

ストラクチャードテキスト サンプルプログラム集

The screenshot displays the CX-Programmer software interface. The main window shows a structured text program titled "新規PLG1: 平均値をもとめる [FB スタートアップ]". The program includes a loop for calculating the average of a set of values. A comment block describes the program's purpose: "STサンプル : 半径と角度から移動後のX座標、Y座標を求める". The program uses the following logic:

```

合計値 := 0.0;
FOR i := 0 TO 平均する個数 - 1 DO
    合計値 := 合計値 + 加算値[i];
END_FOR;

(* *****
STサンプル : 半径と角度から移動後のX座標、Y座標を求める
***** *)

X座標 := 半径 * COS (DEG_TO_RAD (角度));
Y座標 := 半径 * SIN (DEG_TO_RAD (角度));

IF 角度 = 90.0 THEN
    X座標 := 0.0;
ELSIF 角度 = 180.0 THEN
    Y座標 := 0.0;
ELSIF 角度 = 270.0 THEN
    X座標 := 0.0;
ELSIF 角度 = 360.0 THEN
    Y座標 := 0.0;
END_IF;
  
```

The right-hand side of the screenshot shows a ladder logic diagram for the same program. It includes a timer T0 (0.00) and a network of logic involving a function block "長さ角度からXY座標を算出". The inputs to this block are "アーム長さ" (Arm Length) with a value of +100.0000 and "角度" (Angle) with a value of +60.0000. The outputs are "X座標" (X Coordinate) with a value of +50.0000 and "Y座標" (Y Coordinate) with a value of +86.60255.

はじめに

- ・本書内のサンプルプログラムをご使用になる前に、「CX-Programmerオペレーションマニュアル」の「はじめに」、「安全上のご注意」、「安全上の要点」を必ずお読みください。
- ・本書は、ファンクションブロック・ストラクチャードテキストの基本的な操作を説明しています。詳しい操作や機能については、CX-Programmerのヘルプやマニュアルを参照してください。
- ・本書に記載されているサンプルプログラムは実際の動作を保証するものではありません。なお、各プログラムはエラー処理が省略されているため、必要によりエラー処理を追加してください。
- ・PDFファイルで提供されているマニュアルを表示するには、Acrobat Reader 5.0以上がパソコンにインストールされていることが必要です。
- ・本書で使用している画面は予告無く変更することがあります。
- ・本書に記載されている各社の商品名・サービス名・機能名およびロゴは、各社の商標または登録商標です。
- ・本書では、各社の商標または登録商標に(R)、TMの記号を表示していません。
- ・本書では、各社の製品名を省略して記載することがあります。
- ・本書および本書内のサンプルプログラムは、以下のURLよりダウンロードできます。
<http://www.omron-cxone.com/seminar/>

目 次

第1章 はじめに

FB/STの基本知識、サポートソフトでの作成方法について説明します。

- ◆ ファンクションブロック(FB)、ストラクチャードテキスト(ST)とは 1-1
- ◆ ストラクチャードテキスト(ST)を使ったファンクションブロック(FB)の作成の仕方 ... 1-2

第2章 徹底比較！ラダーとSTプログラム

機器構成や外部機器からの入力データなどを例にラダーとSTを比較して以下のアプリケーション事例を説明します。

- ◆ 変位センサからのデータをアナログ入力ユニットに取り込んでスケールリングする 2-1
- ◆ 重量データからのデータをアナログ入力ユニットに取り込んで移動平均を求める 2-3
- ◆ バーコードリーダーからの受信データから文字列を抽出する 2-5
- ◆ ワークの高さを測定し良品・不良品を判定する 2-7
- ◆ X、Y座標から移動する長さや角度を求める 2-9

第3章 サンプルプログラム

STで使用する各種関数、ステートメントの使い方、FBとしての部品化についてサンプルプログラムで説明します。

四則演算・算術関数

- ◆ 数値の切り上げ、切り捨て、四捨五入 ～四則演算を使う①～ 3-3
- ◆ 移動量からモータへの指令パルス数を求める ～四則演算を使う②～ 3-4
- ◆ アナログデータをスケールリングする ～四則演算を使う③～ 3-5
- ◆ トータル時間(秒)から時・分・秒を求める ～MOD関数を使う～ 3-6
- ◆ 二点間の距離を求める ～平方根や指数関数を使う～ 3-7
- ◆ 半径と角度からX、Y座標を求める ～三角関数・ラジアン変換を使う～ 3-8

ステートメント(ST構文)

- ◆ 配列データの平均値を求める ～繰り返し処理(FOR文)を使う～ 3-9
- ◆ 配列データを並べ替える ～繰り返し処理(FOR文)と条件分岐(IF文)の組み合わせ～ 3-10
- ◆ 選択分岐を行う(電卓機能) ～選択分岐(CASE文)を使う～ 3-11

文字列関数

- ◆ 文字列の長さ(文字数)を検出する ～文字列操作関数(LEN関数)を使う～ 3-12
- ◆ 文字列を抽出する ～文字列操作関数(LEFT、RIGHT関数)を使う～ 3-13
- ◆ 文字列を抽出する ～文字列操作関数(MID関数)を使う～ 3-14
- ◆ 文字列を結合する① ～文字列操作関数(CONCAT関数)を使う～ 3-15
- ◆ 文字列を結合する② ～文字列操作関数(CONCAT関数、LEN関数)の組み合わせ～ 3-16
- ◆ 文字列を置換する ～文字列操作関数(REPLACE関数)を使う～ 3-17

PLC関連のプログラミングテクニック

- ◆ 年月日情報をPLCから収集する ～PLCの特殊補助リレーを使う～ 3-18
- ◆ ビットシフトを行う ～ビット操作関数(SHL関数、SHR関数)を使う～ 3-19
- ◆ ビット回転を行う ～ビット操作関数(ROL関数、ROR関数)を使う～ 3-20
- ◆ ST言語での微分回路 ～立ち上がり微分の作り方～ 3-21

既存のFB部品を組み合わせる(FBネスティング)

- ◆ 既存FB部品を別のFBから呼び出す ～FBネスティング～ 3-22

付録

- 付録1 変数の登録方法① グローバル変数／ローカル変数／FB内部変数の登録 付録-1
- 変数の登録方法② 配列変数、文字列変数の登録 付録-2
- 付録2 定数、文字列の記述方法 付録-3
- 付録3 ウォッチウィンドウ(値のモニタ、設定)への登録方法 付録-4
- 付録4 データ型一覧 付録-6
- 付録5 STの構成 付録-7

第1章 はじめに

FB/STの基本知識、サポートソフトでの作成方法について説明します。

- ◆ ファンクションブロック(FB)、ストラクチャードテキスト(ST)とは
- ◆ ストラクチャードテキスト(ST)を使ったファンクションブロック(FB)の作成の仕方

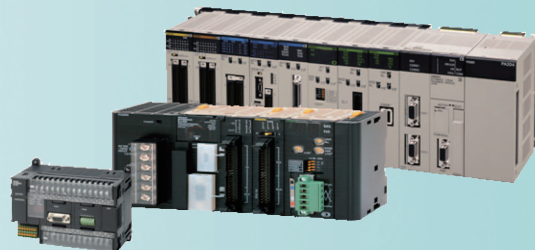
はじめに

ファンクションブロック(FB)、ストラクチャードテキスト(ST)とは

オムロンでは、SYSMAC CS/CJ/CPシリーズおよびFA統合ツールパッケージCX-Oneで、ファンクションブロック(以下FBと呼びます)によるプログラムの部品化・再利用機能やストラクチャードテキスト言語(ST言語、以下STと呼びます)などの各種言語に対応しています。

マイクロPLCからビルディングブロックPLCまですべてのPLC(※注1)で対応できます。アプリケーションの規模やI/O点数に関係なくプログラムの部品化とその再利用ができますのでプログラム設計生産性の向上が可能となります。ここではそれぞれの特長についてかんたんに説明します。

※注1：SYSMAC CP1EではFB/STが使用できません。



SYSMAC CP/CJ/CSシリーズ

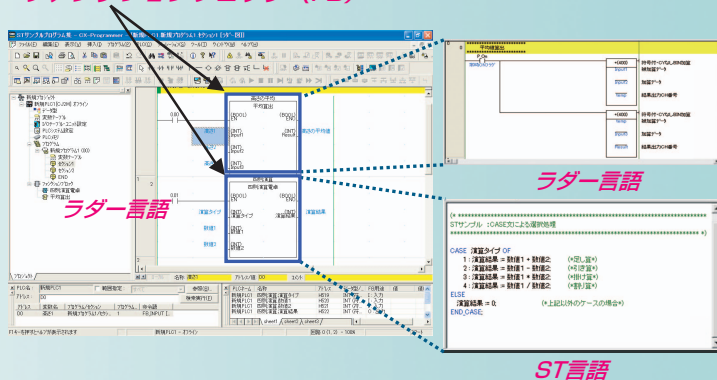
ファンクションブロック(FB)

今まではプログラムを流用し類似の機械を製造するとき、既存のプログラムをコピー＆ペーストして行っていたが、それでは流用後にアドレス変更が生じるため、流用箇所のデバッグが必要となっていました。

FBは、定型的な処理(機能)を1つのかたまりとして部品化し、流用時はそのFBに入力、出力のアドレスを与えるだけで再利用可能な機能部品です。さらに、稼働実績のあるプログラム部品として再利用できるためデバッグは不要となり、大幅な設計工数削減ができます。

FBの内部には、ラダー言語やST言語を選択して記述できますので、処理に応じた最適な言語が使用可能となります。

ファンクションブロック (FB)



ストラクチャードテキスト言語 (ST言語)

これまでのラダーでは、演算処理や情報処理のプログラムや配列処理、浮動小数点演算の応用命令を使ったプログラムなどは記述しにくく、またデバッグも大変でした。

これに対してSTで記述すると非常にかんたんに分かりやすくプログラミングすることができるようになります。

STでは、C言語やVisual Basic(VB)言語などのようなプログラミング方法でPLCの制御プログラムを記述できます。

構文もIF文(条件分岐)、FOR文(繰り返し処理)などや高級言語でみられる構文が使用できます。また、LEFT / RIGHT / MID関数やLEN関数などの文字列関数や、配列処理などのプログラミングも可能です(※注2)。

さらに、データ型変換命令(INT_TO_REALなど)も準備していますので、これまでよりも浮動小数点用の命令語を使用する際のデータ型変換が容易になります。

さらに、プログラムの変更履歴もコメントにより自由に行うことができ、視認性向上も可能です。

※注2：SYSMAC CPシリーズでは文字列関数、配列は使用できません。

(* *****

STサンプル：配列データの平均を算出する

- ・作成日：2009年6月4日
- ・改定日：2010年8月31日
- ・担当者：××設計課 ○○
- ・レビジョン：Rev.B

***** *)

```
合計値 := 0.0;  
FOR i := 0 TO 平均する個数 - 1 DO  
    合計値 := 合計値 + 加算値[i];  
END_FOR;
```

平均値 := 合計値 / INT_TO_REAL(平均する個数);

(* *****

STサンプル：文字列の抽出(LEFT関数、RIGHT関数)

***** *)

```
抽出データ左 := LEFT ( 受信データ, 抽出文字数 );  
抽出データ右 := RIGHT ( 受信データ, 抽出文字数 );
```

STプログラムのサンプル例

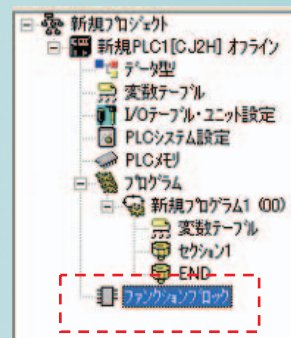
はじめに ストラクチャードテキスト(ST)を使ったファンクションブロック(FB)の作成の仕方

ここでは、本書のサンプルプログラムをCX-One(CX-Programmer)を使って作成する方法を説明します。

■FBの新規作成

1. CX-Programmerを起動し、PLC機種を指定し(ここでは、CJ2Hを指定)、新規作成します。
その後、右図のように、[ファンクションブロック]で右クリックし、[ファンクションブロック挿入]→[ストラクチャードテキスト]を選択すると、「ファンクションブロック1」が作成されます。

次に「ファンクションブロック1」で右クリックし[名前の変更]を選択します。ここでは「練習1」とします。



2. 作成したFB「練習1」をダブルクリックするとSTプログラムを作成する画面が表示されます。
また、ここにプログラムで使用する変数名、データ型、および用途(入力、出力、内部)なども指定します。



3. ここでは、1から指定した数字までの足し算を行うプログラムを作成します。作成するプログラムは右のとおりです。



プログラミングは以下を注意してください。

- ・代入文は、“=”ではなく“:=”(イコールの前にコロンが必要)
- ・行の最後はセミコロン;が必要
- ・FORやTOなどの構文は大文字でも小文字でも構いません。(構文などの予約語は青色に変わります)
- ・半角スペースやタブは行内でも自由に使用できます。全角スペースや全角の括弧“(”はエラーになります。

```
goukei := 0;
FOR i := 1 TO saidai DO
    goukei := goukei + i;
END_FOR;
```

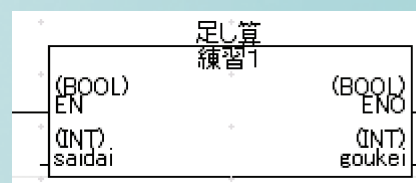
4. 次にプログラムで使った変数を登録します。
このプログラムでは、内部変数、入力変数、出力変数を使用します。それぞれの設定は以下になります。

変数種別	変数名	データ型
内部変数	i	INT
入力変数	saidai	INT
出力変数	goukei	INT



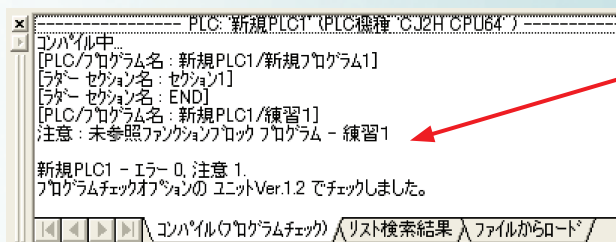
※変数名は半角でも全角でも使用できますが、使用した変数名とプログラムで使用する文字列を一致させてください。

5. 上記により、入力が“saidai”、出力が“goukei”という入出力をもったFB部品ができあがります(右図)。



作成したFB部品“練習1”

6. 作成したFBおよびSTプログラムの確認を行います。メニューから[PLC]→[全プログラムチェック]を選択します。表示されるエラー内容に従ってプログラムを修正していきます。以下のように“注意”だけであれば正しくプログラムできている状態です。

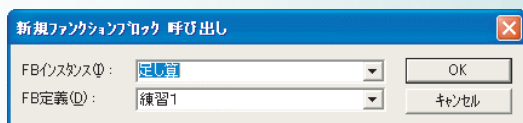
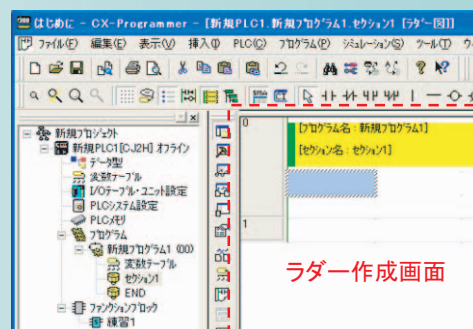


作成したFB部品“練習1”がまだラダープログラムで使用されていないため“注意”となっています。
FB部品の使用は次に説明します。

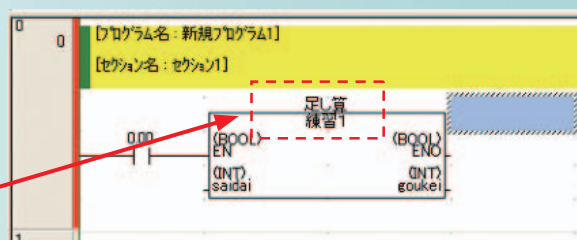
これで指定した数値まで足し算するFB部品のデバッグは完了です。

■作成したFB部品をラダーで使用する (FBのインスタンス化)

1. 作成したFB部品“練習1”をラダープログラムで使用します。“セクション1”をダブルクリックし右のようにラダー作成画面にします。
2. ラダー画面上でa接点(ここでは、アドレス“0.00”)を入力します。次に作成したFB部品“練習1”を入力します。
FBを入力する場合は“F”を入力します。ここでは、FB定義を“練習1”、FBインスタンス名に“足し算”と設定します。



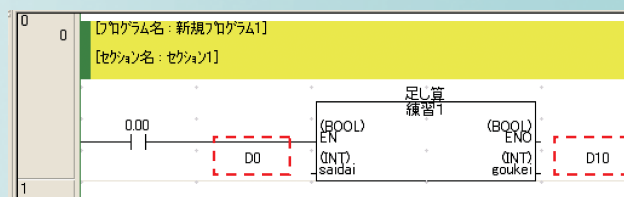
“練習1”というFB部品を“足し算”という処理で使用するという意味です。



3. 挿入したFBに対して入力アドレス、出力アドレスを設定します。ここでは、入力に“D0”、出力に“D10”を設定します。



※ここではFBへの入力・出力のパラメータを変数を使わずに設定しています。(データメモリの“D0”、“D10”を使用)本書では以降、入力・出力アドレスを変数名で設定して説明します。

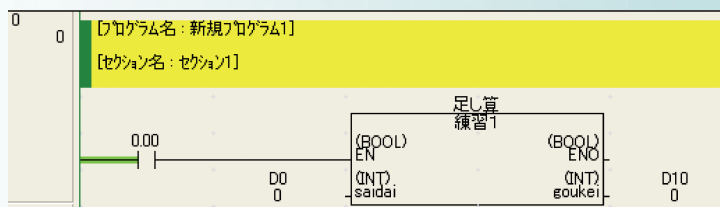


以上で、作成したファンクションブロック部品“練習1”をラダープログラム上でFBインスタンス名“足し算”という処理名で使用するところまで説明しました。

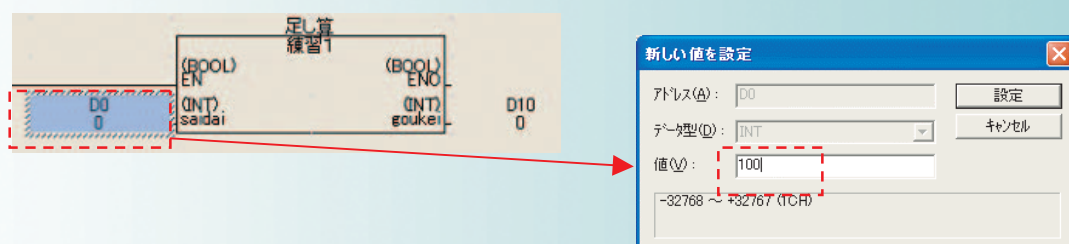
次に、ラダープログラム上で使用した“足し算”をデバッグする方法を説明します。
ここでは、ラダーシミュレーションソフト(CX-Simulator)を使用します。

■ラダーで使用したFBの動作確認をする (CX-Simulatorでの動作確認)

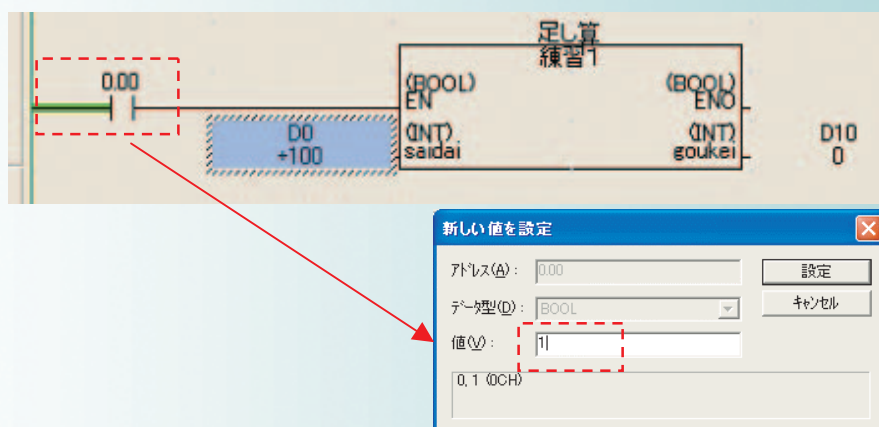
1. 作成したラダープログラムを実行する前に、再度プログラムのプログラムチェックを行います。メニューから[PLC]→[全プログラムチェック]を選択します。エラーがない状態から[シミュレーション]→[シミュレータオンライン接続]を選択します。プログラム転送の画面の後、プログラムが実行されている状態になります。



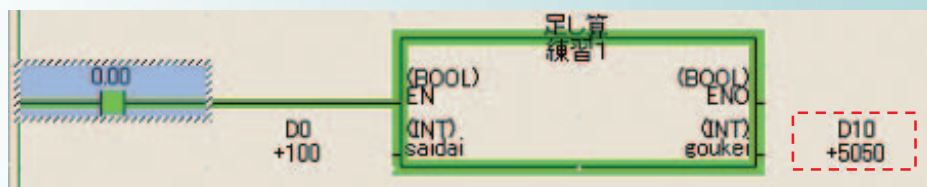
2. 次にFBへの入力パラメータ“D0”に数値を入力します。下記のように“D0”をダブルクリックします。値を設定するダイアログが表示されますので、ここでは“100”を設定します。



