2：入力変数（ワード型） ： IN1 出力変数（ワード型） ： OUT1 流用FB名 ： FB1 ダブルワード型 ： DIN1 FB1(IN1 ： =DIN1); ワード型入力変数にダブルワード型変数を指定する	FB呼出しの引数と一致した型に変更してください。

7 - 4

No.	エラーメッセージ	原因	処置
21	入出力変数,出力変数に値を代入できない変数は指定できません。"*1" (C8014) （*1には入出力変数名，出力変数名が入ります。）	呼び出したFBの入出力変数，出力変数に値を代入できない変数が指定されている。 例1：入出力変数 ： IO_TEST1 IO_TEST1 ： = TRUE; 流用FB名 ： FB1 FB1(IO_TEST1 := TRUE); 入出力変数に定数を渡している 例2：入力変数 ： IN1 出力変数 ： OUT1 流用FB名 ： FB1 ワード型定数ラベル：wCon FB1(IN1 ： =1,OUT1 := wCon); ワード型出力変数に定数ラベルを渡している	呼び出したFBの入出力変数，出力変数に値を代入可能な変数を指定してください。
22	FBの引数として使用できない変数"*1"が使用されています。(C8015) (*1には変数名が入ります。)	呼び出したFBの入力・出力・入出力変数以外の変数に値を渡している。 上記エラーとなる例には以下があります。 例1：入力変数 IN1 出力変数 OUT1 変数 TEST1 流用FB名 ： FB1 FB1(TEST1 := X10);	FBの入力・出力・入出力変数以外の変数をFBの呼び出し時に使用しないでください。
23	引数"*1"が重複して割り付けられています。(C8016) (*1には引数名が入ります。)	FB呼出しで同じ引数を2つ以上使用している。 例1：入出力変数（ビット型） ：INOUT1流用FB名：FB1 FB1(INOUT1 := TRUE,INOUT1 := FALSE);	FB呼出しで同じ引数を使用しないでください。
24	引数"*1"が未定義です。 (C8017) (*1には引数名が入ります。)	呼出しを行うFBの引数が未定義です。 例1：入出力変数 ： INOUT1 流用FB名 ： FB1 FB1(TMP_INOUT1 := TRUE);	呼出しを行うFBの引数を定義してください。
25	整数値"*1"が間違っています。(C8018) (*1には整数値が入ります。)	整数値が不正です。 例1：D1 := 9999999999 ; 整数値が使用可能範囲を超えている	使用できる範囲の整数値に変更してください。
26	定数"*1"が間違っています。 (C8019) (*1には定数が入ります。)	BOOL定数が不正です。 例1：D1 := 2##0011_0101; 間違ったBOOL定数を記述する 例2：M0 :=2 #F; 間違ったBOOL定数を記述する	使用できるBOOL定数の記述に変更してください。
27	配列変数の要素番号にワード型以外が使用されています。 (C8021)	要素指定にワード型以外が使用されている。 例1：ビット型配列 ： BoolAry1 実数型ラベル ： RealVal BoolAry1[RealVal] := x0; 例2：ビット型配列 ： BoolAry1 BoolAry1[D0<D1] 間違った要素指定を記述する	要素のデータ型をワード型に変更してください。

7 - 5

No.	エラーメッセージ	原因	処置
28	配列変数の要素番号が定義された要素数を超えています。(C8022)	指定された要素番号が配列定義の要素範囲を超えている。 例1：ワード型配列ラベル（要素数2）：Kosu Kosu[5] := D0;	要素番号を配列定義の要素範囲内に変更してください。
29	配列変数でない変数に，配列形式の記述がされています。(C8023)	配列変数でない変数に，配列形式の構文を記述した。 例1：ワード型ラベル：W_TMP1 W_TMP1[2] := 100; 配列でない変数に配列の書式で記述する 例2：aaa[1] := D0; 未定義ラベルを配列の書式で記述する	正しい変数の記述に変更してください。
30	"*1"は"*2"の要素ではありません。(C8024) (*1には構造体要素名またはFB変数名，*2には構造体名またはFB名が入ります。)	構造体の要素名が間違っている。またはFBの変数名が間違っている。 例1：構造体要素名 ： mem1 流用構造体名 ： InsSDT1 InsSDT1.mem2 := 100; 間違った構造体要素名を記述する 例2：入力変数 ： IN1 流用FB名 ： FB1 FB1(IN1 := 10); d0 := FB1.aaa; 未定義のFB出力変数を記述する	正しい構造体の要素名に変更してください。または正しいFBの変数名に変更してください。
31	FB出力として使用できない変数"*1"がFB"*2"に使用されています。(C8025) (*1にはFB変数名，*2にはFB名が入ります。)	FB出力として使用できないFB変数を使用した。 例1:内部変数（ワード型） ： TEMP1 流用FB名 ： FB1 D100 := FB1.TEMP1; 内部変数をFB出力として使用する	正しいFB変数を使用して，FB出力として記述してください。
32	変数"*1" (FB:*2)は引数以外に使用できません。(C8026) (*1にはFB変数名，*2にはFB名が入ります。)	FB変数の使用方法が間違っている。 例1: [FB定義] 入力変数 ： IN1 出力変数 ： OUT1 流用FB名 ： FB1 X1 := FB1.IN1; 入力変数をFB出力として使用する	FBの引数に正しいFB変数を使用してください。
33	未定義の構造体です。(C8027)	構造体名が不正である。 例1: 構造体 ： 無し ワード型ラベル ： W_TMP2 W_TMP2.mem1 := 100; 間違った構造体を記述する	正しい構造体名に変更してください。