3. 下記状態から、FBを実行させるために、a接点“0.00”をONさせます。上記同様に、a接点上でダブルクリックし、“1”を設定します。



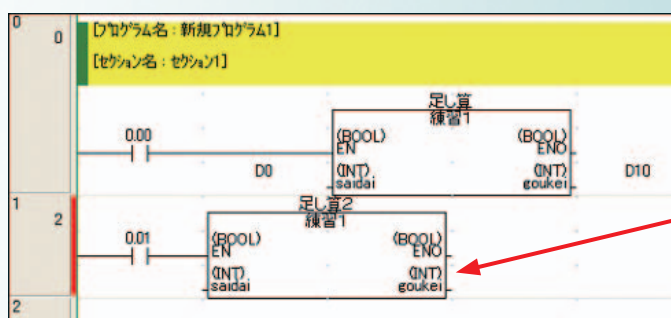
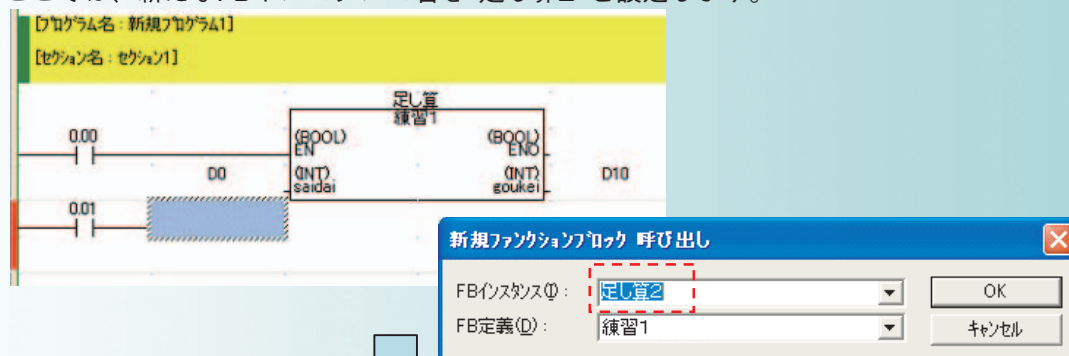
4. 下記のように、FBの出力パラメータ“D10”に計算結果が出力されます（1から100までの合計：5050）。



このように、シミュレーションによりFBの実行結果を確認することができます。

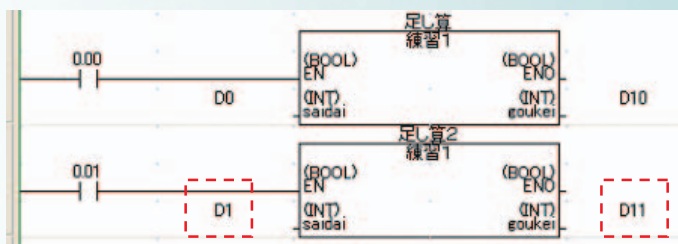
■既存のFBを流用する (新たなFBインスタンスを作る)

1. FB部品“練習1”を使って、新たな足し算処理を追加します。
下記のように、FBを起動する接点(ここではa接点 0.01)をおき、“F”を押し、“ファンクションブロック呼び出し”を行います。
ここでは、新たなFBインスタンス名を“足し算2”と設定します。

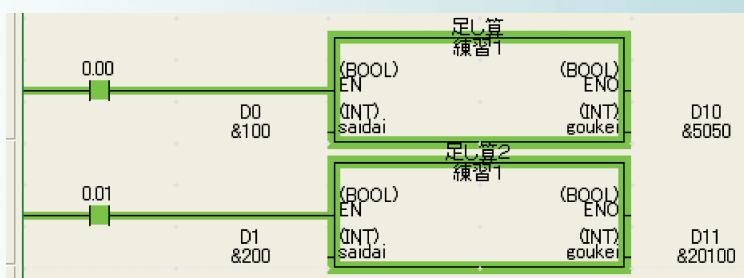


FB部品“練習1”を流用して、新たな足し算処理“足し算2”が追加されます。
この段階では、入力、出力のアドレスが設定されていないため、エラー状態(左母線が赤)となっています。

2. 次に新たに追加したFB(“足し算2”)に対して、入力、出力のアドレス(データメモリ)を設定します。
ここでは、入力アドレスに“D 1”、出力アドレスに“D11”を設定します。



3. 前ページ同様に、CX-Simulatorを使って動作確認を行います。
下記のように、新たに追加したFBの入力(データメモリの“D1”)に“200”を入れると、出力(データメモリの“D11”)に“20100”が出力されることが確認できます。



以上のように、プログラムをFBとして部品化しておき、その処理に対応した入力、出力アドレスを指定するだけで容易にプログラムの流用が行えるようになります。

第2章 徹底比較！ラダーとSTプログラム

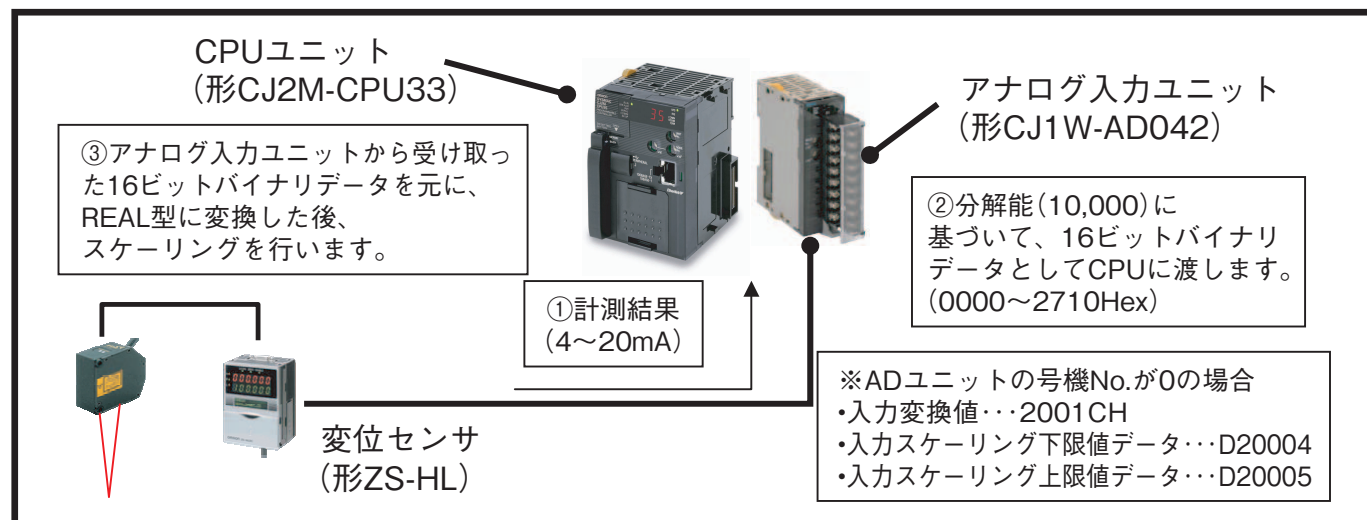
機器構成や外部機器からの入力データなどを例にラダーとSTを比較して以下のアプリケーション事例を説明します。

- ◆ 変位センサからのデータをアナログ入力ユニットに取り込んでスケーリングする
- ◆ 重量データからのデータをアナログ入力ユニットに取り込んで移動平均を求める
- ◆ バーコードリーダからの受信データから文字列を抽出する
- ◆ ワークの高さを測定し良品・不良品を判定する
- ◆ X、Y座標から移動する長さや角度を求める

変位センサからのデータをアナログ入力ユニットに取り込んでスケールリングする

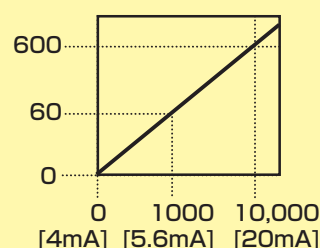
■概要と機器構成

変位センサの分解能を指定して、取り込んだアナログデータ(測定値)を、指定した変換後最大値、変換後最小値に基づきスケールリングします。



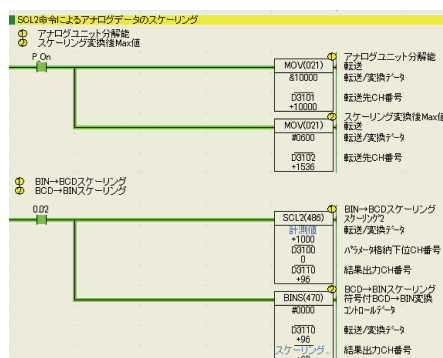
例) 分解能10,000、4~20mAレンジの変位センサの出力値(計測値:1000)をアナログ入力ユニットで取り込み、0~600で変換される場合

$$\frac{(\text{変換後最大値}(600) - \text{変換後最小値}(0))}{\text{分解能}(10000)} \times \text{計測値}(1000) + \text{変換後最小値}(0) = \text{スケールリング値}(60)$$



■ラダーでの記述

ラダーではスケールリング命令(SCL/SCL2など)を使用してスケールリングします。以下の例では、分解能:10,000、変換後のスケールリングを0~600としたとき、計測したアナログデータ(圧力生データ)が1,000の場合、スケールリング値が60となります。



■ラダーでの課題

- ・ラダーでは、SCL2命令のコントロールデータの設定を理解し、適正な数値をセットする必要があります。
- ・ラダーでは、数式のように記述できません。

第3章にも関連情報を記載していますのでご参照ください。

■外部機器からのデータ入出力

変位センサの分解能、測定データ、指定した変換後最大値/最小値を入力としてスケーリングします。



■ST内で使用する変数の定義

STプログラム内で使用する変数を定義します。

変数種別	変数名	データ型
内部変数	r分解能	REAL
	r変換後min	REAL
	r変換後max	REAL
	r測定値	REAL
	r変換後データ	REAL

変数種別	変数名	データ型
入力変数	分解能	INT
	測定値	INT
	変換後min	INT
	変換後max	INT

変数種別	変数名	データ型
出力変数	変換後データ	INT

■STプログラムでの記述

(* *****
STサンプル : アナログデータをスケーリングする
***** *)

```
r分解能 := INT_TO_REAL(分解能);
r変換後max := INT_TO_REAL(変換後max);
r変換後min := INT_TO_REAL(変換後min);
r測定値 := INT_TO_REAL(測定値);
r変換後データ := INT_TO_REAL(変換後データ);
```

CPU内部に取り込まれた16ビットバイナリデータをINT型として扱い、INT型からREAL型に変換します。

```
r変換後データ := ( r変換後max - r変換後min ) / r分解能 * r測定値 + r変換後min;
```

スケーリングします。

```
変換後データ := REAL_TO_INT(r変換後データ);
```

変換後データをREAL型からINT型に変換します。

■STでの効果

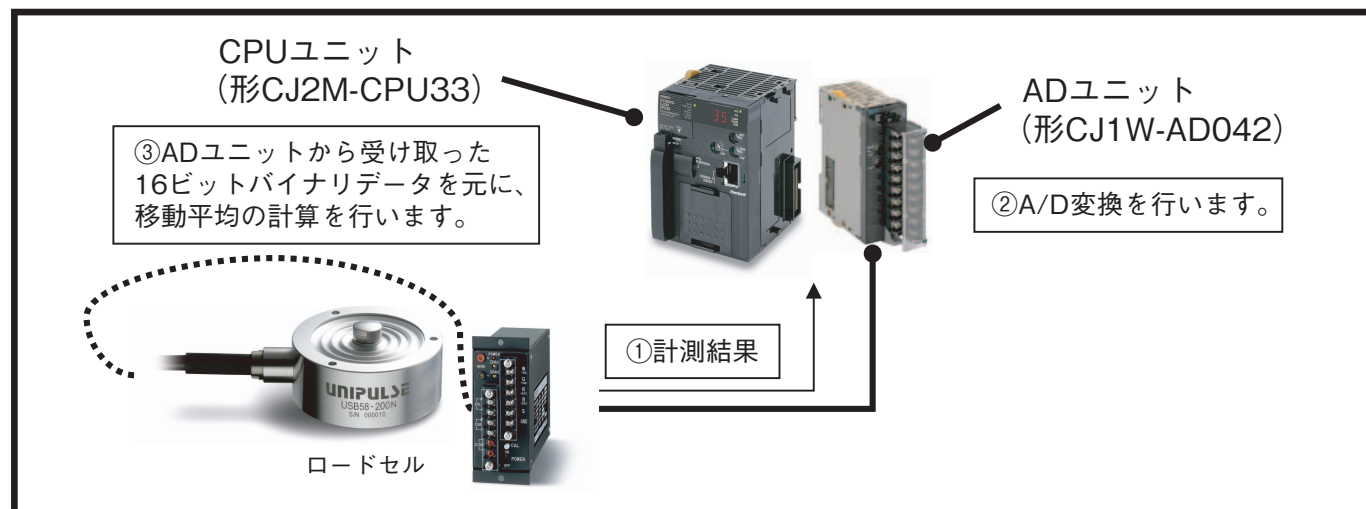


- ・数式のイメージで記述できます。
- ・メーカ毎に特有のラダーの応用命令、コントロールデータ等を理解する必要はありません。
- ・ラダーでは必要となった演算途中で使用するワークメモリの確保も必要ありません。

重量データからのデータをアナログ入力ユニットに取り込んで移動平均を求める

■概要と機器構成

重量計から取り込んだ重量データを指定した平均する個数に基づき移動平均値を算出します。移動平均とは、ある区間を定め、その区間の最新データの平均値を求めることをいいます。

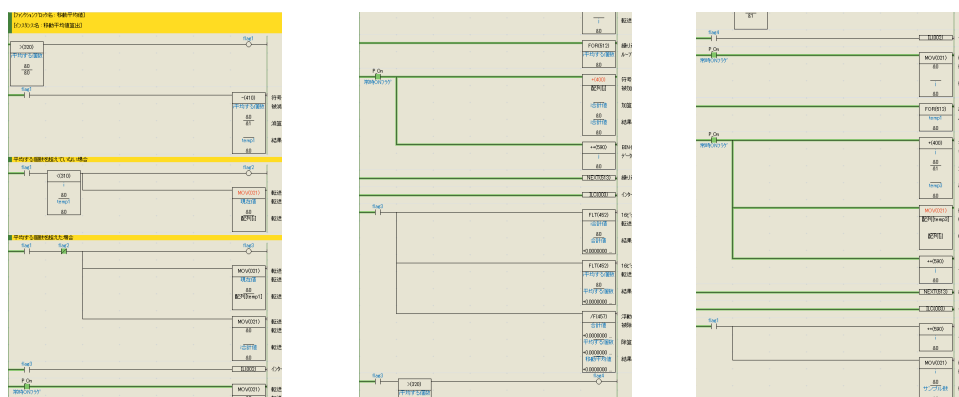


例) 平均区間数が5の場合



■ラダーでの記述

- ・ラダーでは、条件分岐の処理は比較命令やIL命令などを駆使しなければなりません。
- ・ラダーでは、数式のように記述できません。

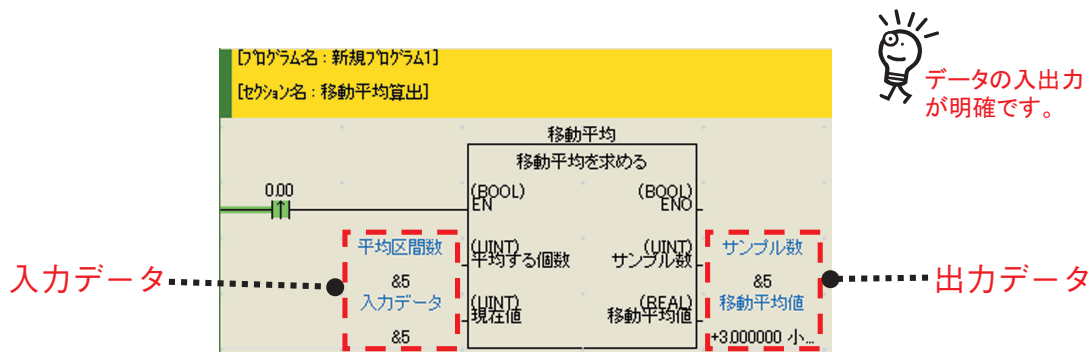


■ラダーでの課題

- ・ラダーでは、条件分岐の処理を比較命令やIL命令を使用するため、処理の流れが把握しにくい。
- ・ラダーでは、数式では容易に記述できる平均の算出も複雑なプログラムになります。

■外部機器からの入出力

重量データと平均区間数を入力として、移動平均値を求めて出力します。



データの入出力が明確です。

■ST内で使用する変数の定義

STプログラム内で使用する変数を定義します。

変数種別	変数名	初期値	データ型
内部変数	配列	—	UINT(10)
	i	0	UINT
	j	—	UINT
	合計値	—	UINT

※配列データの設定の仕方は、「付録1」を参照してください。

変数種別	変数名	データ型
入力変数	平均する個数	UINT
	現在値	UINT

変数種別	変数名	データ型
出力変数	サンプル数	UINT
	移動平均値	REAL

■STプログラム

```

(* *****
STサンプル : アナログ入力ユニットからの重量データの移動平均を求める
***** *)
IF 平均する個数 > 0 THEN
    IF i < 平均する個数 - 1 THEN
        (* 平均する個数を超えていない場合 *)
        (* 変数に登録だけでFBを抜ける *)
        配列[i] := 現在値;
        移動平均値 := 0.0;
    ELSE
        (* 平均する個数を超えた場合 *)
        (* 最新の値を最終の配列に格納する *)
        (* 合計値を初期化する *)
        配列[平均する個数 - 1] := 現在値;
        合計値 := 0;
        FOR j := 0 TO 平均する個数 - 1 DO
            (* 平均する個数分、値を合計する *)
            合計値 := 配列[j] + 合計値;
        END_FOR;
        移動平均値 := UINT_TO_REAL(合計値) / UINT_TO_REAL(平均する個数);
        IF 平均する個数 > 1 THEN
            FOR j := 0 TO 平均する個数 - 2 DO
                配列[j] := 配列[j+1];
            END_FOR;
        END_IF;
        配列[平均する個数 - 1] := 現在値;
        i := i + 1;
        サンプル数 := i;
    END_IF;
END_IF;

```

■STでの効果



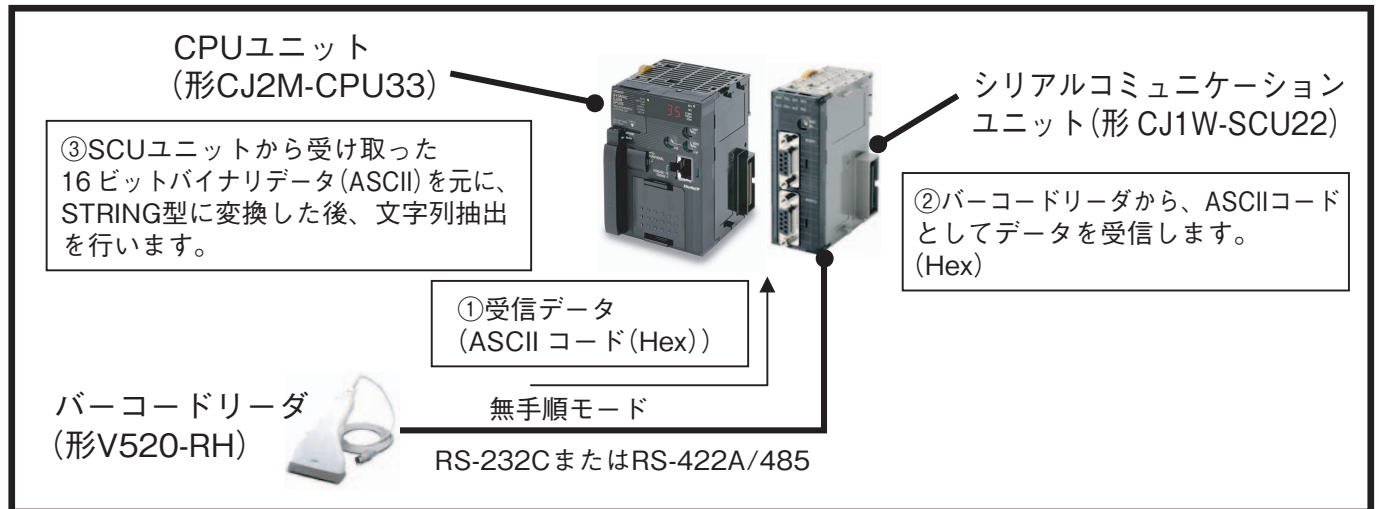
※本プログラムは複数スキャンに渡って処理が完了しますのでスキャンタイムが延びる可能性があります。
 ※平均する個数は1～10を指定できます。
 ※入力データを取り込むタイミングは、0.00がOFFからONになった時です。
 ※サンプル数が平均する個数よりも小さい場合、移動平均値は0.0になります。

- ・IF～THEN～ELSE～END_IFやFOR～END_FORなど高級言語のような記述ができます。
- ・数式のイメージで、複数にわたるデータを配列を使って簡単に記述できます。

バーコードリーダからの受信データから文字列を抽出する

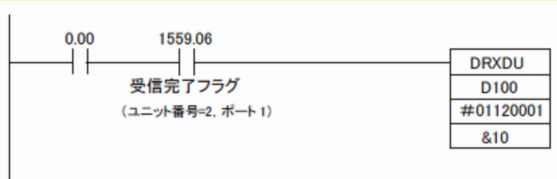
■概要と機器構成

バーコードリーダからの受信データをシリアルコミュニケーションユニット(SCU)で取り込み、取り込んだ受信データ(ASCII)を指定した文字数に応じて文字列を抽出します。



<外部機器からPLCへの受信時>

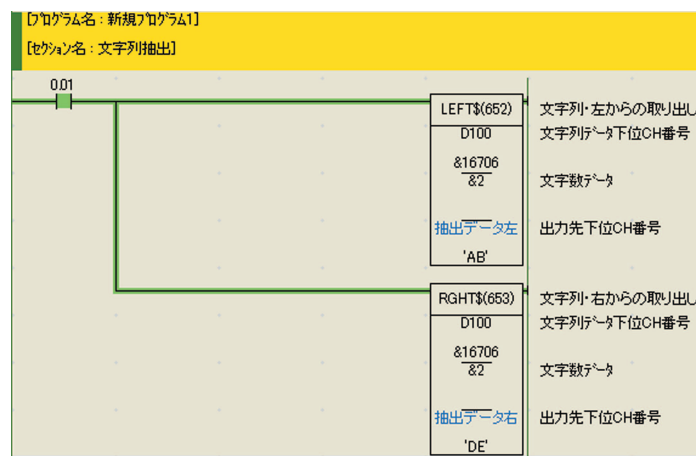
0.00がONのとき受信完了フラグ(1559.06)がONすると、ユニット番号2のシリアルコミュニケーションユニットポート1に接続された機器からの受信データを、そのまま無変換で、D100の下位バイトから10バイト格納します。



注1:DRXDU命令は、CJ2 CPUユニットVer.1.1以降とCJ1W-SCU22/32/42の組み合わせで使用可能です。
注2:ユニット宛てのDRXDU命令の場合、入力条件に、受信完了フラグのa接点をAND条件として組んでください。

■ラダーでの記述

接点0.01がONすると、SCUユニットからのデータを、指定した範囲で文字列抽出します。下の例は受信データ"ABCDE"の時に、2バイト分抽出する例になります。



■ラダーでの課題

- ・ラダーでは、受信データはASCIIコードのため、文字列に変換する必要があります。

第3章にも関連情報を記載していますのでご参照ください。

■外部機器からのデータ入出力

バーコードリーダからの受信データと抽出文字数を入力とし、抽出文字数で指定された分だけ抽出データを出力します。



■ST内で使用する変数の定義

STプログラム内で使用する変数を定義します。

変数種別	変数名	データ型
入出力変数	受信データ	STRING(20)
	抽出データ左	STRING(20)
	抽出データ右	STRING(20)

変数種別	変数名	データ型
入力変数	抽出文字数	UINT

■STプログラム

```
(* *****  
STサンプル : SCUユニットからの文字列データを抽出する  
***** *)
```

```
抽出データ左 := LEFT ( 受信データ, 抽出文字数);  
抽出データ右 := RIGHT ( 受信データ, 抽出文字数);
```

■STでの効果

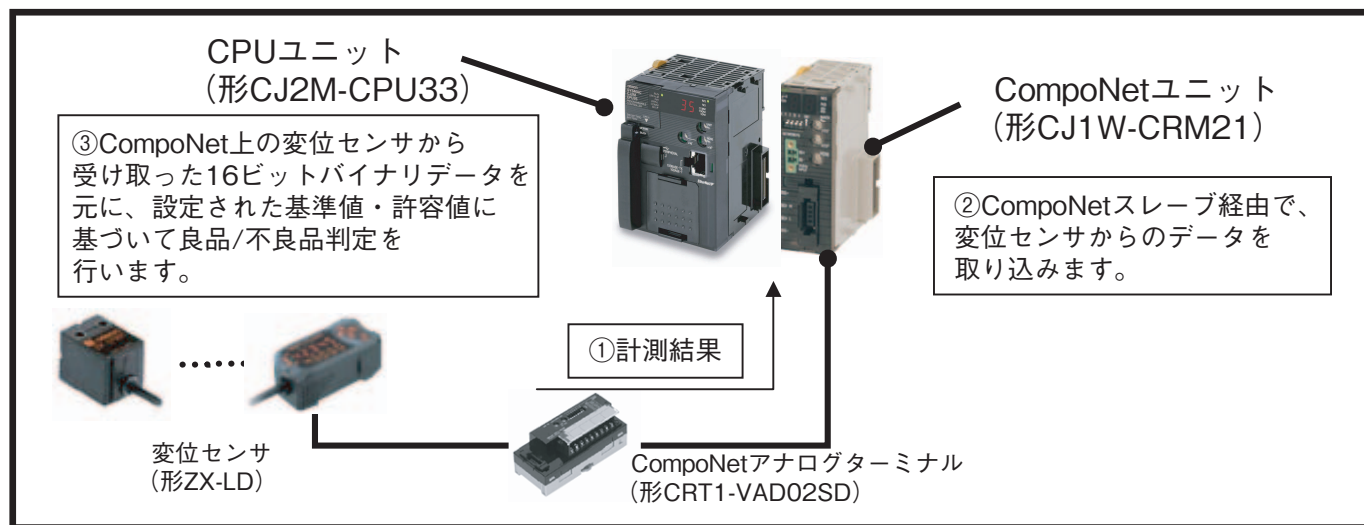


- ・文字列操作が高級言語のプログラミングと同様に行えます。

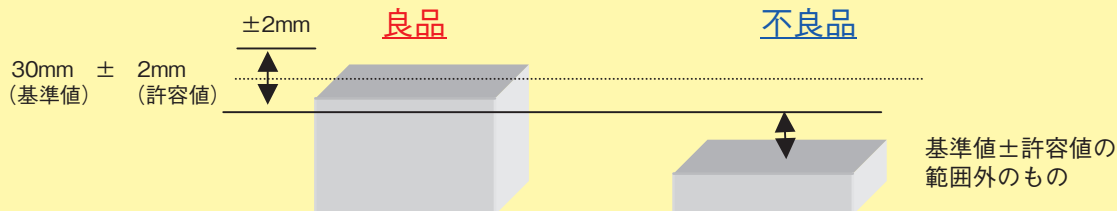
ワークの高さを測定し良品・不良品を判定する

■概要と機器構成

CompoNet上の変位センサで計測したデータをCPUユニットに取り込み、設定された基準値・許容値に基づいて良品・不良品の判定を行います。

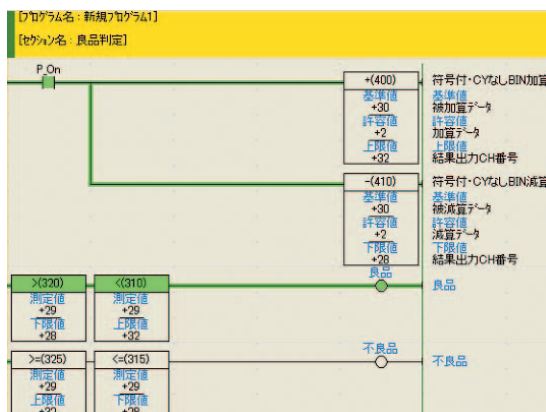


例)高さの基準値が30mm・許容値が2mmの場合



■ラダーでの記述

基準値・許容値を設定し、計測した比較値が上下限值内にあるかどうかで良品/不良品判定を行います。以下は、基準値30mm、許容値2mmの時に比較値が31mmの場合の例になります。

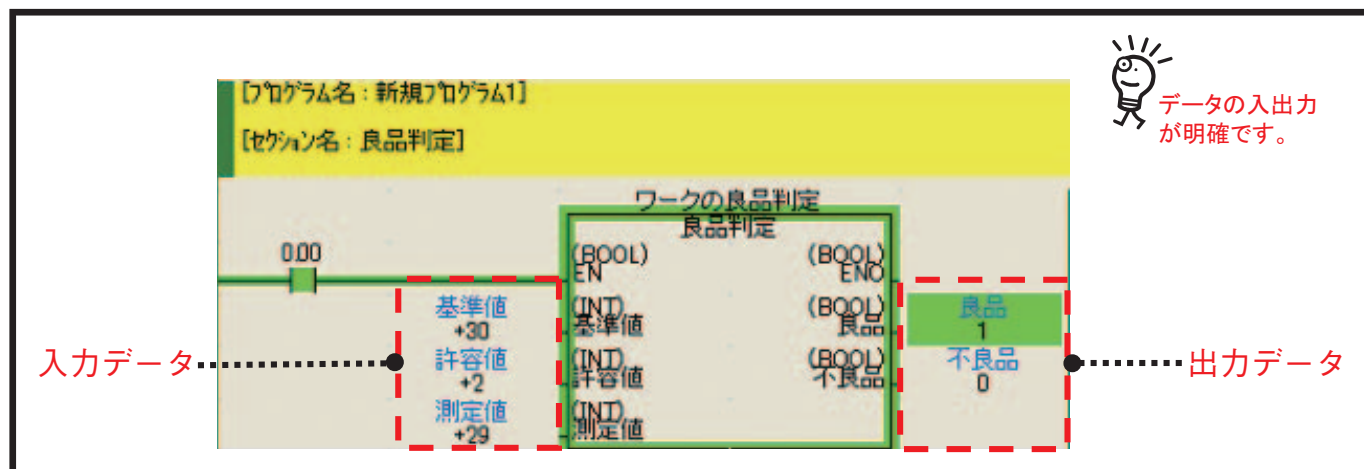


■ラダーでの課題

・ラダーでは、範囲内かどうかを検出する場合、複数の比較命令を使用しなければなりません。

■外部機器からのデータ入出力

変位センサで計測したデータ(測定値)を入力とし、良品/不良品の判定結果(良品・不良品)を出力します。



■ST内で使用する変数の定義

STプログラム内で使用する変数を定義します。

変数種別	変数名	データ型
内部変数	上限値	INT
	下限値	INT

変数種別	変数名	データ型
入力変数	基準値	INT
	許容値	INT
	比較値	INT

変数種別	変数名	データ型
出力変数	良品	BOOL
	不良品	BOOL

■STプログラム

```
(* *****
STサンプル : 良品/不良品判定を行う
***** *)

上限値 := 基準値 + 許容値;
下限値 := 基準値 - 許容値;

(* 比較値との上下限チェック *)
IF (比較値 > 下限値) AND (比較値 < 上限値) THEN      (* 下限値<比較値<上限値であれば良品 *)
  良品 := TRUE;
  不良品 := FALSE;
ELSE                                                  (* そうでなければ不良品 *)
  不良品 := TRUE;
  良品 := FALSE;
END_IF;
```

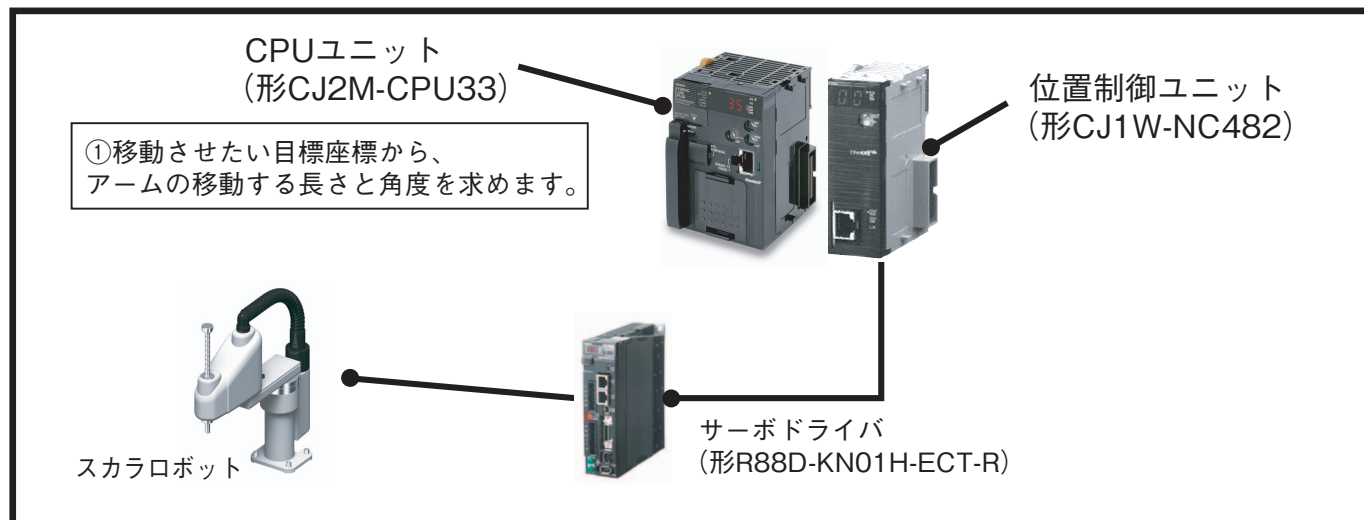
■STでの効果

- ・数式のイメージで記述できます。
- ・メーカー毎に特有のラダーの応用命令、コントロールデータ等を理解する必要はありません。
- ・条件分岐のプログラムが非常に組みやすくなります。

X、Y座標から移動する長さと角度を求める

■概要と機器構成

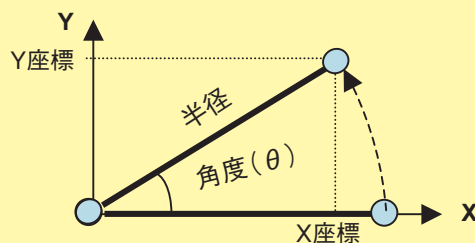
移動させたい目標のX、Y座標から、スカラロボットのアームの移動する長さと角度を求めます。



X/Y座標から半径・角度を求める公式

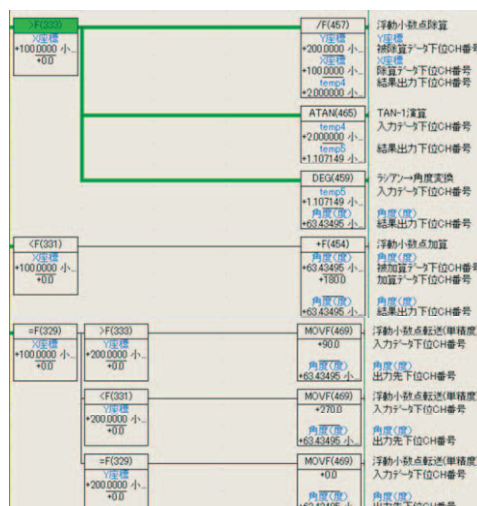
$$\text{半径} = \sqrt{X\text{座標}^2 + Y\text{座標}^2}$$

$$\text{角度} = \text{ATAN}\left(\frac{Y\text{座標}}{X\text{座標}}\right) \quad (\text{角度:RAD})$$



■ラダーでの記述

べき乗や三角関数、浮動小数点命令などの応用命令を使用し、ロボットのアームの長さと移動角度を算出します。

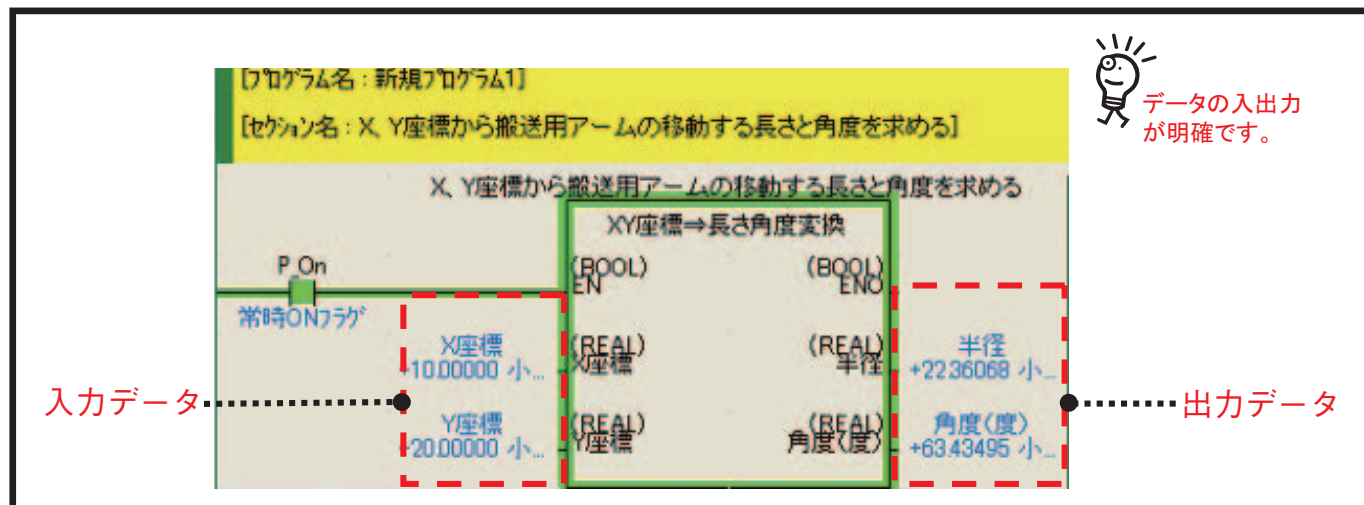


■ラダーでの課題

- ・ラダーでは、演算結果を一時的に格納するためのテンポラリのメモリが必要になります。
- ・ラダーでは、各命令のコントロールデータの設定を理解し、適正な数値をセットする必要があります。
- ・ラダーでは、数式のように記述できません。

■外部機器からのデータ入出力

移動させたい目標のX、Y座標を入力とし、長さと角度に変換して出力します。



■ST内で使用する変数の定義

STプログラム内で使用する変数を定義します。

変数種別	変数名	データ型
入力変数	X座標	REAL
	Y座標	REAL

変数種別	変数名	データ型
出力変数	半径	REAL
	角度	REAL

■STプログラム

(* *****)
STサンプル : X/Y座標から、長さと角度を求める
(* *****)

半径 := SQRT (X座標 ** 2 + Y座標 ** 2);

IF X座標 > 0.0 THEN (* X座標がプラスのとき *)

角度 := RAD_TO_DEG (ATAN(Y座標 / X座標));

ELSIF X座標 < 0.0 THEN (* X座標がマイナスのとき *)

角度 := RAD_TO_DEG (ATAN(Y座標 / X座標)) + 180.0;

ELSE (* X座標が0.0のとき *)

IF Y座標 > 0.0 THEN

角度 := 90.0;

ELSIF Y座標 < 0.0 THEN

角度 := 270.0;

ELSE

角度 := 0.0;

END_IF;

END_IF;

X座標が0.0の場合の例外処理です。
Y座標が0.0の場合の例外処理も、X座標の例外処理を参考に、
必要に応じて追加します。

■STでの効果

- ・数式のイメージで記述できます。
- ・メーカー毎に特有のラダーの応用命令、コントロールデータ等を理解する必要はありません。
- ・ラダーでは必要となったワークメモリの確保の必要がありません。

第3章 サンプルプログラム

STで使用する各種関数、ステートメントの使い方、FBとしての部品化についてサンプルプログラムで説明します。

四則演算・算術関数

- ◆ 数値の切り上げ、切り捨て、四捨五入 ～四則演算を使う①～
- ◆ 移動量からモータへの指令パルス数を求める ～四則演算を使う②～
- ◆ アナログデータをスケーリングする ～四則演算を使う③～
- ◆ トータル時間(秒)から時・分・秒を求める ～MOD関数を使う～
- ◆ 二点間の距離を求める ～平方根や指数関数を使う～
- ◆ 半径と角度からX、Y座標を求める ～三角関数・ラジアン変換を使う～

ステートメント(ST構文)

- ◆ 配列データの平均値を求める ～繰り返し処理(FOR文)を使う～
- ◆ 配列データを並べ替える ～繰り返し処理(FOR文)と条件分岐(IF文)の組み合わせ～
- ◆ 選択分岐を行う(電卓機能) ～選択分岐(CASE文)を使う～

文字列関数

- ◆ 文字列の長さ(文字数)を検出する ～文字列操作関数(LEN関数)を使う～
- ◆ 文字列を抽出する ～文字列操作関数(LEFT、RIGHT関数)を使う～
- ◆ 文字列を抽出する ～文字列操作関数(MID関数)を使う～
- ◆ 文字列を結合する① ～文字列操作関数(CONCAT関数)を使う～
- ◆ 文字列を結合する② ～文字列操作関数(CONCAT関数、LEN関数)の組み合わせ～
- ◆ 文字列を置換する ～文字列操作関数(REPLACE関数)を使う～

PLC関連のプログラミングテクニック

- ◆ 年月日情報をPLCから収集する ～PLCの特殊補助リレーを使う～
- ◆ ビットシフトを行う ～ビット操作関数(SHL関数、SHR関数)を使う～
- ◆ ビット回転を行う ～ビット操作関数(ROL関数、ROR関数)を使う～
- ◆ ST言語での微分回路 ～立ち上がり微分の作り方～

既存のFB部品を組み合わせる(FBネスティング)

- ◆ 既存FB部品を別のFBから呼び出す ～FBネスティング～

本サンプルプログラムは弊社ホームページよりダウンロードできます。

URL : <http://www.omron-cxone.com/seminar/>

なお、本書に掲載されているサンプルプログラムは実際の動作を保証するものではありません。

サンプルプログラムの読み方

本書では実際のアプリをイメージしたストラクチャードテキスト(ST)サンプルプログラムとして掲載しました。また、記載したSTサンプルプログラムはファンクションブロック(FB)として部品化し、それを再利用した形で記載しています。