7 - 6

No.	エラーメッセージ	原因	処置
34	*1番目に値を代入できない変数は指定できません。 ″*2″(C8028) (*1にはエラー箇所,*2には関数名, ″:=″が入ります。)	値が代入される箇所に, 定数や入力変数などの値を代入できない変数を指定する。 例1：ラベル（定数型） : cnt cnt := D10; ラベル定数に代入する 例2：ABS_E(TRUE,d0,K10); 関数の出力変数に定数を記述する 例3：FB入力変数（ワード型） : IN1 BPLUS_3_M(M0,K1,D0,FB1.IN1); 値が出力される引数に入力変数を指定する	値を代入可能な変数に変更してください。
35	″*1″番目の変数の型が不一致です。″*2″ (C8029) (*1には引数エラー箇所, *2には関数名が入ります。)	変数の型が不一致です。 例1：ワード型配列 : IntAry1[0..1] M1 := BACOS_MD(TRUE, IntAry1,D1);	関数の引数の指定されたエラー箇所の型を修正してください。もしくは変数の型を修正してください。
36	″*1″で不正な型が使用されています。(C8030) (*1には″:=″や″*″等の演算子が入ります。)	変数・デバイスの左辺と右辺のデータ型が異なる。 例1：D0 : =TRUE; ワードデバイスに対してビット型を代入する 例2：D1 := D2*M1; ワード型とビット型を演算する 例3：M0 := d1 > M1; ワード型とビット型を比較する	変数・デバイスの左辺と右辺のデータ型を同じにしてください。
37	関数″*1″の*2個の引数は定義と不一致です。(C8031) (*1には関数名,*2には定義と不一致である引数個数が入ります。)	関数呼出しの引数の数が定義と不一致です。 例1：ABS(); 定義されている引数の数より少なく記述する 例2：d0 := ABS(10,10); 定義されている引数の数より多く記述する	正しい関数の引数の個数に変更してください。

38	書式の型が不正です。(C8032)	制御構文において, 書式の型が不一致です。
		例1：ダブルワード型 : DwLBL
		FOR DwLBL := W1 TO W2 BY W3 DO
		W5 := W6;
		END_FOR;
		反復変数と最終値の式・増加式のデータ型が不一致
		例2：CASE W1 OF
		1: D0 := 1;
		2147483648 : D0 := 2;
		ELSE
		D0 := 10;
		END_CASE;
		整数式と選択値のデータ型が不一致
		例3：IF W1 THEN
		D100 := 1;
		END_IF;
		ブール式にワード型を指定する
			書式の型を合わせるように変更してください。

7 - 7

No.	エラーメッセージ	原因	処置
39	定数変数(FOR構文内)に代入はできません。 (C8033)	定数変数に書き込もうとした。 上記エラープログラム例には以下があります。 例1：定数ラベル ： tei FOR tei := W10 TO W20 BY W30 DO R10 := R20; END_FOR;	定数変数(FOR構文内)に書き込みはできません。
40	FOR構文にて、ワード/ダブルワード型以外の変数が使用されています。(C8034)	FOR構文でワード・ダブルワード以外の型の変数を使用している。 （反復変数に文字列・配列・構造体変数名を指定した場合等） 例1：文字列ラベル ： Str1, Str2, Str3, Str4 FOR Str1 := Str2 TO Str3 BY Str4 DO D0 := D100; END_FOR; 反復変数に文字列変数名を指定した	FOR構文において,正しい型を使用してください。
41	構文に"*1"が不足しています。 (C8039) (*1には DO UNTIL OF THEN DO が入ります。)	FOR構文に"DO"が記述されていない。 例1：FOR D1 := D2 TO D3 BY D4	FOR構文に"DO"を記述してください。
		REPEAT構文に"UNTIL"が記述されていない。	REPEAT構文に"UNTIL"を記述してください。
		CASE条件文に"OF"が記述されていない。	CASE条件文に"OF"を記述してください。
		IF条件文に"THEN"が記述されていない。	IF条件文に"THEN"を記述してください。
		ELSIF条件文に"THEN"が記述されていない。	ELSIF条件文に"THEN"を記述してください。
		WHILE構文に"DO"が記述されていない。	WHILE構文に"DO"を記述してください。
42	呼び出したFB"*1"の引数が間違っています。(C8040) (*1にはFB名が入ります。)	呼出したFBの入力・出力・入出力変数の記述が不正。 例1：流用FB名 ： FB1 FB1(X10); 例2：入力変数 ： IN1 流用FB名 ： FB1 FB1(FB1.IN1);	正しいFBの呼出し記述に変更してください。
43	関数の戻り値を格納する変数が指定されていません。 (C8041)	EN/ENOがない関数で,戻り値・戻り値を格納する変数が存在しなかった。 戻り値を格納する変数が指定されていない例には以下があります。 例1：INT_TO_DINT(D0);	関数の戻り値を記述してください。

7 - 8

No.	エラーメッセージ	原因	処置

44	制御構文のネストや条件が多いか制御構文の間が長すぎます。
	(C9017)
		制御構文のネストや条件が多いか制御構文のプログラムが長い。
		例1：IF D0 = 0 THEN
		IF D1 = 0 THEN
		・・・
		END_IF;
		END_IF;
		IF文内で598回以上ネストした
		例2：FOR D0 := 0 TO 100 BY 1 DO
		FOR D1 := 0 TO 100 BY 1 DO
		・・・
		END_FOR;
		END_FOR;
		FOR文内で299回以上ネストした
		例3：WHILE D0 < 10 DO
		WHILE D1 < 10 DO
		・・・
		END_WHILE;
		END_WHILE;
		WHILE文を598回以上ネストした
		例4：CASE W0 OF
		0: D0 := 0;
		1: D0 := 1;
		・・・
		1491: D0 := 1491;
		END_CASE;
		CASE文で整数選択値を
		1492個以上使用した
			制御構文のプログラムが長すぎます。ネストの数を少なくする,条件を少なくするなど,制御構文のプログラムを短くしてください。

45	関数"*1"の実行条件ENの値が正しくありません。(C9019) (*1には関数名が入ります。)	特定な関数では，実行条件ENに常時 TRUE を入れなくてはいけないが，FALSE を入れている。 例1：EI_M(FALSE); 例2：DI_M(0); 例3：COM_M (FALSE);	実行条件ENに正しい値を指定してください。
46	システムファイルの読出しに失敗しました。(C9020)	システムファイルが壊れている。	再度インストールしてください。
47	Z0,Z1はシステムで使用するため使用できません。(C9035)	Z0，Z1を使用している。 例1：INC_M(M10, D0Z1); 例2：Z0 := 10;	Z0，Z1 を使用しないよう変更ください。
48	定数*1は要素番号(*2..*3)の範囲外です。(C9039) (*1には要素番号，*2，*3には要素数が入ります。)	配列の要素番号が不正（範囲外）です。 例1：ワード型配列ラベル： IntAry1[255] IntAry1[K255] := 0;	指定した定数を要素数の範囲内にしてください。
49	0で除算しています。(C9065)	0で除算している。 例1：D0 := 10/0; 例2：D1 := W1/K0;	0で除算している個所を変更ください。

7 - 9

No.	エラーメッセージ	原因	処置
50	関数"*1"の戻り値は直接参照できません。(C9066) (*1には関数名が入ります。)	文字列関数（***_STR(), LEFT(), RIGHT()関数を指します）の戻り値を直接参照して演算できなかった場合。 例1：M0 := INT_TO_STR(D0) < "AAA";	エラーとなった文字列関数を別プログラムとし，その文字列関数の戻り値を使用するようプログラムを修正してください。
51	システムファイルの読出しに失敗しました。(C9072)	システムファイルが壊れている。	システムファイルが壊れています。再インストールしてください。
52	関数"*1"の変換時にエラーが発生しました。(C9076) (*1には関数名が入ります。)	変換結果に誤りがある。 例1：TIMER_H_M(X0,TC0,-1); 第3引数にマイナス値が使用されている	関数の引数には指定可能なデータ型，または指定可能範囲のデータを使用してください。
53	入力変数に式が使用されています。(C9118)	MELSEC関数の先頭デバイスを指定する入力変数に演算式や関数が指定された。 MELSEC関数の指定する入力変数に要素番号が可変のビット型配列要素が指定された。 MELSEC関数の指定する入力変数に要素番号が可変のビット型以外の配列要素が指定された。 例1：BMOV_M(X0,MAX(D0,D1,D2), D100,D200); 入力変数に関数を使用した 例2：TO_M(X0,D0+1,D1,D2,D3); 入力変数に演算式を使用した 例3：DTO_M(X0,Dint1+K8X0,D1,D2,D3); 入力変数に演算式を使用した 例4：BKRST_M(X0,ARY[D0],D1); 先頭デバイスを指定する入力変数S1に要素番号が可変のビット型配列要素を指定した 例5：BKPLUS_M(M0,ARY[D1], ARY[D2], ARY[D3], ARY[D4]); 先頭デバイスを指定する入力変数S1,S2に要素番号が可変の配列要素指定を行った	・演算式や関数が指定された場合 先頭デバイスを指定する入力変数には演算式や関数を指定することはできません。ラベル名はまたはデバイスを指定してください。 ・要素番号が可変のビット型配列要素が指定された場合 要素番号が可変のビット型配列要素指定は，デバイスの先頭を指定する引数には指定できません。要素番号を定数に修正する，もしくはラベル名またはビットデバイスを指定してください。 ・要素番号が可変のビット型以外の配列要素が指定された場合 要素番号が可変の配列要素指定を行うとコンパイラ内部で使用するインデックスレジスタに限りがあるため，要素番号を定数に修正する，もしくはラベル名またはビットデバイスを指定する,1つの関数内で使用する配列要素指定数を減らすなどの修正をしてください。
54	変換結果に誤りがあります。(F0028) "*1" (*1には不正な変換結果を表示します。)	STの文法上は正しいが，デバイスの仕様などによりエラーとなる。 例1：TS0 := TRUE;	エラーメッセージで表示されるリストの内容を確認してプログラムを修正してください。

7 - 10

No.	エラーメッセージ	原因	処置
55	文字列の使用可能な最大文字数は32文字までです。(F0102)	設定した最大値以上の文字数を使用している。 最大値以上の文字数を使用しているエラー例には以下があります。 例1：文字列ラベル：Str1 Str1 := ″123456789012345 678901234567890 123″; 文字列数が33文字の場合	文字列を32文字以内に変更してください。
56	不正なデバイス、または範囲外の数値が使用されています。(F0137) ″*1″ (*1には不正な変換結果を表示します。)	不正なデバイスまたは範囲外の数値を使用している。 例1：D0 := K-32769; 例2：D0 := H10001; 例3：M0 := COUNTER_M(TRUE,CC2,-1);	正しいデバイスまたは範囲内の数値に変更してください。
57	TIMER_Mの引数にタイマ以外のデバイスを使用しています。(F0177)	TIMER_Mの引数にタイマ以外のデバイスを使用している。 例1:TIMER_M(X0, CC0, 2);	関数TIMER_Mの引数にタイマデバイスを使用してください。
58	COUNTER_Mの引数にカウンタ以外のデバイスを使用しています。(F0178)	COUNTER_Mの引数にカウンタ以外のデバイスを使用している。 例1：COUNTER_M(X0, TC0, 2);	関数COUNTER_Mの引数にカウンタデバイスを使用してください。
59	CONCAT(_E)関数の引数と戻り値に同じ変数を使用しています。(F0196)	CONCAT(_E)関数の引数と戻り値に同じ変数を使用する。 例1：文字列ラベル ： Str1・Str2 Str1 := CONCAT(Str2,Str1);	CONCAT(_E)関数の引数と戻り値には異なる変数を使用して下さい。
60	INSERT(_E)関数の引数と戻り値に同じ変数を使用しています。(F0206)	INSERT(_E)関数の引数と戻り値に同じ変数を使用する。 例1：文字列ラベル ： Str1 Str1 := INSERT(Str1,Str2, D0); 引数と戻り値に同じ変数を使用した	INSERT(_E)関数の引数と戻り値には異なる変数を使用してください。
61	不正なデバイス種別が使用されています。(C10000)	不正なデバイス種別(タイマ・積算タイマ・カウンタ・ポインタ)を使用する。 例1：Timer1 := 0; デバイス種別タイマを使用した	不正なデバイス種別(タイマ・積算タイマ・カウンタ・ポインタ)は使用できません。使用可能なデバイス種別に変更してください。
62	指定されたデバイス，数値が範囲を超えている、または使用できません。(C10001)	デバイス番号が使用できる範囲を超えている。または使用できないデバイスが指定されている。または数値が使用できる範囲を超えている。 例1：M0 := X2000; Xのデバイス番号に1FFFを超えるデバイス番号を指定した 例2：D0 := A0; QCPU/LCPUでアキュムレータを使用した 例3：ダブルワード型ラベル：DW1 DW1 := K2147483648;	デバイス番号を使用できる範囲に修正してください。 または使用可能なデバイスに変更してください。または数値を使用できる範囲に修正してください。