掲載されているFBを使用するときは、FBへのアドレス割付を実際の機器で使われているアドレスを指定します。

STサンプル(アプリ例)のタイトル

使用している関数、ステートメント

サンプルプログラムの概要

一般的な公式や演算式や図表

FBによる部品化とFBへのアドレス割付の使用例

FB内部で使用している変数とFBの実行結果

STサンプルプログラム

四則演算・算術関数

半径と角度からX、Y座標を求める ～三角関数・ラジアン変換を使う～

SIN/COS (数値演算・算術関数: 三角関数)

ラジアンを引数として角度の正弦値(SIN)、余弦値(COS)をもとめます。

DEG TO RAD (変換関数)

度で指定した角度をラジアンに変換します。

■サンプルプログラムの概要

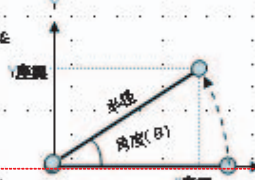
ロボットアームなどの回転体において、回転体の長さ(半径)と回転する角度を与えたときの移動後のX座標、Y座標を求めます。
公式は以下の通りです。

$$X座標 = 半径 \times \cos(\theta)$$

$$Y座標 = 半径 \times \sin(\theta)$$

下記の例では、アーム長さ(半径)が1000mm、回転角度が30度の場合です。

変数名	アドレス	値
アーム長さ	D400	1000
角度	D402	30



変数名	アドレス	値
X座標	D410	866.02
Y座標	D412	500.00

■変数の定義(FB内部)

変数種類	変数名	データ型
入力変数	半径	REAL
	角度	REAL
出力変数	X座標	REAL
	Y座標	REAL

■FBの実行結果



■STプログラム

```

(*:*****:*****:*****:*****:*****:*****:*****:*****:*****:*****:*)
STサンプル: 半径と角度から移動後のX座標、Y座標を求める
*****:*****:*****:*****:*****:*****:*****:*****:*****:*****:*)

X座標 := 半径 * COS ( DEG_TO_RAD ( 角度 ) );
Y座標 := 半径 * SIN ( DEG_TO_RAD ( 角度 ) );

IF 角度 = 90.0 THEN
    X座標 := 0.0;
ELSEIF 角度 = 180.0 THEN
    Y座標 := 0.0;
ELSEIF 角度 = 270.0 THEN
    X座標 := 0.0;
ELSEIF 角度 = 360.0 THEN
    Y座標 := 0.0;
END_IF;
        
```

DEG_TO_RAD 関数で角度変換します。

 基準上の位置に対する例外処理をします。
 (1回転以内の場合)

STサンプルプログラムの補足説明

FB外部の変数(アドレス)

サンプルプログラムのFBに記載されているFBへの入力、出力の変数とそのアドレスを記載しています。
FBへの入力や出力のアドレスは、グローバル変数テーブル、またはローカル変数テーブルに登録します。



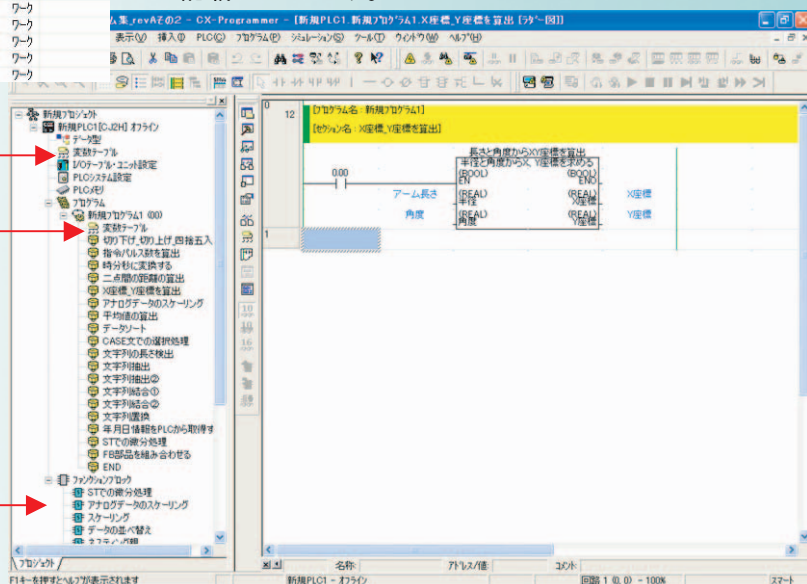
FB外部に指定するアドレスは変数名、あるいはアドレス(データメモリのD200など)、どちらでも設定可能です。

倍長データ、文字列データ、配列データなどの複数アドレスを使う場合は変数で登録し、変数のデータ長やデータ型を指定します。これにより、指定した型でデータの値を設定したりモニタすることができます。

名称	データ型	アドレス / 値	ラック位置	用途	コメント
サンプル数	UINT	D10		ワーク	
平均値	REAL	D12		ワーク	
加算値	REAL[D10]	D20		ワーク	
減値	UINT	D60		ワーク	
切り下げ値	UINT	D61		ワーク	
切り上げ値	UINT	D62		ワーク	
四捨五入値	UINT	D63		ワーク	
モータ分解能	REAL	D60		ワーク	
目標移動量	REAL	D62		ワーク	
モータ1回転あたりの移動	REAL	D64		ワーク	
指令パルス数	REAL	D70		ワーク	
演算タイプ	INT	D80		ワーク	
減値1	INT	D81		ワーク	

ラダープログラムでのFB使用

作成したFB部品をラダープログラムで使用します。
FBの入力、出力アドレスは、FB外部の変数(アドレス)”に記載しています。



変数の定義(FB内部)

FB内部で使用する変数名です。
FB内部のプログラムとセットで作成します。

名称	データ型	AT	初期値	保持	コメント
EN	BOOL		FALSE		ファンクションブロック実行制御フラグ
半径	REAL		0.0		
角度	REAL		0.0		

内部変数	入力変数	出力変数	入出力変数	外部変数
(※ *****) STサンプル : 半径と角度から移動後のX座標、Y座標を求める (*****) $X座標 := 半径 * \cos(DEG_TO_RAD(角度));$ $Y座標 := 半径 * \sin(DEG_TO_RAD(角度));$ IF 角度 = 90.0 THEN $X座標 := 0.0;$ ELSIF 角度 = 180.0 THEN $Y座標 := 0.0;$ ELSIF 角度 = 270.0 THEN $X座標 := 0.0;$ ELSIF 角度 = 360.0 THEN $Y座標 := 0.0;$ END_IF;				

STプログラム

FB内部に記述するSTプログラムです。
変数の定義(FB内部)とセットで作成します。

数値の切り上げ、切り捨て、四捨五入 ～四則演算を使う①～

四則演算・算術関数

+(加算)、*(乗算)、/(除算)

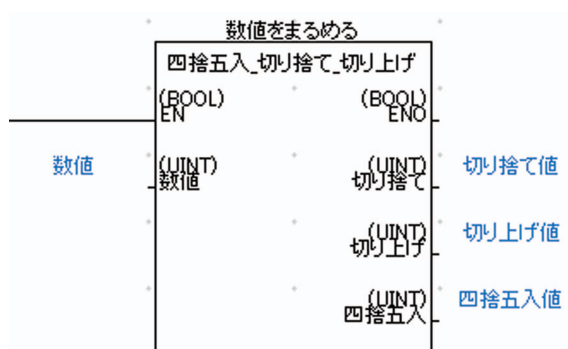
2つの数値の加算・乗算・除算を行います。

■サンプルプログラムの概要

入力された数値を1の位で切り捨て、切り上げ、四捨五入します。
下記の例では、“45”を入力すると、切り捨て値“40”、切り上げ値“50”、四捨五入値“50”を出力します。

FB外部の変数(アドレス)

変数名	アドレス	値
数値	D50	45



FB外部の変数(アドレス)

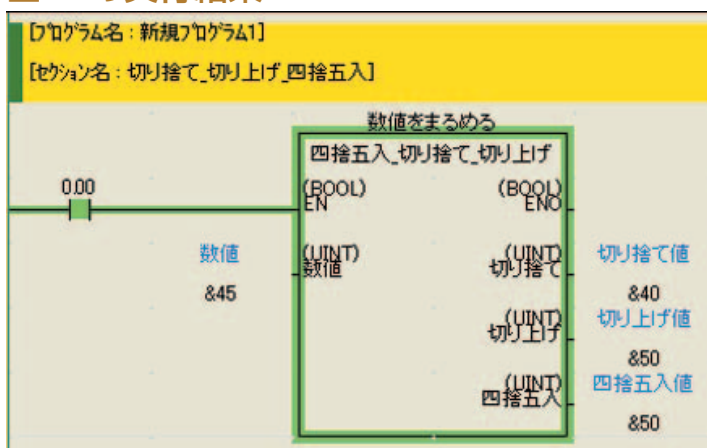
変数名	アドレス	値
切り捨て値	D51	40
切り上げ値	D52	50
四捨五入値	D53	50

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	数値	UINT

変数種別	変数名	データ型
出力変数	切り捨て	UINT
	切り上げ	UINT
	四捨五入	UINT

■FBの実行結果



■STプログラム

```
(* *****
STサンプル : 切り捨て、切り上げ、四捨五入
***** *)
```

```
(* 切り捨て後の値の算出 *)
切り捨て := 数値 / 10 * 10;
```

```
(* 切り上げ後の値の算出 *)
切り上げ := (数値 + 9) / 10 * 10;
```

```
(* 四捨五入後の値の算出 *)
四捨五入 := (数値 + 5) / 10 * 10;
```

移動量からモータへの指令パルス数を求める ～四則演算を使う②～

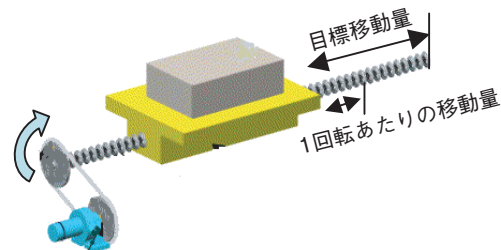
四則演算・算術関数

* (乗算)、/ (除算)

2つの数値の乗算・除算を行います。

■サンプルプログラムの概要

モータ分解能、目標移動量、モータ1回転あたりの移動量から指令パルス数を求めます。
指令パルス数は以下の式によって求めることができます。

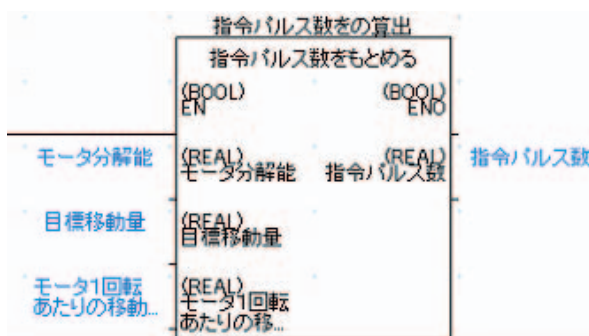


$$\text{指令パルス数(パルス)} = \text{モータ分解能(パルス/rev)} \times \frac{\text{目標移動量(mm)}}{\text{モータ1回転あたりの移動量(mm/rev)}}$$

下記の例では、モータ分解能が1000(パルス/rev)、目標移動量が10mm、モータ1回転あたりの移動量が2.5mmの場合です。

FB外部の変数(アドレス)

変数名	アドレス	値
モータ分解能	D60	1000
目標移動量	D62	10
モータ1回転あたりの移動量	D64	2.5



FB外部の変数(アドレス)

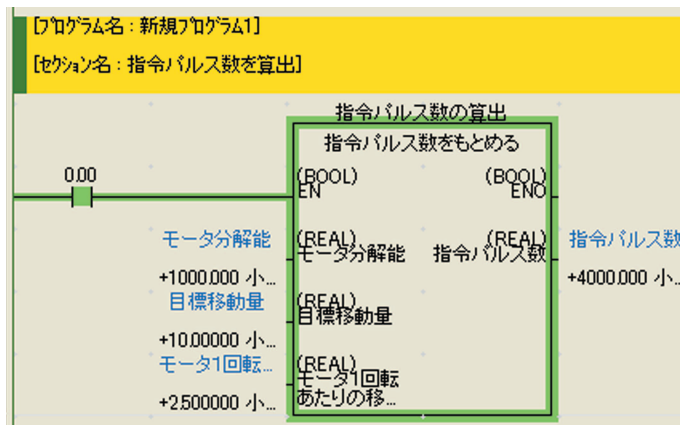
変数名	アドレス	値
指令パルス数	D70	4000

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	モータ分解能	REAL
	目標移動量	REAL
	モータ1回転あたりの移動量	REAL

変数種別	変数名	データ型
出力変数	指令パルス数	REAL

■FBの実行結果



■STプログラム

```
(* *****
STサンプル : 指令パルス数を求める
***** *)
```

指令パルス数 := モータ分解能 * (目標移動量 / モータ1回転あたりの移動量);

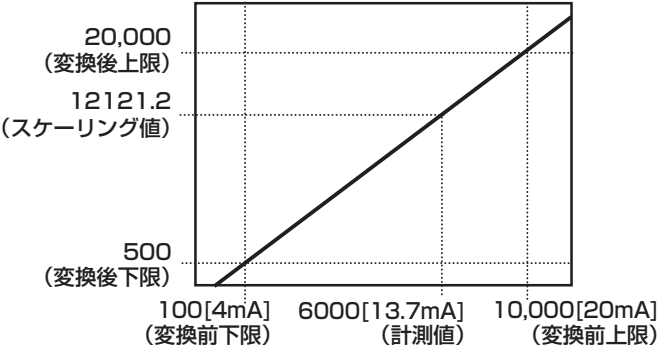
アナログデータをスケーリングする ～四則演算を使う③～

+ (加算)、* (乗算)、/ (除算)

2つの数値の加算・乗算・除算を行います。

■サンプルプログラムの概要

ADユニットや変位センサで取り込んだアナログデータ(測定値)を変換後の上限/下限に基づきスケーリングします。
ここでは、4mA～20mAの電流値が100～10,000にPLC内部で変換され、最終的に500～20,000に変換される場合です。



スケール値 =
$$\frac{(\text{変換後上限} - \text{変換後下限})}{(\text{変換前上限} - \text{変換前下限})} \times (\text{計測値} - \text{変換前下限}) + \text{変換後下限}$$

下記の例では、PLC内部で変換された値：6000(約13.7mAに相当)の場合のスケーリングの例です。

FB外部の変数(アドレス)

変数名	アドレス	値
変換前上限	D600	10,000
変換前下限	D602	100
アナログ値	D604	6000
変換後上限	D606	20,000
変換後下限	D608	500



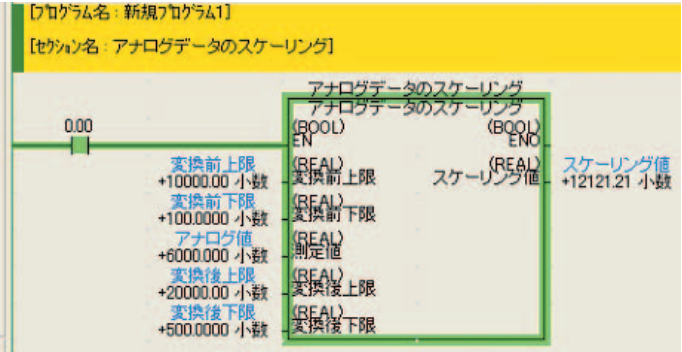
FB外部の変数(アドレス)

変数名	アドレス	値
スケール値	D610	12121.2

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	変換前上限	REAL
	変換前下限	REAL
	測定値	REAL
	変換後上限	REAL
	変換後下限	REAL
出力変数	スケール値	REAL

■FBの実行結果



■STプログラム

```
(* *****  
STサンプル : アナログデータをスケールングする  
***** *)  
  
スケール値 := ( 変換後上限 - 変換後下限 ) / (変換前上限-変換前下限) * (測定値 - 変換前下限) + 変換後下限;
```

トータル時間(秒)から時・分・秒を求める ～ MOD関数を使う～

MOD関数（数値演算・算術関数：除算の余り）

数値を除数で割ったときの余りを返します。

■サンプルプログラムの概要

入力された積算時間(秒)を時間、分、秒に変換します。
下記の例では、積算秒に100,000秒を入力すると、27時間46分40秒に変換されます。
ここでは、入力のデータ範囲を倍長符号なし10進数(UDINT)で指定していますので、0～4,294,967,295の範囲で扱えます。

FB外部の変数(アドレス)

変数名	アドレス	値
積算秒	D120	100000



FB外部の変数(アドレス)

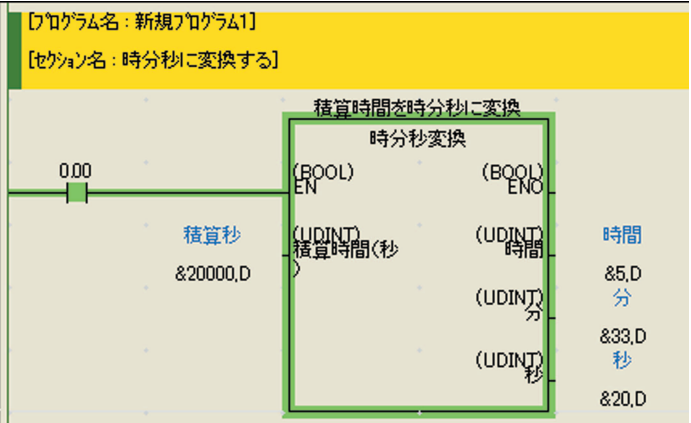
変数名	アドレス	値
時間	D130	27
分	D132	46
秒	D134	40

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	積算時間_秒	UDINT

変数種別	変数名	データ型
出力変数	時間	UDINT
	分	UDINT
	秒	UDINT

■FBの実行結果



■STプログラム

```
(* *****  
STサンプル  : 積算時間(秒)を時間・分・秒に変換する  
***** *)
```

```
時間 := 積算時間_秒 / 3600;  
分 := (積算時間_秒 MOD 3600) / 60;  
秒 := (積算時間_秒 MOD 3600) MOD 60;
```

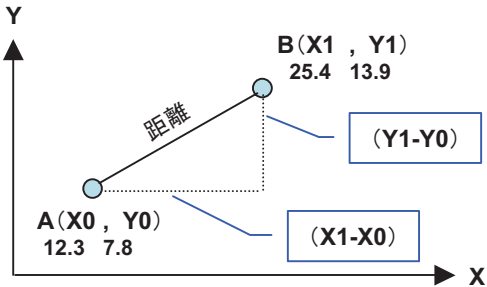
二点間の距離を求める ～平方根や指数関数を使う～

SQRT (数値演算・算術関数：平方根)	数値の平方根を求めます。
** (数値演算・算術関数：べき乗)	数値を指定した指数に基づき、べき乗します。

■サンプルプログラムの概要

2点間A-B((X0,Y0)と(X1,Y1))の距離を求めます。
2点間の距離を求める公式は三平方の定理より以下ようになります。

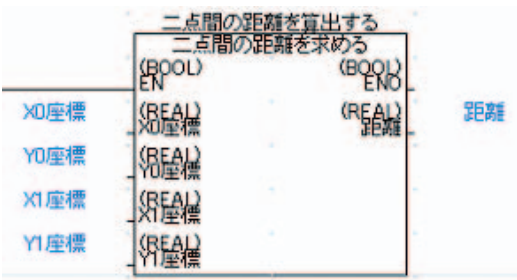
距離 = $\sqrt{(X1-X0)^2 + (Y1-Y0)^2}$



下記の例では、A(12.3, 7.8)とB(25.4, 13.9)の間の距離を算出しています。

FB外部の変数(アドレス)

変数名	アドレス	値
X0座標	D140	12.3
Y0座標	D142	7.8
X1座標	D144	25.4
Y1座標	D146	13.9



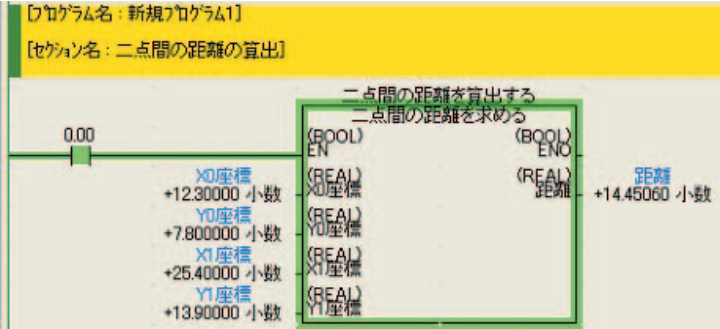
FB外部の変数(アドレス)

変数名	アドレス	値
距離	D148	14.45

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	X0座標	REAL
	Y0座標	REAL
	X1座標	REAL
	Y1座標	REAL
出力変数	距離	REAL

■FBの実行結果



■STプログラム

(

STサンプル : 二点間の距離を求める

)

距離 := SQRT ((X1座標 - X0座標)**2 + (Y1座標 - Y0座標)**2);

} SQRTは平方根を算出します。
"***"は数値を指定した指数に基づきべき乗します。

半径と角度からX、Y座標を求める ～三角関数・ラジアン変換を使う～

四則演算・算術関数

SIN / COS (数値演算・算術関数：
三角関数)

ラジアンを引数として角度の正弦値(SIN)、余弦値(COS)を
もとめます。

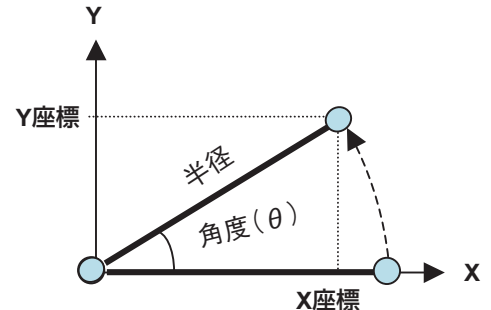
DEG_TO_RAD (型変換関数)

度で指定した角度をラジアンに変換します。

■サンプルプログラムの概要

ロボットアームなどの回転物において、回転物の長さ(半径)と
回転する角度を与えたときの移動後のX座標、Y座標を求めます。
公式は以下の通りです。

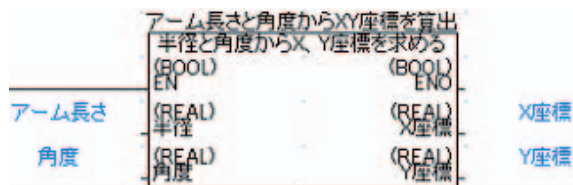
$$\begin{aligned} \text{X座標} &= \text{半径} \times \text{COS}(\theta) \\ \text{Y座標} &= \text{半径} \times \text{SIN}(\theta) \end{aligned}$$



下記の例では、アーム長さ(半径)が1000mm、回転角度が30度の場合です。

FB外部の変数(アドレス)

変数名	アドレス	値
アーム長さ	D400	1000
角度	D402	30



FB外部の変数(アドレス)

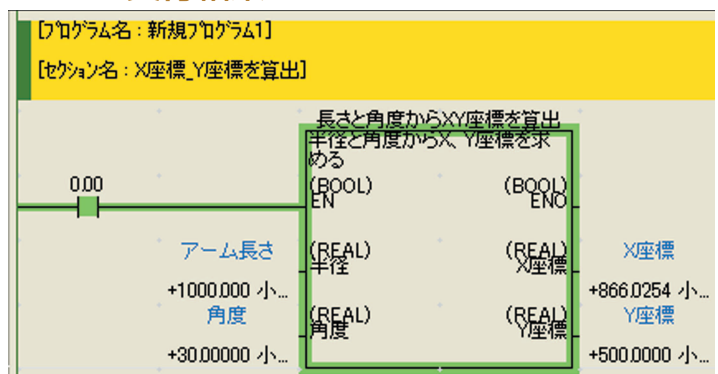
変数名	アドレス	値
X座標	D410	866.02
Y座標	D412	500.00

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	半径	REAL
	角度	REAL

変数種別	変数名	データ型
出力変数	X座標	REAL
	Y座標	REAL

■FBの実行結果



■STプログラム

(* *****
STサンプル : 半径と角度から移動後のX座標、Y座標を求める
***** *)

X座標 := 半径 * COS (DEG_TO_RAD (角度));
Y座標 := 半径 * SIN (DEG_TO_RAD (角度));

DEG_TO_RAD関数で単位変換します。

IF 角度 = 90.0 THEN
X座標 := 0.0;
ELSIF 角度 = 180.0 THEN
Y座標 := 0.0;
ELSIF 角度 = 270.0 THEN
X座標 := 0.0;
ELSIF 角度 = 360.0 THEN
Y座標 := 0.0;
END_IF;

演算上の誤差に対する例外処理をします。
(1回転以内の場合)

配列データの平均値を求める ～繰り返し処理(FOR文)を使う～

ステートメント(ST構文)

+(加算)、/(除算)

2つの数値の加算・除算を行います。

FOR文 (繰り返し処理)

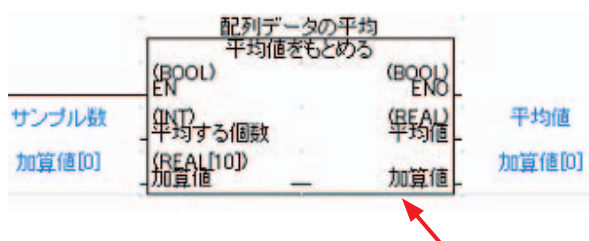
指定した条件が成立するまで繰り返し処理を行います。

■サンプルプログラムの概要

配列に登録された浮動小数点データを指定したサンプル数で平均を算出します。
下記例では、サンプル数を5と指定し、配列変数“加算値[0]～加算値[4]”(アドレス：D20～D29(浮動小数点データのため2CH/データ))の5つのデータをFBに入力すると、その平均値を出力します。
ここでは、サンプル数の最大を10個まで使用できる例です。

FB外部の変数(アドレス)

変数名	アドレス	値
サンプル数	D10	5
加算値 [0]	D20	20.9
加算値 [1]	D22	21.3
加算値 [2]	D24	21.8
加算値 [3]	D26	20.5
加算値 [4]	D28	20.7



FB外部の変数(アドレス)

変数名	アドレス	値
平均値	D12	21.0
加算値 [0]	D20	20.9
加算値 [1]	D22	21.3
加算値 [2]	D24	21.8
加算値 [3]	D26	20.5
加算値 [4]	D28	20.7

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	平均する個数	UINT

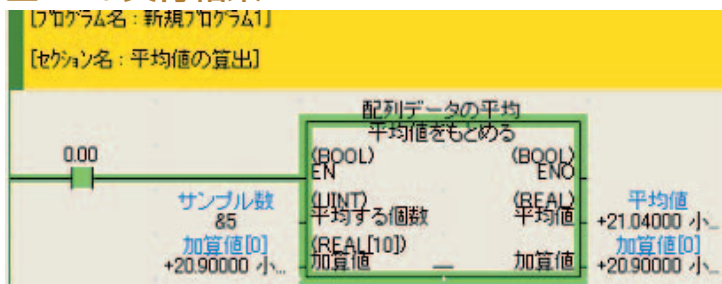
変数種別	変数名	データ型
出力変数	平均値	REAL

変数種別	変数名	データ型
内部変数	合計値	REAL
	i	INT

変数種別	変数名	データ型
入出力変数	加算値	REAL[10]

※ 配列データの設定の仕方は、「付録1」を参照してください。

■FBの実行結果



■STプログラム

(* **** STサンプル : 配列データの平均を算出する **** *)

合計値 := 0.0;

FOR i := 0 TO 平均する個数 - 1 DO

合計値 := 合計値 + 加算値[i];

END_FOR;

平均値 := 合計値 / INT_TO_REAL(平均する個数);

FB内部変数を初期化します。

指定した回数ループします。

除算する2つのデータ型をそろえるために
UINT_TO_REALで型変換します。

■動作の確認方法

※本プログラムは複数スキャンに渡って処理が完了しますのでスキャンタイムが延びる可能性があります。

CX-Programmerのウォッチウィンドウで下記のように値をセットし、FB起動フラグ(0.00)をONします。

PLC名	名称	アドレス	データ型/表示形式	FB用途	値	値(2進数)
新規PLC1	新規プログラム1 加算値[0]	D20	REAL (単精度浮動小数点倍長)		+20.90000 小数	+20.90000 小数
新規PLC1	新規プログラム1 加算値[1]	D22	REAL (単精度浮動小数点倍長)		+21.30000 小数	+21.30000 小数
新規PLC1	新規プログラム1 加算値[2]	D24	REAL (単精度浮動小数点倍長)		+21.80000 小数	+21.80000 小数
新規PLC1	新規プログラム1 加算値[3]	D26	REAL (単精度浮動小数点倍長)		+20.50000 小数	+20.50000 小数
新規PLC1	新規プログラム1 加算値[4]	D28	REAL (単精度浮動小数点倍長)		+20.70000 小数	+20.70000 小数
新規PLC1	新規プログラム1 サンプル数	D10	UINT (10進数・16ビット)		85	0000 0000 0000 0101
新規PLC1	新規プログラム1 平均値	D12	REAL (単精度浮動小数点倍長)		+0.0000000 小数	+0.0000000 小数



配列データのウォッチウィンドウへの登録の仕方は付録3を参照してください。

配列データを並べ替える

ステートメント(ST構文)

～繰り返し処理(FOR文)+条件分岐(IF文)の組み合わせ～

FOR文 (繰り返し処理)

指定した条件が成立するまで繰り返し処理を行います。

IF文 (条件分岐処理)

条件式が成立すると成立した処理を行います。

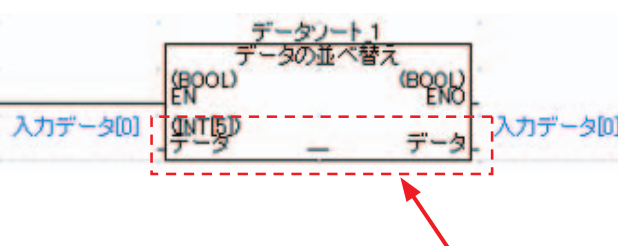
■サンプルプログラムの概要

入力された符号付き10進データを昇順で並べ替えます。

下記例では、配列変数“データ[0]～データ[4]”(アドレス:D100～D104)の5つのデータをFBに入力すると、昇順に並べ替えて同じ配列変数に出力します。

FB外部の変数(アドレス)

変数名	アドレス	値
入力データ [0]	D100	45
入力データ [1]	D101	60
入力データ [2]	D102	20
入力データ [3]	D103	-10
入力データ [4]	D104	5



FB外部の変数(アドレス)

変数名	アドレス	値
入力データ [0]	D100	-10
入力データ [1]	D101	5
入力データ [2]	D102	20
入力データ [3]	D103	45
入力データ [4]	D104	60

※ 配列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

※ FBへの配列データの設定は、配列名を指定する、あるいは、配列の先頭アドレス“D100”を入力します。

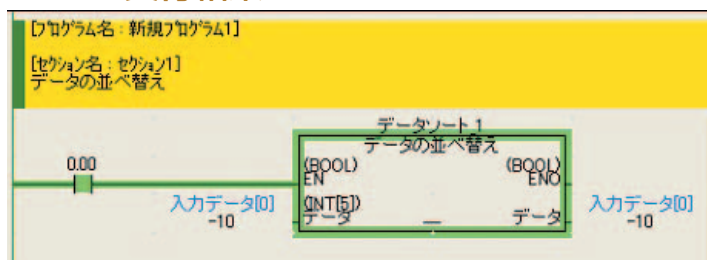
■変数の定義(FB内部)

変数種別	変数名	データ型
入出力変数	データ	INT[5]

※ 配列データの設定の仕方は、「付録1」を参照してください。

変数種別	変数名	データ型
内部変数	i	UINT
	work	INT

■FBの実行結果



■STプログラム

```
(
*****
STサンプル : 入力データをソートする
*****
)
```

```
FOR i := 0 TO 3 DO
  IF データ[i] > データ[i+1] THEN
    work := データ[i];
    データ[i] := データ[i+1];
    データ[i+1] := work;
  END_IF;
END_FOR;
```

1 つ前の配列
データより小さい
とき、入れ替
えます。

4 回ループします。

■動作の確認方法

※本プログラムは複数スキャンに渡って処理が完了しますのでスキャンタイムが延びる可能性があります。

CX-Programmerのウォッチウィンドウで下記のように値をセットし、FB起動フラグ(0.00)をONします。

PLC名	名称	アドレス	データ型/表示形式	FB用途	値	値(2進数)
新規PLC1	新規プログラム1 入力データ[0]	D100	INT (符号付10進数)		+45	0000 0000 0010 1101
新規PLC1	新規プログラム1 入力データ[1]	D101	INT (符号付10進数)		+60	0000 0000 0011 1100
新規PLC1	新規プログラム1 入力データ[2]	D102	INT (符号付10進数)		+20	0000 0000 0001 0100
新規PLC1	新規プログラム1 入力データ[3]	D103	INT (符号付10進数)		-10	1111 1111 1111 0110
新規PLC1	新規プログラム1 入力データ[4]	D104	INT (符号付10進数)		+5	0000 0000 0000 0101



配列データのウォッチウィンドウへの登録の仕方は付録3を参照してください。

PLC名	名称	アドレス	データ型/表示形式	FB用途	値	値(2進数)
新規PLC1	新規プログラム1 入力データ[0]	D100	INT (符号付10進数)		-10	1111 1111 1111 0110
新規PLC1	新規プログラム1 入力データ[1]	D101	INT (符号付10進数)		+5	0000 0000 0000 0101
新規PLC1	新規プログラム1 入力データ[2]	D102	INT (符号付10進数)		+20	0000 0000 0001 0100
新規PLC1	新規プログラム1 入力データ[3]	D103	INT (符号付10進数)		+45	0000 0000 0010 1101
新規PLC1	新規プログラム1 入力データ[4]	D104	INT (符号付10進数)		+60	0000 0000 0011 1100

左記のようにデータが並べ替えられます。

選択分岐を行う(電卓機能) ～選択分岐(CASE文)を使う～

ステートメント(ST構文)

CASE文

複数の条件により、条件式にしたがって各処理を分岐します。

+(加算)、-(減算)、
*(乗算)、/(除算)

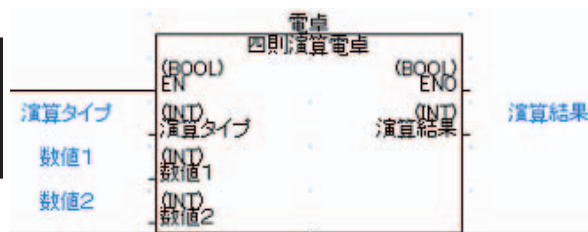
2つの数値の加算・減算・乗算・除算を行います。

■サンプルプログラムの概要

指定した演算タイプ(加・減・乗・除)の条件式に従って、四則演算を行い出力します。
下記の例では、2つの整数と演算タイプ(1:加算、2:減算、3:乗算、4:除算)を入力すると、演算タイプに従って計算します。
ここでは、2つの整数を、「10」と「2」として演算タイプを変更させた場合です。

FB外部の変数(アドレス)

変数名	アドレス	値
演算タイプ	D80	1～4
数値1	D81	10
数値2	D82	2



FB外部の変数(アドレス)

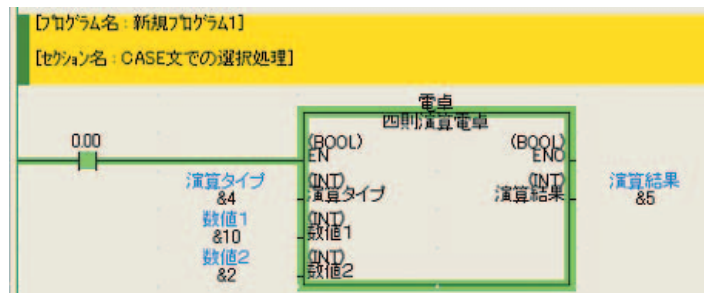
変数名	アドレス	値
演算結果	D83	(※1)

(※1)演算タイプによる演算結果
1: 10 + 2 = 12
2: 10 - 2 = 8
3: 10 * 2 = 20
4: 10 / 2 = 5
その他: 0

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	演算タイプ	INT
	数値1	INT
	数値2	INT
出力変数	演算結果	INT

■FBの実行結果



■STプログラム

```
( *
STサンプル : CASE文による選択処理
***** )
```

CASE 演算タイプ OF

```
1: 演算結果 := 数値1 + 数値2; (*足し算*)
2: 演算結果 := 数値1 - 数値2; (*引き算*)
3: 演算結果 := 数値1 * 数値2; (*掛け算*)
4: 演算結果 := 数値1 / 数値2; (*割り算*)
```

ELSE

```
演算結果 := 0; (*上記以外のケースの場合*)
```

END_CASE;

文字列の長さ(文字数)を検出する ～文字列操作関数(LEN関数)を使う～

文字列関数

LEN関数(文字列操作関数)

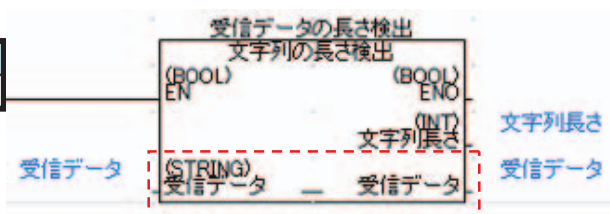
指定した文字列の長さ(文字数)を検出します。

■サンプルプログラムの概要

バーコードリーダーやRFIDなどから受信した文字列データに対して、その長さ(文字数)を検出します。

FB外部の変数(アドレス)

変数名	アドレス	値
受信データ	D210	(*1)



FB外部の変数(アドレス)

変数名	アドレス	値
文字列長さ	D202	+10
受信データ	D210	(*1)

文字列長さ: "+10"

(*1) 受信データは下記とします。

A	B	C	-	D	E	-	1	2	3
---	---	---	---	---	---	---	---	---	---

※ 文字列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

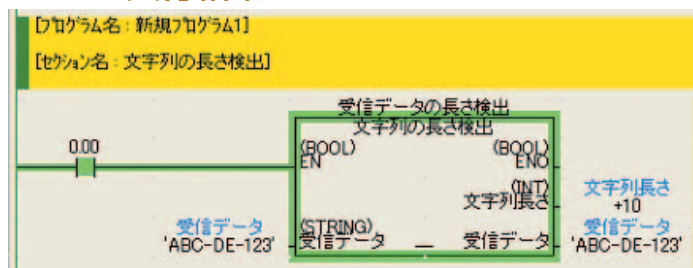
■変数の定義(FB内部)

変数種別	変数名	データ型
出力変数	文字列長さ	INT

変数種別	変数名	データ型
入出力変数	受信データ	STRING(20)

※ 文字列データの設定の仕方は、「付録1」を参照してください。

■FBの実行結果



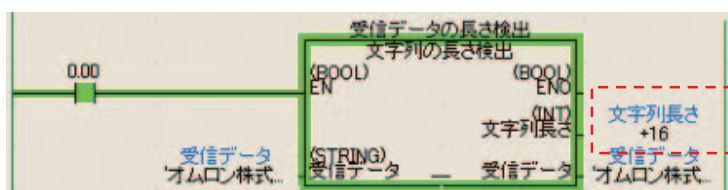
■STプログラム

```
(* *****
STサンプル : 文字列の長さ検出する(LEN関数)
***** *)
```

文字列長さ:= LEN(受信データ);

■動作の確認方法

ラダーモニタ画面で、受信データを変更し、指定した文字列の長さ(文字数)が検出されたことを確認します。



受信データを”オムロン株式会社“に変更すると、左記のように文字数が検出されます。
・ “オムロン株式会社”の文字数:16

※ 漢字のような2バイト文字を抽出するときは、1文字あたり2文字として検出されます。

文字列を抽出する ～文字列操作関数(LEFT、RIGHT関数)を使う～

LEFT関数 (文字列操作関数)

指定した文字列の左端から指定した文字数を抽出します。

RIGHT関数 (文字列操作関数)

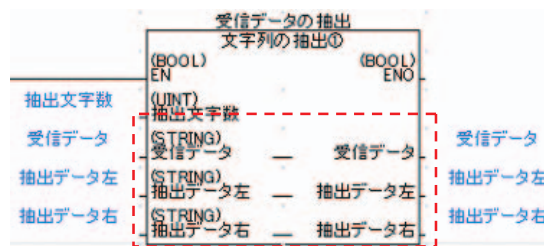
指定した文字列の右端から指定した文字数を抽出します。

■サンプルプログラムの概要

バーコードリーダーやRFIDなどから受信した文字列データに対して、指定した位置から指定した文字数を抽出します。

FB外部の変数(アドレス)

変数名	アドレス	値
抽出文字数	D200	5
受信データ	D210	(*1)
抽出データ左	D230	(*2)
抽出データ右	D250	(*3)



FB外部の変数(アドレス)

変数名	アドレス	値
受信データ	D210	(*1)
抽出データ左	D230	(*2)
抽出データ右	D250	(*3)

(*2) 左から5文字抽出: "OMRON"
(*3) 右から5文字抽出: "-1234"

(*1) 受信データは下記とします。

O	M	R	O	N	-	1	2	3	4
---	---	---	---	---	---	---	---	---	---

(*2) 右記の実行結果となります。

※ 文字列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

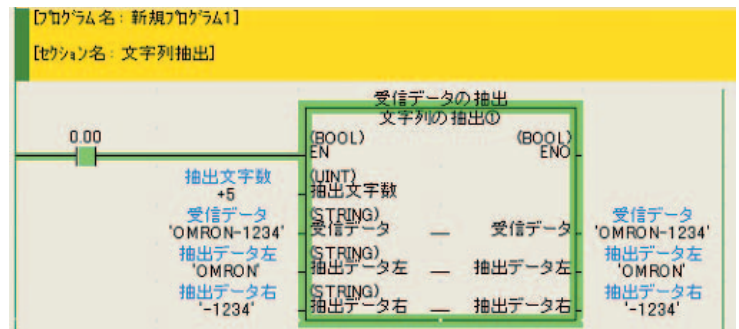
■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	抽出文字数	UINT

変数種別	変数名	データ型
入出力変数	受信データ	STRING (20)
	抽出データ左	STRING (20)
	抽出データ右	STRING (20)