No.	エラーメッセージ	原因	処置
63	関数または演算子の個数が多すぎます。(C10002)	1文内で関数または演算子が合わせて1025個以上使用されている。 例1：D0 := 1+1+1+1+・・・+1+1; 1文内で演算子"+"を1025個以上使用した 例2：d0 := ABS(ABS(ABS(ABS(wlabel) ・・・・)); 1文内で関数ABSを1025個以上使用した	1文内で関数または演算子を使用する場合，1025個未満となるよう修正してください。
64	配列要素指定のネストが多すぎます。(C10003)	配列要素指定で17個以上ネストした場合。 例1：Array1[Array1[Array1[Array1 [Array1[Array1[.....]]]]] ; 配列要素指定で17個以上ネストした	配列要素指定のネストは5個までに変更してください。6個以上は未サポートです。17個以上でエラーとなります。
65	指定されたFB名は既に使用されています。(C10004)	プログラム中にて，同じFB名で2回以上のFB呼出しを行う。 例1：流用FB名：FB1 入出力変数名：INOUT1 FB1(INOUT1 := D100); FB1(INOUT1 := D101);	同じFB名でのFB呼出しは1回のみとしてください。
66	出力変数がFBの呼び出し前で使用されています。(C10005)	FB出力をFB呼出しより前で行っている。 例1：流用FB名：FB1 入出力変数名 ： INOUT1 出力変数名 ： OUT1 D0 := FB1.OUT1; FB1(INOUT1 := D100);	FB出力は，FB呼出しの後へ変更してください。
67	"*1"で不正な型が使用されています。(C10006) (*1には演算子が入ります。)	ダブルワード,実数型の代入文や演算でデータ型の範囲を超える値を使用した。 例1：ダブルワード型ラベル：w_Dword w_Dword := -2147483649;	正しい範囲を指定してください。
68	関数"*1"で不正な型が使用されています。(C10007) (*1には関数名が入ります。)	MELSEC関数の引数に対して，不正なデータ型を使用した。 例1：RST_M(M0,ddev1); 関数RST_Mの第二引数にダブルワード型を指定した 例2：DECO_M(M0,Real1,K8,Real2); 関数DECO_Mの第二･四引数に実数型を指定した 例3：COMRD_S_MD(M0,ddev1, Str32); 関数COMRD_S_MDの第二引数にダブルワード型を指定した	引数には正しいデータ型の変数を使用してください。

7 - 12

付　録

付. 1　ラベル・FB名で使用できない文字列

STプログラミング時，ラベル・FB名として使用できない文字列を示します。

デバイス名，命令名，関数名で使用している文字列はラベル・FB名として使用できません。

下表に示す文字列を使用した場合は，登録／コンパイルを実行したときにエラーになります。

	ラベル・FB名で使用できない文字列
A	A, ACALL, ACJ, ACTION, ANB, ANY, ANY_BIT, ANY_DATE, ANY_DERIVED, ANY_ELEMENTARY, ANY_INT, ANY_MAGNITUDE, ANY_NUM, ANY_REAL, ANY_SIMPLE, ANY_STRING, ARRAY, AT
B	B, BEND, BL, BLOCK, BOOL, BOOL_TO_BYTE(DINT, DWORD, INT, REAL, SINT, UDINT, UINT, USINT, WORD), BY, BYTE, BYTE_TO_BOOL(DINT, DWORD, INT, REAL, SINT, STRING, UDINT, UINT, USINT, WORD), BWORKR, BWORKRP, BWORKW, BWORKWP, B_BCD_TO_DINT(INT, SINT)
C	C, CASE, CAL, CALC, CALCN, CONFIGURATION, CONSTANT, CTD, CTU, CTUD
D	D, DATE, DATE_AND_TIME, DINT, DINT_TO_BCD(BOOL, BYTE, DWORD, INT, REAL, SINT, STRING, TIME, UDINT, UINT, USINT, WORD), DO, DT, DWORD, DWORD_TO_BOOL(BYTE, DINT, INT, REAL, SINT, STRING, UDINT, UINT, USINT, WORD), DX, DY, D_BCD_TO_DINT(INT, SINT)
E	E, ELSE, ELSIF, EN, END, END_ACTION, END_CASE, END_FOR, END_FUNCTION, END_PROGRAM, END_IF, END_REPEAT, END_RESOURCE, END_STEP, END_STRUCT, END_TRANSITION, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EQ_STRING, EXIT
F	F, FALSE, FD, FOR, FROM, FUNCTION, FUNCTION_BLOCK, FX, FY, F_EDGE, F_TRIG
G	G, GE, GE_STRING, GT, GT_STRING
H	H
I	I, IF, INITIAL_STEP, INT, INT_TO_BOOL(BYTE, DINT, DWORD, REAL, SINT, STRING, UDINT, UINT, USINT, WORD)
J	J, JMPC, JMPCN
K	K
L	L, LDN, LE, LE_STRING, LIMIT_STRING, LINT, LREAL, LT, LT_STRING, LWORD
M	M, MAX_STRING, MIN_STRING, MOD, MPP, MPS, MRD
N	N, NE, NE_STRING, NOP, NOT
O	OF, ON, ORB, ORN
P	P, PROGRAM
Q	Q
R	R, R1, RCALL, RCJ, READ, READ_ONLY, READ_WRITE, REAL, REAL_TO_BOOL(BYTE, DINT, DWORD, INT, SINT, STRING, UDINT, UINT, USINT, WORD), RECV, REPEAT, RESOURCE, RETAIN, RETC, RETCN, RETURN, REQ, RS, R_EDGE, R_TRIG
S	S, SB, SD, SEND, SEL_STRING, SFCP, SFCPEND, SG, SINT, SINT_TO_BOOL(BYTE, DINT, DWORD, INT, REAL, STRING, UDINT, UINT, USINT, WORD), SM, SR, SREAD, ST, STEP, STEPC, STEPD, STEPG, STEPI, STEPID, STEPIR, STEPISC, STEPISE, STEPIST, STEPN, STEPR, STEPSC, STEPSE, STEPST, STN, STRING, STRING_TO_BYTE(DINT, DWORD, INT, REAL, SINT, TIME, UDINT, UINT, USINT, WORD), STRUCT, SW, SWRITE, SZ
T	T, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_STRING, TO, TOD, TOF, TON, TP, TR, TRAN, TRANA, TRANC, TRANCA, TRANCO, TRANCOC, TRANCOCJ, TRUNC_DINT(INT, SINT), TRANJ, TRANL, TRANO, TRANOA, TRANOC, TRANOCA, TRANOCJ, TRANOJ, TRANSITION, TRUE, TYPE