※ 文字列データの設定の仕方は、「付録1」を参照してください。

■FBの実行結果



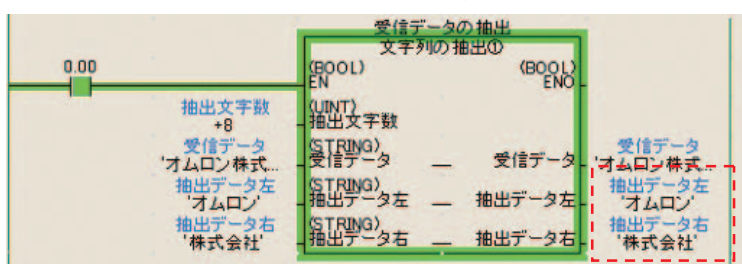
■STプログラム

(* *****
STサンプル : 文字列を抽出する(LEFT関数、RIGHT関数)
***** *)

抽出データ左 := LEFT (受信データ, 抽出文字数);
抽出データ右 := RIGHT (受信データ, 抽出文字数);

■動作の確認方法

ラダーモニタ画面で、抽出文字数、受信データを変更し、指定した文字列が抽出されたことを確認します。



受信データを"オムロン株式会社"に変更すると、左記のように文字列が抽出されます。
・ 左から8文字抽出 → "オムロン"
・ 右から8文字抽出 → "株式会社"

※ 漢字のような2バイト文字を抽出するときは、1文字あたり2文字として指定します。

文字列を抽出する ～文字列操作関数(MID関数)を使う～

文字列関数

MID関数（文字列操作関数）

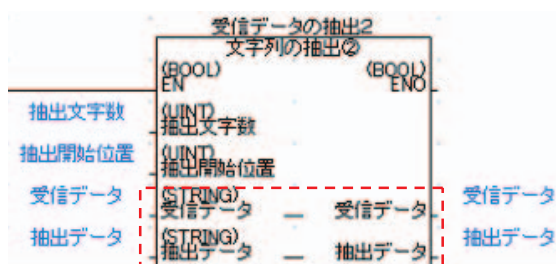
指定した文字列の指定した開始位置から指定した文字数を抽出します。

■サンプルプログラムの概要

バーコードリーダーやRFIDなどから受信した文字列データに対して、指定した位置から指定した文字数を抽出します。

FB外部の変数(アドレス)

変数名	アドレス	値
抽出文字数	D200	2
抽出開始位置	D201	5
受信データ	D210	(*1)
抽出データ	D270	(*2)



FB外部の変数(アドレス)

変数名	アドレス	値
受信データ	D210	(*1)
抽出データ	D270	(*2)

(*2)左端5文字から2文字抽出: "DE"

(*1) 受信データは下記とします。

A	B	C	-	D	E	-	1	2	3
---	---	---	---	---	---	---	---	---	---

(*2) 右記の実行結果となります。

※ 文字列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

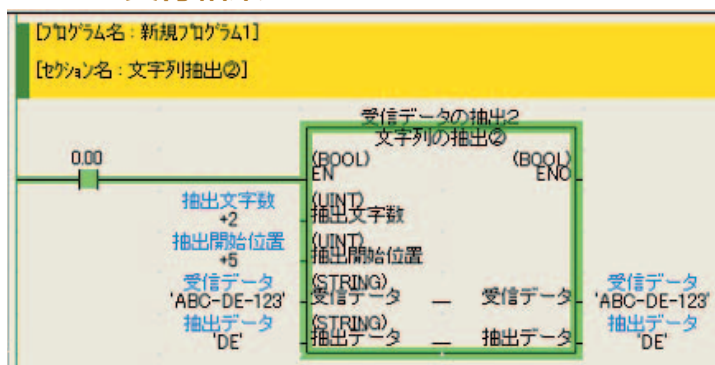
■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	抽出文字数	UINT
	抽出開始位置	UINT

変数種別	変数名	データ型
入出力変数	受信データ	STRING(20)
	抽出データ	STRING(20)

※ 文字列データの設定の仕方は、「付録1」を参照してください。

■FBの実行結果



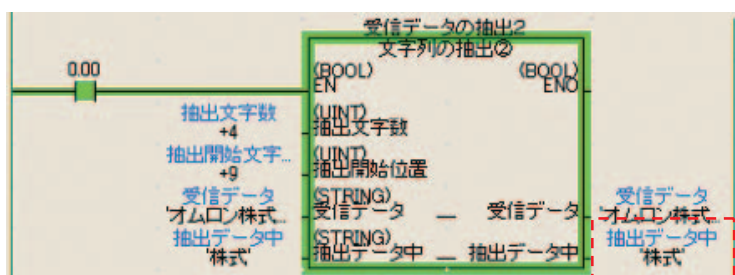
■STプログラム

```
(* *****
STサンプル  : 文字列を抽出する(MID関数)
***** *)
```

抽出データ:= MID (受信データ, 抽出文字数, 抽出開始位置);

■動作の確認方法

ラダーモニター画面で、抽出文字数、受信データを変更し、指定した文字列が抽出されたことを確認します。



受信データを"オムロン株式会社"に変更すると、左記のように文字列が抽出されます。

- 抽出文字数: 4
 - 抽出開始位置: 9
- "株式"

※ 漢字のような2バイト文字を抽出するときは、1文字あたり2文字として指定します。

文字列関数

文字列を結合します。

下記の例では、2つの入力文字列を入力し、結合した文字列を出力します。

※ 文字列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

※ 文字列データの設定の仕方は、「付録1」を参照してください。

【プログラム名: 新規プログラム1】

【セクション名: 文字列結合①】

文字列の結合
文字列を結合する①

0.00

(BOOL) EN (BOOL) ENO

入力文字列1 'CX' 入力文字列1 'CX'

入力文字列2 'Programmer' 入力文字列2 'Programmer'

結合した文字列 'CXProgrammer' 結合した文字列 'CXProgrammer'

結合した文字列 := CONCAT(入力文字列 1, 入力文字列 2);

文字列を結合する②

文字列関数

～文字列操作関数(CONCAT関数、LEN関数)の組み合わせ～

CONCAT関数 (文字列操作関数)

文字列を結合します。

LEN関数 (文字列操作関数)

指定した文字列の長さ (文字数) を検出します。

INT_TO_STRING (型変換関数)

INT型のデータをSTRING型に変換します。

■サンプルプログラムの概要

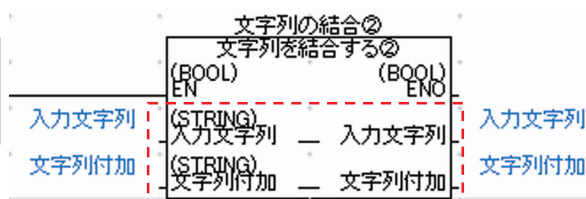
文字列データや文字列定数を結合します。

下記の例では、入力文字列の文字数をカウントし、その文字列に文字列定数と文字数を付加します。

ここでは、入力文字列 'abcde' という文字列に対して ' ->' という文字列定数と入力文字列の文字数を結合して出力する例です。

FB外部の変数(アドレス)

変数名	アドレス	値
入力文字列	D500	(*1)
文字列付加	D520	(*2)



FB外部の変数(アドレス)

変数名	アドレス	値
入力文字列	D500	(*1)
文字列付加	D520	(*2)

(*1) 入力文字列

a	b	c	d	e					
---	---	---	---	---	--	--	--	--	--

(*2) 文字列定数と文字数を結合した文字列

a	b	c	d	e	-	>	5		
---	---	---	---	---	---	---	---	--	--

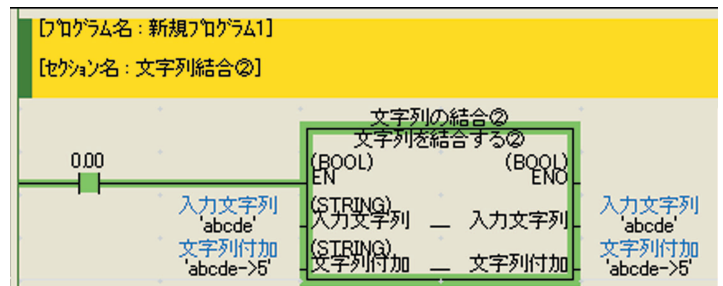
※ 文字列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

■変数の定義(FB内部)

変数種別	変数名	データ型
入出力変数	入力文字列	STRING(10)
	文字列付加	STRING(20)

※ 文字列データの設定の仕方は、「付録1」を参照してください。

■FBの実行結果



■STプログラム

```
(
*****
STサンプル : 文字列を結合する②(CONCAT関数)
*****
)
```

文字数 := LEN(入力文字列);

文字数付加 :=CONCAT(入力文字列, '-',INT_TO_STRING(文字数));

文字数をカウントします。

文字列定数は' ' で囲みます。
また、数値データから文字列データに変換してから結合します。

文字列を置換する ～文字列操作関数(REPLACE関数)を使う～

文字列関数

REPLACE関数 (文字列操作関数)

文字列の中で指定した文字列1を文字列2に置き換えます。

IF文 (条件分岐処理)

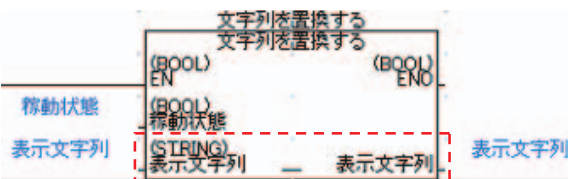
条件式が成立すると成立した処理を行います。

■サンプルプログラムの概要

文字列データを指定した位置から指定した文字数分、指定した文字列で置換します。
下記の例では、稼動状態フラグの状態によって、表示文字列の中の“正常”という文字列を“異常”という文字列に置き換えます。
ここでは、表示文字列に対して、先頭から8番目より4文字分、置換文字列としてセットされた“異常”という文字で置き換える場合です。(文字は半角文字数で換算します。)

FB外部の変数(アドレス)

変数名	アドレス	値
稼動状態	W50.00	0/1
表示文字列	D550	(*1)



FB外部の変数(アドレス)

変数名	アドレス	値
稼動状態	W50.00	0/1
表示文字列	D550	(*1)

(*1) 表示文字列 (稼動状態: OFF時)

A	工	程	は	正	常	で	す		
---	---	---	---	---	---	---	---	--	--

(*1) 稼動状態がONのとき、“文字列” “正常”が“異常”に置換されます。

A	工	程	は	異	常	で	す		
---	---	---	---	---	---	---	---	--	--

※ 文字列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

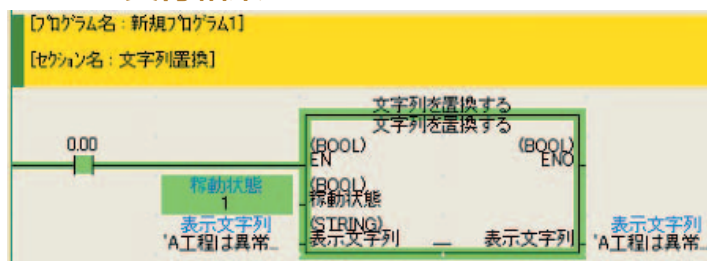
■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	稼動状態	BOOL

変数種別	変数名	データ型
入出力変数	表示文字列	STRING(20)

※ 文字列データの設定の仕方は、「付録1」を参照してください。

■FBの実行結果



■STプログラム

```
(* *****
STサンプル : 文字列を置換する
***** *)
```

```
表示文字列 := 'A工程は正常です';
置換文字列1 := '異常'; 置換文字列2 := '正常';
```

```
IF 稼動状態 = TRUE THEN
    表示文字列 := REPLACE(表示文字列, 置換文字列1, 4, 8);
ELSE
    表示文字列 := REPLACE(表示文字列, 置換文字列2, 4, 8);
END_IF;
```

デフォルトで表示する文字列、置き換える文字列をセットします。

稼動状態に従って、先頭から8番目より4文字分、置換え文字列で置換します。

■動作の確認方法

CX-Programmerのウォッチウィンドウで下記のように値をセットし、FB起動フラグ(0.00)をONします。

PLC名	名称	アドレス	データ型/表示形式	FB用途	値
新規PLC1	新規プログラム1.稼動状態	W50.00	BOOL (On/Off,接点)		0
新規PLC1	新規プログラム1.表示文字列	D550	STRING		'A工程は異常です'

PLC名	名称	アドレス	データ型/表示形式	FB用途	値
新規PLC1	新規プログラム1.稼動状態	W50.00	BOOL (On/Off,接点)		1
新規PLC1	新規プログラム1.表示文字列	D550	STRING		'A工程は異常です'

年月日情報をPLCから収集する ～ PLCの特殊補助リレーを使う～

PLCテクニック

LEFT関数／RIGHT関数（文字列操作関数）

指定した文字列の左端から指定した文字数を抽出します。
指定した文字列の右端から指定した文字数を抽出します。

CONCAT関数（文字列操作関数）

文字列を結合します。

WORD_TO_STRING（型変換関数）

WORD型のデータをSTRING型に変換します。

■サンプルプログラムの概要

SYSMAC CS/CJ/CPシリーズの年月日情報を取り出し、ファンクションブロック(FB)内で文字列として扱います。

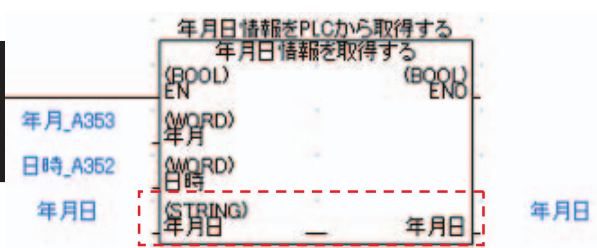
SYSMAC CS/CJ/CPシリーズでは、特殊補助リレー A351 ～ A353に年月日や時分秒情報が格納されています。

FB外部の変数(アドレス)

変数名	アドレス	値
----	A353	(※1)
----	A352	(※1)
年月日	D800	(※2)

(※1) 特殊補助リレーの割付は以下の通りです。

	15-8	7-0
A353	年	月
A352	日	時
A351	分	秒



FB外部の変数(アドレス)

変数名	アドレス	値
年月日	D800	(※2)

(※2) '年'、'月'、'日' 情報を文字列として加工して下記のように出力します。

年月日
'2010年08月07日'

※ 文字列データのような一連のデータをFBに入力するときは、入出力変数を使用します。

■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	年月	WORD
	日時	WORD

変数種別	変数名	データ型
入出力変数	年月日	STRING[20]

※ 文字列データの設定の仕方は、「付録1」を参照してください。

■FBの実行結果



A353: "10年08月"の意味
A352: "07日18時"の意味

※ "16進モニタ" (HEX) でモニタすると年月日の数値で確認できます。

■STプログラム

```
(* *****
STサンプル : 年月日情報をPLCから取得する
***** *)
```

```
年 := LEFT(WORD_TO_STRING(年月), 2);
月 := RIGHT(WORD_TO_STRING(年月), 2);
日 := LEFT(WORD_TO_STRING(日時), 2);
```

```
年月日 := CONCAT ('20', 年, '年', 月, '月', 日, '日');
```

ビットシフトを行う ～ビット操作関数(SHL関数、SHR関数)を使う～

SHL関数(ビット列操作関数)
SHR関数(ビット列操作関数)

ビット列を指定したシフト数分、左方向にシフトします。
ビット列を指定したシフト数分、右方向にシフトします。

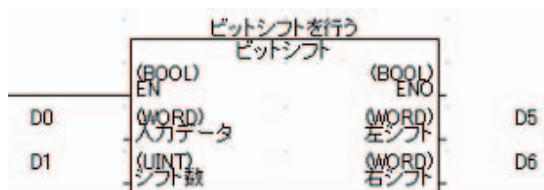
■サンプルプログラムの概要

ビット列データを指定したシフト数分、左方向および右方向にシフトします。

ここでは、データメモリ“D0”に格納された16ビットの文字列“0001 0010 0011 0100 (#1234 (HEX))”に対して、データメモリ“D1”にシフト数(4ビット)を指定し、それぞれを左右にシフトして、データメモリ“D5”と“D6”に出力する場合があります。(下記の例では、FB外部変数に変数名を使用しない例です)

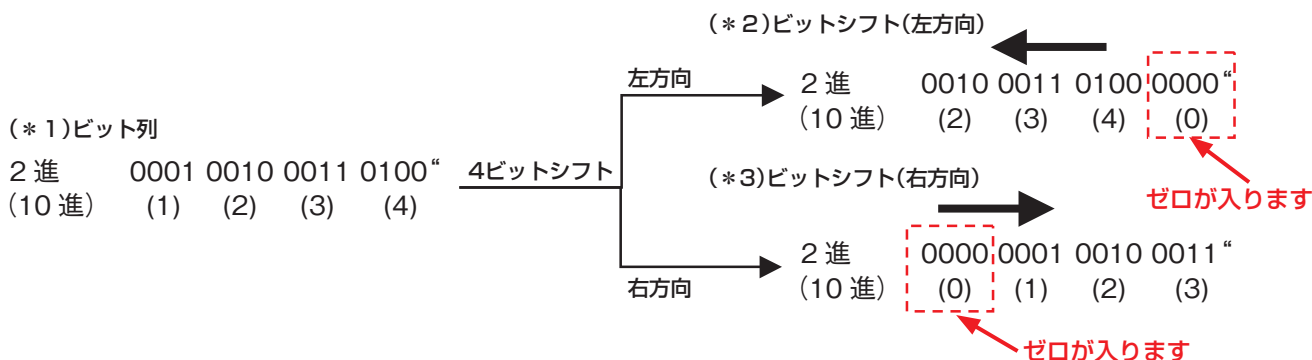
FB外部の変数(アドレス)

変数名	アドレス	値
---	D0	(※1)
---	D1	4



FB外部の変数(アドレス)

変数名	アドレス	値
---	D5	(※2)
---	D6	(※3)



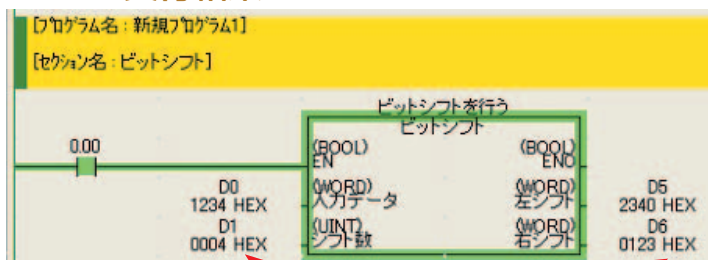
■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	入力データ	WORD
	シフト数	UINT

変数種別	変数名	データ型
出力変数	左シフト	WORD
	右シフト	WORD

※ PLC内部のビット列データを扱うときは、WORD型を指定します。

■FBの実行結果



※ モニタ形式を“16進モニタ”(HEX)に切り替えてモニタします。

■STプログラム

```
(* *****
STサンプル : ビットシフト
***** *)
```

```
左シフト := SHL(入力データ, シフト数);
右シフト := SHR(入力データ, シフト数);
```

```
(* 左方向にシフトします *)
(* 右方向にシフトします *)
```

ビット回転を行う ～ビット操作関数(ROL関数、ROR関数)を使う～

ROL関数(ビット列操作関数)
ROR関数(ビット列操作関数)

ビット列を指定したシフト数分、左方向に回転します。
ビット列を指定したシフト数分、右方向に回転します。

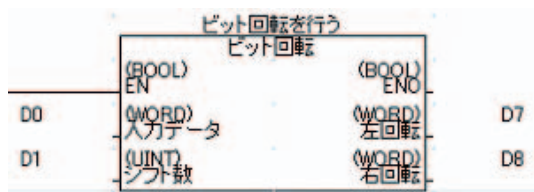
■サンプルプログラムの概要

ビット列データを指定したシフト数分、左方向および右方向に回転します。

ここでは、データメモリ“D0”に格納された16ビットの文字列“0001 0010 0011 0100 (#1234 (HEX))”に対して、データメモリ“D1”にシフト数(4ビット)を指定し、それぞれを左右に回転し、データメモリ“D7”と“D8”に出力する場合です。(下記の例では、FB外部変数に変数名を使用しない例です)

FB外部の変数(アドレス)

変数名	アドレス	値
---	D0	(※1)
---	D1	4



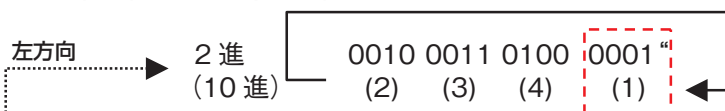
FB外部の変数(アドレス)

変数名	アドレス	値
---	D7	(※2)
---	D8	(※3)

(※1)ビット文字列

2進 (10進) 0001 0010 0011 0100 “ 4ビット回転
(1) (2) (3) (4)

(※2)ビット回転(左方向)



(※3)ビット回転(右方向)

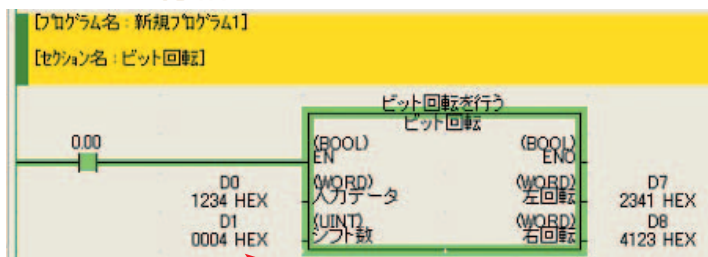


■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	入力データ	WORD
	シフト数	UINT

変数種別	変数名	データ型
出力変数	左回転	WORD
	右回転	WORD

■FBの実行結果



※ PLC内部のビット列データを扱うときは、WORD型を指定します。

※ モニタ形式を“16進モニタ”(HEX)に切り替えてモニタします。

■STプログラム

(* **** STサンプル : ビット回転 **** *)