付 - 1

	ラベル・FB名で使用できない文字列
U	U,UDINT,UDINT_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UINT,UNTIL,USINT,WORD), UINT,UINT_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UDINT,USINT,WORD), ULINT,UNTIL,USINT,USINT_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UDINT,UINT,WORD)
V	V,VAR,VAR_CONSTANT,VAR_EXT,VAR_EXTERNAL,VAR_EXTERNAL_FB,VAR_EXTERNAL_PG, VAR_GLOBAL,VAR_GLOBAL_FB,VAR_GLOBAL_PG,VAR_IN_OUT,VAR_INPUT,VAR_OUTPUT,VAR_TEMP,VD,VOID
W	W,WHILE,WITH,WORD,WORD_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UDINT,UINT,USINT), WORKR,WORKRP,WORKW,WORKWP,WRITE,WSTRING,W_BCD_TO_DINT(INT,SINT)
X	X,XOR,XORN
Y	Y
Z	Z,ZNRF,ZR

ラベル定義をする場合の注意事項

1. スペースは使用できません。
2. 先頭文字に数字は使用できません。
3. 下記は使用できません。
   (,), ＊, /, ＋, －, <, >, ＝, &,
   !, ", #, $, %, ', ~, ^, |, @, `, [,], {, }, ;, :, , , ., ?, ¥, _
   アンダースコアは文字列の最後にある場合，あるいは2つ以上連続して使用されている場合のみエラーになります。
4. デバイス名は使用できません。
   デバイス名の後に0～Fの文字を付加した場合もエラーとなります。
   例）XFFF,M100
5. ラベル名に「EnDm」(例：E001D9)を使用しないでください。
   (n, mは，任意の数値を示します)
   実数値として認識されラベル名として使用できない場合があります。
6. 命令名（シーケンス命令，基本命令，応用命令，SFC命令など），関数名（MELSEC関数，IEC関数）は使用できません。

付 - 2

付. 2　GX DeveloperとGX Works2におけるST命令対応表

ST プログラムで使用できる命令は，GX DeveloperとGX Works2で異なります。
そのためGX Works2で作成したST プログラムを含むプロジェクトをGX Developer形式で保存し，GX Developerで読み出したときに，そのままコンパイルするとエラーになる場合があります。その場合は下表に従ってST プログラムを修正してください。

GX Works2	GX Developer
BACOS	BACOS_MD
BAND	BAND_MD
BASIN	BASIN_MD
BATAN	BATAN_MD
BCD	BCD_M
BCOS	BCOS_MD
BDSQR	BDSQR_MD
BIN	BIN_M
BKAND	BKAND_M
BKBCD	BKBCD_M
BKBIN	BKBIN_M
BKOR	BKOR_M
BKRST	BKRST_M
BKXNR	BKXNR_M
BKXOR	BKXOR_M
BMOV	BMOV_M
BRST	BRST_M
BSET	BSET_M
BSFL	BSFL_M
BSFR	BSFR_M
BSIN	BSIN_MD
BSQR	BSQR_MD
BTAN	BTAN_MD
BTOW	BTOW_MD
BXCH	BXCH_M
CML	CML_M
COM	COM_M
DATERD	DATERD_MD
DATEWR	DATEWR_MD
DBAND	DBAND_MD
DBCD	DBCD_M
DBIN	DBIN_M
DBL	DBL_M
DCML	DCML_M
DDEC	DDEC_M

GX Works2	GX Developer
DEC	DEC_M
DECO	DECO_M
DELTA	DELTA_M
DFLT	DFLT_M
DFRO	DFRO_M
DGRY	DGRY_M
DI	DI_M
DINC	DINC_M
DIS	DIS_M
DLIMIT	DLIMIT_MD
DMAX	DMAX_M
DMIN	DMIN_M
DNEG	DNEG_M
DOR	DOR_M
DRCL	DRCL_M
DRCR	DRCR_M
DROL	DROL_M
DROR	DROR_M
DSER	DSER_M
DSFL	DSFL_M
DSFR	DSFR_M
DSORT	DSORT_M
DSUM	DSUM_M
DTEST	DTEST_MD
DTO	DTO_M
DWSUM	DWSUM_M
DXCH	DXCH_M
DXNR	DXNR_M
DXOR	DXOR_M
DZONE	DZONE_MD
EI	EI_M
EMOD	EMOD_M
ENCO	ENCO_M
ENEG	ENEG_M
EREXP	EREXP_M

GX Works2	GX Developer
ESTR	ESTR_M
EVAL	EVAL_M
FLT	FLT_M
FMOV	FMOV_M
FROM	FROM_M
GBIN	GBIN_M
GRY	GRY_M
HOUR	HOUR_M
INC	INC_M
MIDR	MIDR_M
NDIS	NDIS_M
NEG	NEG_M
NUNI	NUNI_M
OUT	OUT_M
PLOW	PLOW_M
POFF	POFF_M
PSCAN	PSCAN_M
PSTOP	PSTOP_M
QCDSET	QCDSET_M
QDRSET	QDRSET_M
RCL	RCL_M
RCR	RCR_M
RFS	RFS_M
RND	RND_M
RSET	RSET_MD
RST	RST_M
SECOND	SECOND_M
SEG	SEG_M
SER	SER_M
SET	SET_M
SFL	SFL_M
SFR	SFR_M
SFT	SFT_M
SORT	SORT_M
SRND	SRND_M

(次ページへ)

付 - 3

GX Works2	GX Developer
STOP	STOP_M
SUM	SUM_M
SWAP	SWAP_MD
TEST	TEST_MD
UNI	UNI_M

GX Works2	GX Developer
WAND	WAND_M
WDT	WDT_M
WOR	WOR_M
WSUM	WSUM_M
WTOB	WTOB_MD

GX Works2	GX Developer
WXNR	WXNR_M
WXOR	WXOR_M
XCH	XCH_M
ZONE	ZONE_MD

付 - 4

索　引

索引 - 1

【E】

索引 - 2

【F】

【G】

【H】

【I】

【L】

【M】

【N】

【O】

【P】

【Q】

【R】

索引 - 3

【S】

【T】

【U】

【V】

【W】

索引 - 4

索引 - 5

索引 - 6

索引 - 7

索引 - 8

索引 - 9

保証について

ご使用に際しましては，以下の製品保証内容をご確認いただきますよう，よろしくお願いいたします。

１．無償保証期間と無償保証範囲

無償保証期間中に，製品に当社側の責任による故障や瑕疵（以下併せて「故障」と呼びます）が発生した場合，当社はお買い上げいただきました販売店または当社サービス会社を通じて，無償で製品を修理させていただきます。

ただし，国内および海外における出張修理が必要な場合は，技術者派遣に要する実費を申し受けます。

また，故障ユニットの取替えに伴う現地再調整・試運転は当社責務外とさせていただきます。

【無償保証期間】

製品の無償保証期間は，お客様にてご購入後またはご指定場所に納入後36ヵ月とさせていただきます。

ただし，当社製品出荷後の流通期間を最長6ヵ月として，製造から42ヵ月を無償保証期間の上限とさせていただきます。また，修理品の無償保証期間は，修理前の無償保証期間を超えて長くなることはありません。

【無償保証範囲】

(1) 一次故障診断は，原則として貴社にて実施をお願い致します。
ただし，貴社要請により当社，または当社サービス網がこの業務を有償にて代行することができます。この場合，故障原因が当社側にある場合は無償と致します。

(2) 使用状態・使用方法，および使用環境などが，取扱説明書，ユーザーズマニュアル，製品本体注意ラベルなどに記載された条件・注意事項などにしたがった正常な状態で使用されている場合に限定させていただきます。

(3) 無償保証期間内であっても，以下の場合には有償修理とさせていただきます。
①お客様における不適切な保管や取扱い，不注意，過失などにより生じた故障およびお客様のハードウェアまたはソフトウェア設計内容に起因した故障。
②お客様にて当社の了解なく製品に改造などの手を加えたことに起因する故障。
③当社製品がお客様の機器に組み込まれて使用された場合，お客様の機器が受けている法的規制による安全装置または業界の通念上備えられているべきと判断される機能・構造などを備えていれば回避できたと認められる故障。
④取扱説明書などに指定された消耗部品が正常に保守・交換されていれば防げたと認められる故障。
⑤消耗部品（バッテリ，リレー，ヒューズなど）の交換。
⑥火災，異常電圧などの不可抗力による外部要因および地震，雷，風水害などの天変地異による故障。
⑦当社出荷当時の科学技術の水準では予見できなかった事由による故障。
⑧その他，当社の責任外の場合またはお客様が当社責任外と認めた故障。

２．生産中止後の有償修理期間

(1) 当社が有償にて製品修理を受け付けることができる期間は，その製品の生産中止後7年間です。
生産中止に関しましては，当社テクニカルニュースなどにて報じさせていただきます。

(2) 生産中止後の製品供給（補用品も含む）はできません。

３．海外でのサービス

海外においては，当社の各地域FAセンターで修理受付をさせていただきます。ただし，各FAセンターでの修理条件などが異なる場合がありますのでご了承ください。

４．機会損失，二次損失などへの保証責務の除外

無償保証期間の内外を問わず，当社の責に帰すことができない事由から生じた障害，当社製品の故障に起因するお客様での機会損失，逸失利益，当社の予見の有無を問わず特別の事情から生じた損害，二次損害，事故補償，当社製品以外への損傷，およびお客様による交換作業，現地機械設備の再調整，立上げ試運転その他の業務に対する補償については，当社責務外とさせていただきます。