STサンプル : ビット回転

(* **** STサンプル : ビット回転 **** *)

左回転 := ROL(入力データ, シフト数);
右回転 := ROR(入力データ, シフト数);

(* 左方向に回転します *)

(* 右方向に回転します *)

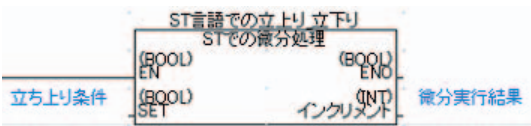
ST言語での微分回路 ～立ち上がり微分の作り方～

■サンプルプログラムの概要

このサンプルプログラムは常時FBが実行している前提で、入力信号の立ち上りで微分処理を行う記述の一例です。
ここでは、立ち上り条件がONし続けても、立ち上り時の微分処理(1回のみ実行：インクリメント)を行なう例です。

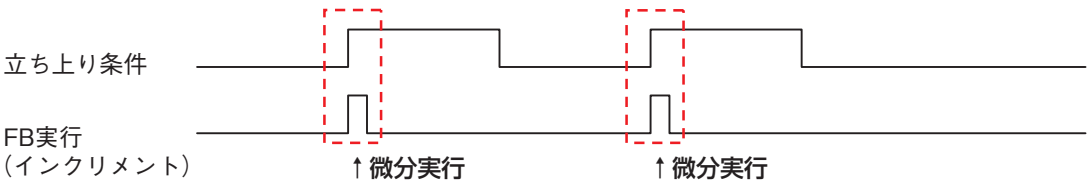
FB外部の変数(アドレス)

変数名	アドレス	値
立ち上り条件	W51.00	ON



FB外部の変数(アドレス)

変数名	アドレス	値
微分実行結果	D900	インクリメント



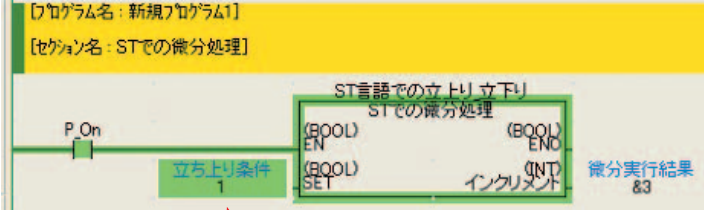
■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	SET	BOOL

変数種別	変数名	データ型
出力変数	インクリメント	INT

変数種別	変数名	データ型
内部変数	Trigger	BOOL

■FBの実行結果



※ 入力信号(立ち上り条件)がONし続けていても、出力の“微分実行結果”の演算結果が微分処理されていることを確認します。

■STプログラム

```
(* *****  
STサンプル  : ST言語での微分回路(立ち上り微分)  
***** *)  
  
IF SET THEN  
  IF Trigger = FALSE THEN  
    (* ***** ここに微分の処理を記述する ***** *)  
    インクリメント:= インクリメント + 1; (* ここではインクリメント処理 *)  
    Trigger := TRUE;  
  END_IF;  
ELSE  
  Trigger := FALSE;  
END_IF;
```

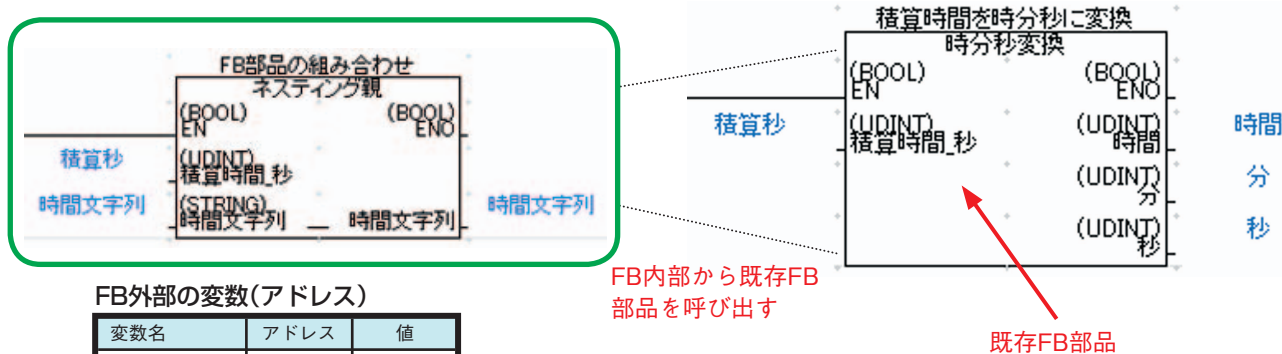
既存FB部品を別のFBから呼び出す ～ FBネスティング～

FBネスティング

■サンプルプログラムの概要

このサンプルプログラムは、既存のFB部品を別のFBから呼び出す(FBネスティング)プログラム例です。
FBネスティングにより、FB部品の組み合わせで新たなFB部品が作成できるようになります。
(パソコン言語でのサブルーチンコールに相当します)

ここでは、FB外部から積算秒のデータを与えて、FB内部から既存のFB部品“時分秒変換FB”を呼び出して“時間”、“分”、“秒”の数値を取得し、その後、“〇〇時△△分□□秒”という時間文字列に変換して出力します。



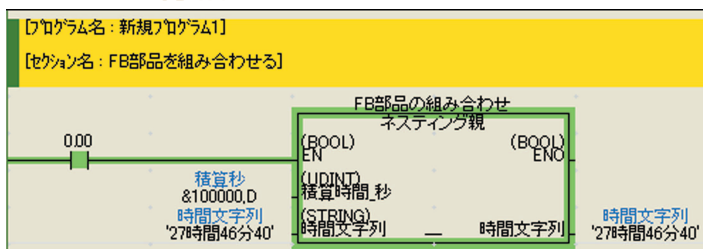
■変数の定義(FB内部)

変数種別	変数名	データ型
入力変数	積算時間_秒	UDINT

変数種別	変数名	データ型
入出力変数	時間文字列	STRING(14)

変数種別	変数名	データ型
内部変数	時分秒データ算出	FUNCTION BLOCK
	時間	UDINT
	分	UDINT
	秒	UDINT
	s時間	STRING(2)
	s分	STRING(2)
	s秒	STRING(2)

■FBの実行結果



※ 既存のFB部品を呼び出すときは、データ型を“FUNCTION BLOCK”を選択します。

※ 算出された時、分、秒の数値データを文字列データとして格納する文字列変数です。

■STプログラム

(* *****)

STサンプル : ネスティング(親)

このFBから時分秒データ算出(既存FB名: 時分秒変換)を呼び出す

(* *****)

時分秒データ算出(積算時間_秒, 時間, 分, 秒);

“時分秒変換”FB 部品の
新しいインスタンス名 入力パラメータ 出力パラメータ

s時間 := UDINT_TO_STRING(時間);

s分 := UDINT_TO_STRING(分);

s秒 := UDINT_TO_STRING(秒);

時間文字列:=CONCAT(s時間,'時',s分,'分',s秒,'秒');

“時分秒変換”という既存FBを“時分秒データ算出”という新たなFBインスタンス名で使用しています。
引数は、FBパラメータの入力、出力の順で記述します。

データ型をUDINT型からSTRING型に変換します。

文字列の結合を行います。

付録

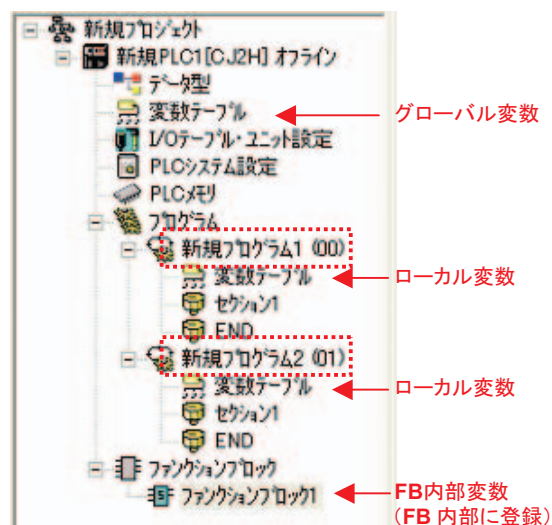
- 付録1 変数の登録方法① グローバル変数／ローカル変数／FB内部変数の登録
変数の登録方法② 配列変数、文字列変数の登録
- 付録2 定数、文字列の記述方法
- 付録3 ウォッチウィンドウ(値のモニタ、設定)への登録方法
- 付録4 データ型一覧
- 付録5 ST言語の構成
 - ・ データ型・演算子一覧
 - ・ ステートメント一覧
 - ・ 関数一覧
 - ・ データ型・数値文字列変換関数

ここでは、CX-Programmerでの変数の登録方法を説明します。

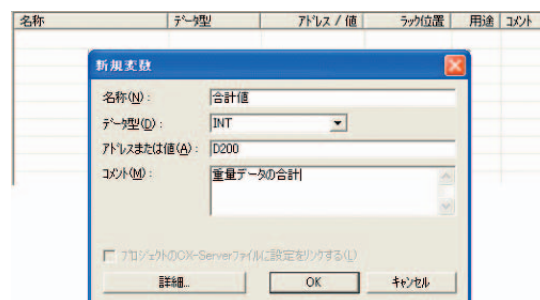
変数は、グローバル変数／ローカル変数(ラダープログラムやFBへの入力・出力で使用する変数)とFB内部で使用する変数の2種類ありますので、それぞれの使用方法、登録の仕方について説明します。

■グローバル変数／ローカル変数の登録方法

1. 右図のように、各種変数をダブルクリックし、変数テーブルを表示します。
 - ・グローバル変数にはプログラム間で共通で使用する変数(またはアドレス)を設定します。(右図の例では、“新規プログラム1”と“新規プログラム2”の両方で使用する変数を登録するときに登録します)
 - ・ローカル変数には1つのプログラム内部でのみ使用する変数を設定します。ローカル変数に登録する場合は、必ず変数名が必要になります。

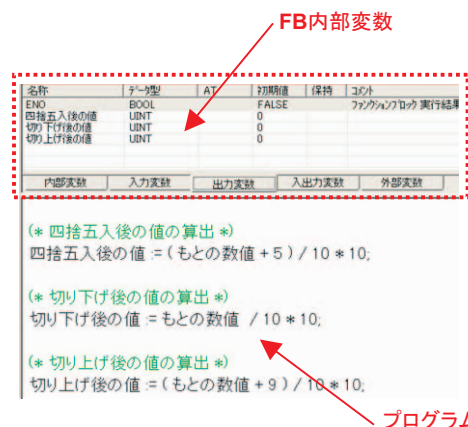


2. 変数テーブル上で右クリックして「変数挿入」または、INSキーを押すと右図のダイアログが表示されます。ここでは、名称(変数名)、データ型、アドレス(変数に割り付けるアドレス)、コメントを入力します。

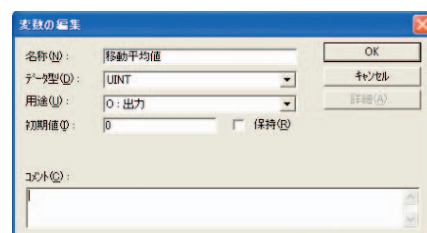


■FB内部変数の登録方法

1. FB内部変数は、FB内部の変数登録画面で行います。右上図では、“ファンクションブロック1”をダブルクリックすると、FB内部変数を登録画面とプログラムを記述する画面が表示されます。FB内部変数はFB内部でのみ有効な変数で、自動的にアドレスが割り付けられる変数です。



2. FB内部変数は、名称(変数名)、データ型の設定に加え、用途(内部変数、入力変数、出力変数、入出力変数、外部変数)を設定します。また、FB内部変数では、初期値、保持オプションが設定できます。



■ 配列変数の登録方法

配列データは、“変数名”と“配列個数”と“1要素のデータ型（データ長）”を設定します。

左記の例では、変数名“abc”、配列個数(5)、1要素のデータ型(INT)で、それぞれの配列に、10、20、30、40、50が格納されている例です。

配列変数は、「内部変数」または「入出力変数」のみに使用可能です。

以下、FB内部変数の登録画面で説明します。

- 右図のように、FB内部変数の変数登録のダイアログを表示し、以下の例で入力します。
 - ・ 名称(変数名)：abc
 - ・ データ型：INT
 - ・ 用途：内部（内部を選択した場合）
(初期値はデフォルトで“0”が登録されています)

- [詳細] ボタンを押すと、右図のダイアログが表示されます。ここで、配列変数にチェックし、配列個数を5と設定します。これにより、下記のように登録されます。

名称	データ型	AT	初期値	保持	コメント
abc	INT(5)		0		

内部変数 入力変数 出力変数 入出力変数

- ・ 変数名：abc
- ・ 配列個数：5個(添え字は[0]～[4])
- ・ 1要素のデータ型(データ長)：INT

abc [0]	abc [1]	abc [2]	abc [3]	abc [4]
(10)	(20)	(30)	(40)	(50)

■ 文字列変数の登録方法

文字列データは、データ型を“STRING”に設定し、文字列の長さ(サイズ)を設定します。

左記の例では、変数名“受信データ”、文字列サイズ(10)の場合で、文字列が“ABCDEFGH”が格納されている例です。

文字列変数は、「内部変数」または「入出力変数」のみに使用可能です。

以下、FB内部変数の登録画面で説明します。

- 右図のように、FB内部変数の変数登録のダイアログを表示し、以下の例で入力します。
 - ・ 名称(変数名)：受信データ
 - ・ データ型：STRING
 - ・ 用途：入出力（入出力を選択した場合）
 - ・ サイズ：10

- これにより、下記のように登録されます。

名称	データ型	AT	初期値	保持	コメント
受信データ	STRING(10)				

内部変数 入力変数 出力変数 入出力変数 外部

- ・ 変数名：受信データ
- ・ データ型：STRING
- ・ サイズ：10

A	B	C	D	E	F	G			
0	1	2	3	4	5	6	7	8	9

■定数の登録方法

以下の構文に従って、定数を16進数、10進数、8進数、2進数で記述することができます。
ここではINT型の変数“var”という変数にすべて1234(10進数)を代入しています。

- ・ (16進数) var := 16#4D2;
- ・ (10進数) var := 1234;
- ・ (8進数) var := 8#2322;
- ・ (2進数) var := 2#010011010010;

■文字列の登録方法

データ型がSTRINGの場合は、文字列データを割り付けることができます。
文字列変数のサイズは半角英数字で1～255文字までで、変数テーブルより変数のサイズを指定できます。
文字列は以下のように指定します。

(Sのデータ型がSTRINGの場合)

- ・ S:='AAA';

文字列はシングルクォーテーションで囲います(例: 'AAA')。
また、大文字/小文字を区別します。

■特殊な文字列、制御コードの登録方法

ドル記号(\$)の後ろにアルファベットを記載したものは、以下のように解釈します。
'\$'記号やそのものやシングルクォーテーションを文字列に含めるときなどに使用します。

- ・ \$\$: ドル記号(ASCIIコード: #24)
- ・ \$': (ASCIIコード: #27)

(* \$記号を文字列で結合する場合 *)

MyStr1 :=CONCAT('abc', '\$\$', 'def');

実行結果は、'abc\$def' となります。

シングルクォーテーションで囲みます。

また、以下の制御コードを扱うときも同様にドル記号の後ろに2桁の数値を記載します。

- ・ \$02: 数値#02(スタートコード)
- ・ \$03: 数値#03(エンドコード)
- ・ \$Lまたは\$L: 改行(ASCIIコード: #0A)
- ・ \$Nまたは\$N: 復帰改行(ASCIIコード: #0D0A)
- ・ \$Pまたは\$P: 改ページまたは新しいページ(ASCIIコード: #0C)
- ・ \$Rまたは\$R: 復帰(ASCIIコード: #0D)
- ・ \$Tまたは\$t: タブ(ASCIIコード: #09)

ウォッチウィンドウ(値のモニタ、設定)への登録方法①

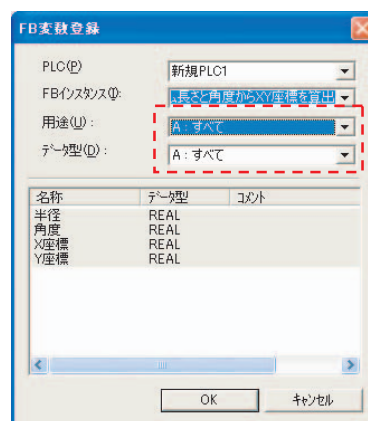
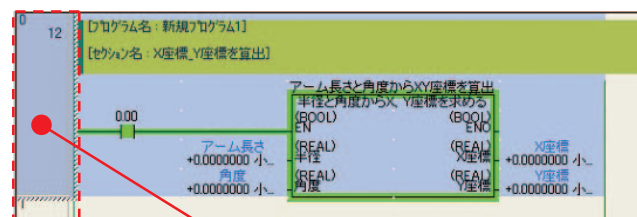
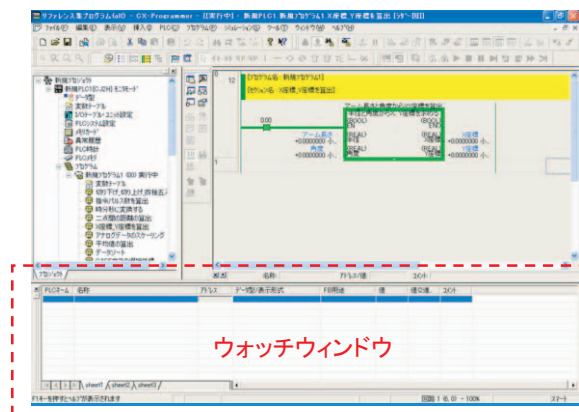
CX-Programmerではウォッチウィンドウで値のモニタや設定ができます。

また、FB内部の変数だけを登録したり、FB外部のアドレスや変数も同時に登録することができます。

右図のように、**ALT + 3** (ウォッチウィンドウ表示のショートカットキー)を押してウォッチウィンドウを表示させます。

■FB外部およびFB内部の変数の登録

1. CX-Programmerをシミュレータなどでオンライン状態にして、右の図の位置をクリックし、回路全体を反転させます。
2. **Ctrl + C** または右クリックから [コピー] を行います。
3. ウォッチウィンドウをクリックした後、**Ctrl + V** または右クリックから [貼り付け] を行います。
4. 右の図のようなFB内部変数を登録するダイアログが表示されます。ここでは、用途、データ型ともに「すべて」を選択します。これにより、FB内部で使用されているすべての変数をウォッチウィンドウに登録することができます。
5. 以下のように、FB外部およびFB内部のすべてのアドレス、変数がウォッチウィンドウに登録されます。



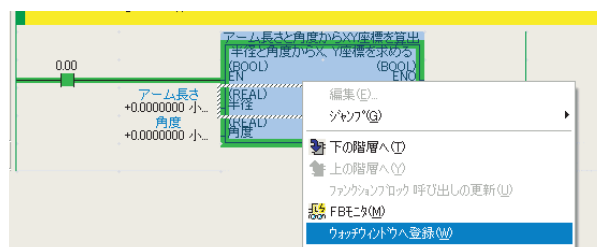
FB外部の変数・アドレス
FB内部の変数

PLC名	名称	アドレス	データ型/表示形式	FB用途	値	値の進	コメント
新規PLC1	新規プログラム1.アーム長さ	0.00	BOOL (On/Off 接点)		1		
新規PLC1	新規プログラム1.X座標	D400	REAL (単精度浮動小数点)		+0.0000	+0.0000	
新規PLC1	新規プログラム1.Y座標	D410	REAL (単精度浮動小数点)		+0.0000	+0.0000	
新規PLC1	新規プログラム1.角度	D402	REAL (単精度浮動小数点)		+0.0000	+0.0000	
新規PLC1	新規プログラム1.X座標	D412	REAL (単精度浮動小数点)		+0.0000	+0.0000	
新規PLC1	アーム長さと角度からXY座標を算出 半径	H621	REAL (単精度浮動小数点)	I: 入力	+0.0000	+0.0000	
新規PLC1	アーム長さと角度からXY座標を算出 角度	H623	REAL (単精度浮動小数点)	I: 入力	+0.0000	+0.0000	
新規PLC1	アーム長さと角度からXY座標を算出 X座標	H625	REAL (単精度浮動小数点)	O: 出力	+0.0000	+0.0000	
新規PLC1	アーム長さと角度からXY座標を算出 Y座標	H627	REAL (単精度浮動小数点)	O: 出力	+0.0000	+0.0000	

■FB内部変数だけの登録

FB内部で使用されている変数だけを登録したい場合は、FB上で右クリックし、[ウォッチウィンドウへ登録]を選択します。

以降は上記4.の操作を行います。

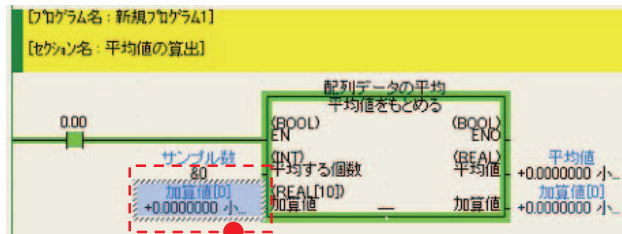


■FB外部変数の配列データの登録

右の図のような配列データをFB外部から与えるときのウォッチウィンドウの登録の仕方は以下のとおりです。

1. 配列データをクリックし、**Ctrl** + **C** または右クリックから[コピー]を行います。

2. ウォッチウィンドウをクリックした後、**Ctrl** + **V** または右クリックから[貼り付け]を行います。
以下のように、配列データ“加算値[0]”が登録されます。



PLCネーム	名称	アドレス	データ型/表示形式	FB用途	値	値の進	コメント
新規PLC1	新規プログラム1 加算値[0]	D20	REAL (単精度浮動小数点)		+0.0000...	+0.0000...	