５．製品仕様の変更

カタログ,マニュアルもしくは技術資料などに記載の仕様は,お断りなしに変更させていただく場合がありますので,あらかじめご承知おきください。

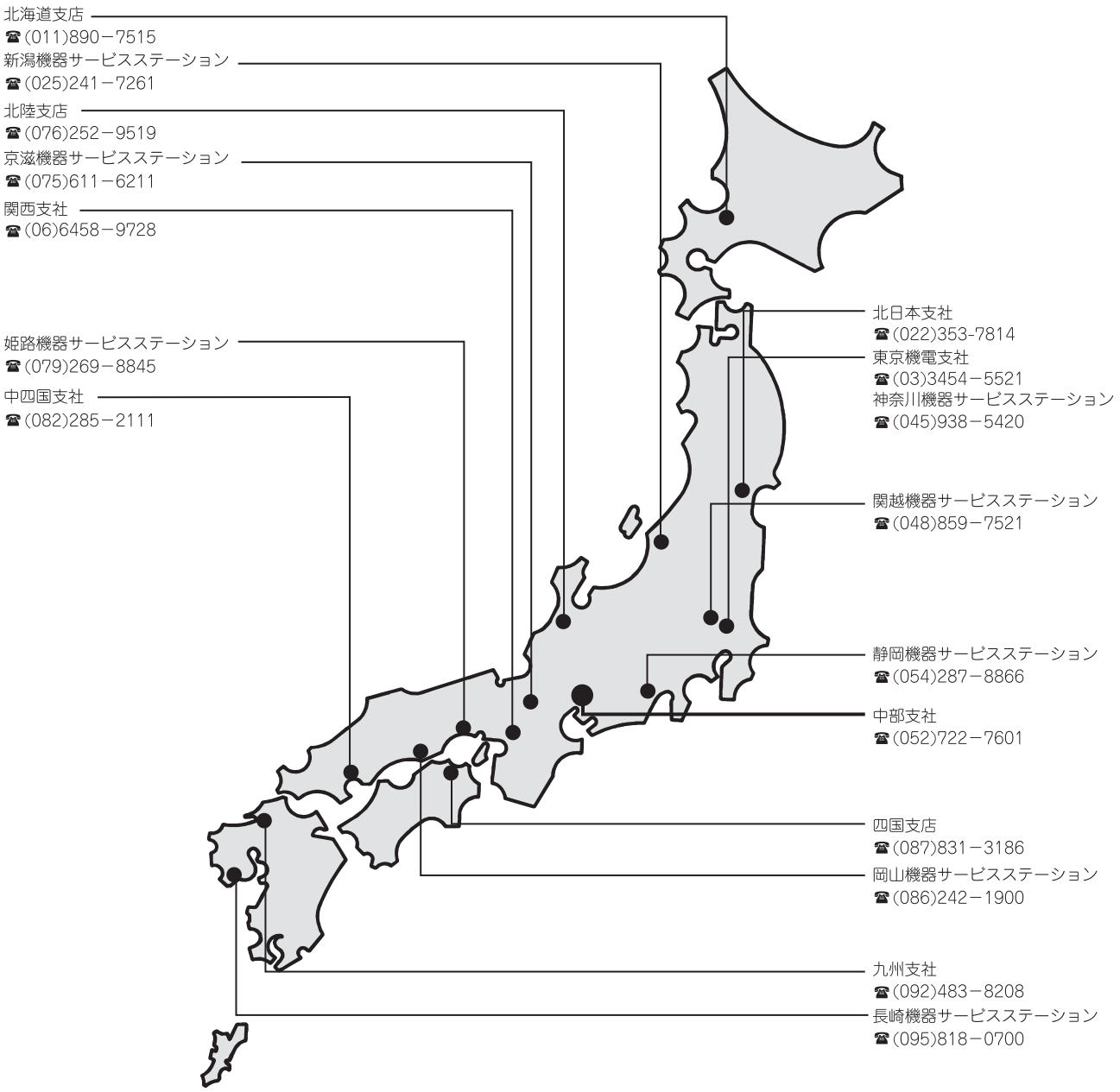
６．製品の適用について

(1) 当社シーケンサをご使用いただくにあたりましては，万一シーケンサに故障・不具合などが発生した場合でも重大な事故にいたらない用途であること，および故障・不具合発生時にはバックアップやフェールセーフ機能が機器外部でシステム的に実施されていることをご使用の条件とさせていただきます。

(2) 当社シーケンサは，一般工業などへの用途を対象とした汎用品として設計・製作されています。したがいまして，各電力会社殿の原子力発電所およびその他発電所向けなどの公共への影響が大きい用途や，鉄道各社殿および官公庁殿向けの用途などで,特別品質保証体制をご要求になる用途には,シーケンサの適用を除外させていただきます。
また，航空，医療，鉄道，燃焼・燃料装置，有人搬送装置，娯楽機械，安全機械など人命や財産に大きな影響が予測される用途へのご使用についても，当社シーケンサの適用を除外させていただきます。
ただし，これらの用途であっても，使途を限定して特別な品質をご要求されないことをお客様にご了承いただく場合には，適用可否について検討いたしますので当社窓口へご相談ください。

以　上

サービスネットワーク（三菱電機システムサービス株式会社）



三菱電機株式会社 〒100-8310 東京都千代田区丸の内2-7-3 (東京ビル)

お問い合わせは下記へどうぞ

本社機器営業部 …… 〒110-0016 東京都台東区台東1-30-7 (秋葉原アイマークビル) …… (03) 5812-1450
北海道支社 …… 〒060-8693 札幌市中央区北二条西4-1 (北海道ビル) …… (011) 212-3794
東北支社 …… 〒980-0013 仙台市青葉区花京院1-1-20 (花京院スクエア) …… (022) 216-4546
関越支社 …… 〒330-6034 さいたま市中央区新都心11-2 (明治安田生命さいたま新都心ビル) …… (048) 600-5835
新潟支店 …… 〒950-8504 新潟市中央区東大通2-4-10 (日本生命ビル) …… (025) 241-7227
神奈川支社 …… 〒220-8118 横浜市西区みなとみらい2-2-1 (横浜ランドマークタワー) …… (045) 224-2624
北陸支社 …… 〒920-0031 金沢市広岡3-1-1 (金沢パークビル) …… (076) 233-5502
中部支社 …… 〒450-6423 名古屋市中村区名駅3-28-12 (大名古屋ビルヂング) …… (052) 565-3314
豊田支店 …… 〒471-0034 豊田市小坂本町1-5-10 (矢作豊田ビル) …… (0565) 34-4112
静岡支店 …… 〒422-8067 静岡市駿河区南町14-25 (エスパティオビル) …… (054) 202-5630
関西支社 …… 〒530-8206 大阪市北区大深町4-20 (グランフロント大阪タワーA) …… (06) 6486-4122
中国支社 …… 〒730-8657 広島市中区中町7-32 (ニッセイ広島ビル) …… (082) 248-5348
四国支社 …… 〒760-8654 高松市寿町1-1-8 (日本生命高松駅前ビル) …… (087) 825-0055
九州支社 …… 〒810-8686 福岡市中央区天神2-12-1 (天神ビル) …… (092) 721-2247

三菱電機 FA 検索
www.MitsubishiElectric.co.jp/fa

メンバー登録無料!

インターネットによる情報サービス「三菱電機FAサイト」

三菱電機FAサイトでは、製品や事例などの技術情報に加え、トレーニングスクール情報や各種お問い合わせ窓口をご提供しています。また、メンバー登録いただくとマニュアルやCADデータ等のダウンロード、eラーニングなどの各種サービスをご利用いただけます。

三菱電機FA機器電話, FAX技術相談

●電話技術相談窓口 受付時間※1 月曜～金曜 9:00～19:00、土曜・日曜・祝日 9:00～17:00

対象機種			電話番号
自動窓口案内			052-712-2444
エッジコンピューティング製品		産業用PC MELIPC	052-712-2370※2
		Edgecross対応ソフトウェア (MTConnectデータコレクタを除く)	
シーケンサ	MELSEC iQ-R/Q/L/QnAS/AnSシーケンサ一般		052-711-5111
	MELSEC iQ-F/FXシーケンサ全般		052-725-2271※3
	ネットワークユニット/シリアルコミュニケーションユニット		052-712-2578
	MELSOFT シーケンサプログラミングツール	MELSOFT GXシリーズ	052-711-0037
	MELSOFT統合エンジニアリング環境	MELSOFT iQ Works (Navigator)	052-799-3591※2
	iQ Sensor Solution		
	MELSOFT通信支援ソフトウェアツール	MELSOFT MXシリーズ	052-712-2370※2
	MELSECパソコンボード	Q80BDシリーズなど	
	C言語コントローラ		
	MESインタフェースユニット/高速データロガーユニット		052-799-3592※2
	MELSEC計装/iQ-R/Q二重化	プロセスCPU/二重化CPU (MELSEC-Qシリーズ)	052-712-2830※2※3
		プロセスCPU/二重化機能SIL2プロセスCPU (MELSEC iQ-Rシリーズ)	
		MELSOFT PXシリーズ	
	MELSEC Safety	安全シーケンサ (MELSEC iQ-R/QSシリーズ)	052-712-3079※2※3
		安全コントローラ (MELSEC-WSシリーズ)	
	電力計測ユニット/絶縁監視ユニット	QEシリーズ/REシリーズ	052-719-4557※2※3
FAセンサ MELSENSOR		レーザ変位センサ	052-799-9495※2
		ビジョンセンサ	
表示器 GOT		GOT2000/1000シリーズ	052-712-2417
		MELSOFT GTシリーズ	

対象機種		電話番号
SCADA MC Works64		052-712-2962※2※6
サーボ/位置決めユニット/シンプルモーションユニット/モーションコントローラ/センシングユニット/組込み型サーボシステムコントローラ	MELSERVOシリーズ	052-712-6607
	位置決めユニット (MELSEC iQ-R/Q/L/AnSシリーズ)	
	シンプルモーションユニット (MELSEC iQ-R/iQ-F/Q/Lシリーズ)	
	モーションCPU (MELSEC iQ-R/Q/AnSシリーズ)	
	センシングユニット (MR-MTシリーズ)	
	シンプルモーションボード	
	C言語コントローラ インタフェースユニット (Q173SCCF)/ポジションボード	
	MELSOFT MTシリーズ/MRシリーズ/EMシリーズ	
センサレスサーボ	FR-E700EX/MM-GKR	052-722-2182
インバータ	FREQROLシリーズ	052-722-2182
三相モータ	三相モータ225フレーム以下	0536-25-0900※2※4
産業用ロボット	MELFAシリーズ	052-721-0100
電磁クラッチ・ブレーキ/テンションコントローラ		052-712-5430※5
データ収集アナライザ	MELQIC IU1/IU2シリーズ	052-712-5440※5
低圧開閉器	MS-Tシリーズ/MS-Nシリーズ	052-719-4170
	US-Nシリーズ	
低圧遮断器	ノーヒューズ遮断器/漏電遮断器/MDUブレーカ/気中遮断器 (ACB) など	052-719-4559
電力管理用計器	電力量計/計器用変成器/指示電気計器/管理用計器/タイムスイッチ	052-719-4556
省エネ支援機器	EcoServer/E-Energy/検針システム/エネルギー計測ユニット/B/NETなど	052-719-4557※2※3
小容量UPS (5kVA以下)	FW-Sシリーズ/FW-Vシリーズ/FW-Aシリーズ/FW-Fシリーズ	052-799-9489※2※6

お問い合わせの際には、今一度電話番号をお確かめの上、お掛け間違いのないようお願い致します。
※1：春季・夏季・年末年始の休日を除く ※2：土曜・日曜・祝日を除く ※3：金曜は17:00まで ※4：月曜～木曜の9:00～17:00と金曜の9:00～16:30
※5：受付時間9:00～17:00 (土曜・日曜・祝日・当社休日を除く) ※6：月曜～金曜の9:00～17:00

●FAX技術相談窓口 受付時間 月曜～金曜 9:00～16:00 (祝日・当社休日を除く)

対象機種	FAX番号
電力計測ユニット/絶縁監視ユニット (QEシリーズ/REシリーズ)	084-926-8340
三相モータ225フレーム以下	0536-25-1258※7
低圧開閉器	0574-61-1955

対象機種	FAX番号
低圧遮断器	084-926-8280
電力管理用計器/省エネ支援機器/小容量UPS (5kVA以下)	084-926-8340

三菱電機FAサイトの「仕様・機能に関するお問い合わせ」もご利用ください。
※7：月曜～木曜の9:00～17:00と金曜の9:00～16:30 (祝日・当社休日を除く)

SH(名)-080363-L(1902)KWIX
形名: QCPU-P-ST-J
形名コード: 13JC11

2019年2月作成
標準価格 3,000円