3. 引き続き、配列データ[1]以降を登録する場合は、リターンキーを入力していくと連続した配列データを登録していくことができます。

編集ダイアログ

PLC(P):

新規PLC1

変数またはアドレス(N):

D20

参照(R)...

データ型/表示形式(D):

REAL (単精度浮動小数点,倍長)

OK

キャンセル



PLCネーム	名称	アドレス	データ型/表示形式	FB用途	値	値の進	コメント
新規PLC1	新規プログラム1 加算値[0]	D20	REAL (単精度浮動小数点)		+0.0000...	+0.0000...	
新規PLC1	新規プログラム1 加算値[1]	D22	REAL (単精度浮動小数点)		+0.0000...	+0.0000...	
新規PLC1	新規プログラム1 加算値[2]	D24	REAL (単精度浮動小数点)		+0.0000...	+0.0000...	
新規PLC1	新規プログラム1 加算値[3]	D26	REAL (単精度浮動小数点)		+0.0000...	+0.0000...	
新規PLC1	新規プログラム1 加算値[4]	D28	REAL (単精度浮動小数点)		+0.0000...	+0.0000...	

データ型	内容	ビット	値の範囲
BOOL	ブール型	1	0 (FALSE)、1 (TRUE)
INT	整数型	16	-32768 ~ +32767
DINT	倍精度整数型	32	-2147483648 ~ +2147483647
LINT	4倍精度整数型	64	-9223372036854775808 ~ +9223372036854775807
UINT	符号なし整数	16	&0 ~ 65535
UDINT	符号なし倍精度整数型	32	&0 ~ 4294967295
ULINT	符号なし4倍精度整数型	64	&0 ~ 18446744073709551615
REAL	実数型	32	$-3.402823 \times 10^{38} \sim -1.175494 \times 10^{-38}$ 、0、 $+1.175494 \times 10^{-38} \sim +3.402823 \times 10^{38}$
LREAL	倍精度実数型	64	$-1.79769313486232 \times 10^{308} \sim -2.22507385850720 \times 10^{-308}$ 、0、 $+2.22507385850720 \times 10^{-308} \sim +1.79769313486232 \times 10^{308}$
WORD	長さ16のビット列型	16	#0000 ~ FFFF または &0 ~ 65535
DWORD	長さ32のビット列型	32	#00000000 ~ FFFFFFFF または &0 ~ 4294967295
LWORD	長さ64のビット列型	64	#0000000000000000 ~ FFFFFFFFFFFFFFFF または &0 ~ 18446744073709551615
STRING	文字列型	可変	—
FUNCTION BLOCK	ファンクションブロックインスタンス	—	—

演算	記号式	演算子で利用できるデータ型
括弧	(式), 配列[識別子]	
関数	関数記号	関数による
べき乗	* *	REAL、LREAL
補数	NOT	BOOL、WORD、DWORD、LWORD
乗算	*	INT、DINT、LINT、UINT、UDINT、ULINT、REAL、LREAL
除算	/	
加算	+	
減算	-	
比較	<, >, <=, >=	BOOL、INT、DINT、LINT、UINT、UDINT、ULINT、WORD、DWORD、LWORD、REAL、LREAL、STRING
等式	=	
不等式	<>	
ブールAND	&	BOOL、WORD、DWORD、LWORD
ブールAND	AND	
ブール排他的論理和	XOR	
ブールOR	OR	

文		機能	例
文の終わり		文を終了する。	;
コメント		(* と、 *)で囲んだ部分がコメントとなる。	(* コメント *)
代入文	代入	左側の変数に、右側の式の結果、変数、値を代入する。	A:= B;
制御文	IF、THEN、ELSIF、ELSE、END_IF	条件式の評価結果が真のとき、文が実行される。	IF <条件式 1> THEN <文 1>; ELSIF <条件式2> THEN <文2>; ELSE <文3>; END_IF;
	CASE、ELSE、END_CASE	整数式の値によって、文が選択される。	CASE <整数式> OF 1: <文1>; 2: <文2>; 3: <文3>; ELSE <文4>; END_CASE;
	FOR、TO、BY、DO、END_FOR	初期値、最終値、増加値の条件によって、文が繰り返し処理される。	FOR <変数> := <初期値> TO <最終値> BY <増加値> DO <文>; END_FOR;
	WHILE、DO、END_WHILE	条件式が真のあいだ、文が繰り返し処理される。	WHILE <条件式> DO <文>; END_WHILE;
	REPEAT、UNTIL、END_REPEAT	条件式が真になるまで、文が繰り返し処理される。	REPEAT <文>; UNTIL <条件式> END_REPEAT;
	EXIT	繰り返し処理を中止する。	EXIT;
	RETURN	呼び出された自ファンクションブロックから、呼び出し元のプログラムの位置に戻ります。	RETURN;
	ファンクションブロック 呼び出し文	ファンクションブロック定義を呼び出す。	<FB 内で使用する場合> データ型がFUNCTION BLOCK の変数名(呼び出し先ファンクションブロック定義の入力変数名:=自身の入力変数名または定数,...,呼び出し先ファンクションブロック定義の出力変数名または定数=>自身の出力変数名,...);

関数名	引数のデータ型	戻り値のデータ型	内容	例
ABS(引数)	INT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL	INT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL	絶対値 引数	a:=ABS(b); (* 変数b の絶対値を変数a に格納 *)
SQRT(引数)	REAL, LREAL	REAL, LREAL	平方根 $\sqrt{\text{引数}}$	a:=SQRT(b); (* 変数b の平方根を変数a に格納 *)
LN(引数)	REAL, LREAL	REAL, LREAL	自然対数 LOG_e 引数	a:=LN(b); (* 変数b の自然対数を変数a に格納 *)
LOG(引数)	REAL, LREAL	REAL, LREAL	常用対数 LOG_{10} 引数	a:=LOG(b); (* 変数b の常用対数を変数a に格納 *)
EXP(引数)	REAL, LREAL	REAL, LREAL	自然指数 $e^{\text{引数}}$	a:=EXP(b); (* 変数b の自然指数を変数a に格納 *)
SIN(引数)	REAL, LREAL	REAL, LREAL	正弦 SIN 引数	a:=SIN(b); (* 変数b のSIN を変数a に格納 *)
COS(引数)	REAL, LREAL	REAL, LREAL	余弦 COS 引数	a:=COS(b); (* 変数b のCOS を変数a に格納 *)
TAN(引数)	REAL, LREAL	REAL, LREAL	正接 TAN 引数	a:=TAN(b); (* 変数b のTAN を変数a に格納 *)
ASIN(引数)	REAL, LREAL	REAL, LREAL	逆正弦 SIN^{-1} 引数	a:=ASIN(b); (* 変数b のASIN を変数a に格納 *)
ACOS(引数)	REAL, LREAL	REAL, LREAL	逆余弦 COS^{-1} 引数	a:=ACOS(b); (* 変数b のACOS を変数a に格納 *)
ATAN(引数)	REAL, LREAL	REAL, LREAL	逆正接関数 TAN^{-1} 引数	a:=ATAN(b); (* 変数b のATAN を変数a に格納 *)
EXPT(<底部>, <指数部>)	底部 REAL, LREAL 指数部 INT, DINT, LINT, UINT, UDINT, ULINT	REAL, LREAL	指数 底部 ^{指数部}	a:=EXPT(b,c); (* 変数b を底部、変数c を指数部とした指数を変数a に格納 *)
MOD	除算対象データ INT, DINT, LINT, UINT, UDINT, ULINT 除算データ INT, DINT, LINT, UINT, UDINT, ULINT	INT, DINT, LINT, UINT, UDINT, ULINT	剰余演算 指定された2つのデータを除算し、その剰余を返します。	戻り値 := MOD(除算対象データ, 除算データ);
SHL	シフト対象データS1 BOOL, WORD, DWORD, LWORD ビット数n INT, DINT, LINT, UINT, UDINT, ULINT	BOOL, WORD, DWORD, LWORD	左ビットシフト ビット文字列左へnビットシフトします。シフトしたとき、右にはゼロが入ります。	戻り値 := SHL(シフト対象データS1, ビット数n);
SHR	シフト対象データS1 BOOL, WORD, DWORD, LWORD ビット数n INT, DINT, LINT, UINT, UDINT, ULINT	BOOL, WORD, DWORD, LWORD	右ビットシフト ビット文字列右へnビットシフトします。シフトしたとき、左にはゼロが入ります。	戻り値 := SHR(シフト対象データS1, ビット数n);

関数名	引数のデータ型	戻り値のデータ型	内容	例
ROL	回転対象データS1 BOOL, WORD, DWORD, LWORD	BOOL, WORD, DWORD, LWORD	左ビット回転 ビット文字列を左へnビット回転します。	戻り値 := ROL(回転対象データS1, ビット数n);
	ビット数n INT, DINT, LINT, UINT, UDINT, ULINT			
ROR	回転対象データS1 BOOL, WORD, DWORD, LWORD	BOOL, WORD, DWORD, LWORD	右ビット回転 ビット文字列を右へnビット回転します。	戻り値 := ROR(回転対象データS1, ビット数n);
	ビット数n INT, DINT, LINT, UINT, UDINT, ULINT			
SEL	選択条件g BOOL	BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL	選択データの抽出 2つのデータから選択条件で指定された1つのデータを選択します。	戻り値 := SEL(選択条件, 抽出対象データS1, 抽出対象データS2);
	抽出対象データS1 BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL			
	抽出対象データS2 BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL			
LIMIT	下限リミットデータ BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL	上下限リミット制御 入力データが上下限リミットデータの範囲内かどうかにより、出力データを制御します。	戻り値 := LIMIT(下限リミットデータ, 入力データ, 上限リミットデータ);
	入力データ BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL			
	上限リミットデータ BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL			
MUX	選択条件n INT, DINT, LINT, UINT, UDINT, ULINT	BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL	マルチプレクサ 選択条件から指定されたデータを抽出する。指定できるデータ数は最大30個。	戻り値 := MUX(選択条件, 抽出対象データS1, 抽出対象データS2, ...);
	抽出対象データS1 BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL			
	抽出対象データS2 BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL			
	:			

関数名	引数のデータ		戻り値のデータ型	内容	例
LEN (文字列)	文字列	STRING	INT	文字列長の検出	a:= LEN (b); (* 変数b の文字列長を変数a に格納 *)
LEFT (<抽出対象文字列>, <抽出文字数>)	抽出対象文字列	STRING	STRING	左からの文字列抽出	a:= LEFT (b,c); (* 変数b の文字列の左端から変数c で指定した文字数分だけ変数a に格納 *)
	抽出文字数	INT, UINT			
RIGHT (<抽出対象文字列>, <抽出文字数>)	抽出対象文字列	STRING	STRING	右からの文字列抽出	a:= RIGHT (b,c); (* 変数b の文字列の右端から変数c で指定した文字数分だけ変数a に格納 *)
	抽出文字数	INT, UINT			
MID (<抽出対象文字列>, <抽出文字数>, <抽出位置>)	抽出対象文字列	STRING	STRING	文字列の抽出	a:= MID (b,c,d); (* 変数b の文字列の変数dで指定した位置から変数c で指定した文字数分だけ変数aに格納 *)
	抽出文字数	INT, UINT			
	抽出位置	INT, UINT			
CONCAT (<結合文字列1>, <結合文字列2>,...) *最大31 個まで指定可能	結合文字列	STRING	STRING	文字列の結合	a:= CONCAT (b,c,...); (* 変数b、変数c の文字列を結合して変数a に格納 *)
INSERT (<挿入対象文字列>, <挿入文字列>, <挿入位置>)	挿入対象文字列	STRING	STRING	文字列の挿入	a:= INSERT (b,c,d); (* 変数b の変数d で指定した位置に変数c の文字列を挿入して変数a に格納 *)
	挿入文字列	STRING			
	挿入位置	INT, UINT			
DELETE (<削除対象文字列>, <削除文字数>, <削除位置>)	削除対象文字列	STRING	STRING	文字列の削除	a:= DELETE (b,c,d); (* 変数b の変数dで指定した位置から変数c で指定した文字数分だけ削除して変数aに格納 *)
	削除文字数	INT, UINT			
	削除位置	INT, UINT			
REPLACE (<置換対象文字列>, <置換文字列>, <置換文字数>, <置換位置>)	置換対象文字列	STRING	STRING	文字列の置換	a:= REPLACE (b,c,d,e); (* 変数b の変数e で指定した位置から変数d で指定した文字数分だけ変数c で置換して変数a に格納 *)
	置換文字列	STRING			
	置換文字数	INT UINT			
	置換位置	INT UINT			
FIND (<検索対象文字列>,<検索文字列>)	検索対象文字列	STRING	INT	文字列の検索	a:= FIND (b,c); (* 変数b の文字列から変数c で指定された文字列を検索し、最初に見つかった文字位置を変数a に格納。見つからなかった場合は変数a に0 を格納。*)
	検索文字列	STRING			
WRITE_TEXT (<書出文字列>,<ディレクトリ名+ファイル名>, <区切り文字>,<パラメータ>)	書出文字列	STRING	—	メモ리카ードへの文字列書込み	WRITE_TEXT (a,b,c,d); (* 変数a で指定した文字列を変数b で指定したディレクトリにテキストファイルとして保存する。変数d が0 の時は追加書込み、1 の時はファイルを新規作成して書き込む *)
	ディレクトリ名+ファイル名	STRING			
	区切り文字	STRING			
	パラメータ	INT UINT WORD			

関数名	引数のデータ		戻り値のデータ型	内容	例
WRITE_TEXT(<書出文字列>,<ディレクトリ名+ファイル名>,<区切り文字>,<パラメータ>)	書出文字列	STRING	—	メモ리카ードへの文字列書込み	WRITE_TEXT(a,b,c,d); (* 変数aで指定した文字列を変数bで指定したディレクトリにテキストファイルとして保存する。変数dが0の時は追加書込み、1の時はファイルを新規作成して書き込む *)
	ディレクトリ名+ファイル名	STRING			
	区切り文字	STRING			
	パラメータ	INT UINT WORD			
TXD_CPU(<送信文字列>)	送信文字列	STRING	—	文字列の送信(CPUユニットのRS-232Cポート)	TXD_CPU(a); (* 変数aで指定した文字列をCPUユニットのRS-232Cポートから送信する *)
TXD_SCB(<送信文字列>,<シリアルポートNo.>)	送信文字列	STRING	—	文字列の送信(SCBのシリアルポート)	TXD_SCB(a,b); (* 変数aで指定した文字列を変数bで指定したSCBのシリアルポートから送信する *)
	シリアルポートNo	INT, UINT, WORD			
TXD_SCU(<送信文字列>,<指定SCUユニット番号>,<シリアルポートNo.>,<内部論理ポートNo.>)	送信文字列	STRING	—	文字列の送信(SCUのシリアルポート)	TXD_SCU(a,b,c,d); (* 変数aで指定した文字列を変数bで指定したユニット番号のSCUの変数cで指定したシリアルポートから送信する。変数dは内部論理ポート番号です *)
	指定SCUユニット番号	INT, UINT, WORD			
	シリアルポートNo	INT, UINT, WORD			
	内部論理ポートNo	INT, UINT, WORD			
RXD_CPU(<受信文字列格納先>,<受信文字数>)	受信文字列格納先	STRING	—	文字列の受信(CPUユニットのRS-232Cポート)	RXD_CPU(a,b); (* CPUユニットのRS-232Cポートに受信したデータを変数bの文字数分の文字列として変数aに出力する *)
	受信文字数	INT, UINT, WORD			
RXD_SCB(<受信文字列格納先>,<受信文字数>,<シリアルポートNo.>)	受信文字列格納先	STRING	—	文字列の受信(SCBのシリアルポート)	RXD_SCB(a,b,c); (* 変数bで指定したSCBのシリアルポートに受信したデータを変数cの文字数分の文字列として変数aに出力する *)
	受信文字数	INT, UINT, WORD			
	シリアルポートNo	INT, UINT, WORD			
DEG_TO_RAD (引数)	REAL, LREAL		REAL LREAL	角度→ラジアン変換	a:= DEG_TO_RAD(b); (* 変数b(角度)をラジアンに変換して変数aに格納 *)
RAD_TO_DEG (引数)	REAL, LREAL		REAL LREAL	ラジアン→角度変換	a:= RAD_TO_DEG(b); (* 変数b(ラジアン)を角度に変換して変数aに格納 *)

■データ型・数値文字列変換関数

■INTからの変換

INT_TO_DINT
 INT_TO_LINT
 INT_TO_UINT
 INT_TO_UDINT
 INT_TO_ULINT
 INT_TO_WORD
 INT_TO_DWORD
 INT_TO_LWORD
 INT_TO_REAL
 INT_TO_LREAL
 INT_TO_BCD_WORD *
 INT_TO_BCD_DWORD *
 INT_TO_STRING *

■DINTからの変換

DINT_TO_INT
 DINT_TO_LINT
 DINT_TO_UINT
 DINT_TO_UDINT
 DINT_TO_ULINT
 DINT_TO_WORD
 DINT_TO_DWORD
 DINT_TO_LWORD
 DINT_TO_REAL
 DINT_TO_LREAL
 DINT_TO_BCD_WORD *
 DINT_TO_BCD_DWORD *
 DINT_TO_STRING *

■LINTからの変換

LINT_TO_INT
 LINT_TO_DINT
 LINT_TO_UINT
 LINT_TO_UDINT
 LINT_TO_ULINT
 LINT_TO_WORD
 LINT_TO_DWORD
 LINT_TO_LWORD
 LINT_TO_REAL
 LINT_TO_LREAL
 LINT_TO_STRING *

■UINTからの変換

UINT_TO_INT
 UINT_TO_DINT
 UINT_TO_LINT
 UINT_TO_UDINT
 UINT_TO_ULINT
 UINT_TO_WORD
 UINT_TO_DWORD
 UINT_TO_LWORD
 UINT_TO_REAL
 UINT_TO_LREAL
 UINT_TO_BCD_WORD *
 UINT_TO_BCD_DWORD *
 UINT_TO_STRING *

■UDINTからの変換

UDINT_TO_INT
 UDINT_TO_DINT
 UDINT_TO_LINT
 UDINT_TO_UINT
 UDINT_TO_ULINT
 UDINT_TO_WORD
 UDINT_TO_DWORD
 UDINT_TO_LWORD
 UDINT_TO_REAL
 UDINT_TO_LREAL
 UDINT_TO_BCD_WORD *
 UDINT_TO_BCD_DWORD *
 UDINT_TO_STRING *

■ULINTからの変換

ULINT_TO_INT
 ULINT_TO_DINT
 ULINT_TO_LINT
 ULINT_TO_UINT
 ULINT_TO_UDINT
 ULINT_TO_WORD
 ULINT_TO_DWORD
 ULINT_TO_LWORD
 ULINT_TO_REAL
 ULINT_TO_LREAL
 ULINT_TO_STRING *

■WORDからの変換

WORD_TO_INT
 WORD_TO_DINT
 WORD_TO_LINT
 WORD_TO_UINT
 WORD_TO_UDINT
 WORD_TO_ULINT
 WORD_TO_DWORD
 WORD_TO_LWORD
 WORD_TO_STRING *

■DWORDからの変換

DWORD_TO_INT
 DWORD_TO_DINT
 DWORD_TO_LINT
 DWORD_TO_UINT
 DWORD_TO_UDINT
 DWORD_TO_ULINT
 DWORD_TO_WORD
 DWORD_TO_LWORD
 DWORD_TO_STRING *

■LWORDからの変換

LWORD_TO_INT
 LWORD_TO_DINT
 LWORD_TO_LINT
 LWORD_TO_UINT
 LWORD_TO_UDINT
 LWORD_TO_ULINT
 LWORD_TO_WORD
 LWORD_TO_DWORD
 LWORD_TO_STRING *

■REALからの変換

REAL_TO_INT
 REAL_TO_DINT
 REAL_TO_LINT
 REAL_TO_UINT
 REAL_TO_UDINT
 REAL_TO_ULINT
 REAL_TO_LREAL

■LREALからの変換

LREAL_TO_INT
 LREAL_TO_DINT
 LREAL_TO_LINT
 LREAL_TO_UINT
 LREAL_TO_UDINT
 LREAL_TO_ULINT
 LREAL_TO_REAL

■BCD_WORDからの変換 *

WORD_BCD_TO_INT
 WORD_BCD_TO_DINT
 WORD_BCD_TO_UINT
 WORD_BCD_TO_UDINT

■BCD_DWORDからの変換 *

WORD_BCD_TO_INT
 DWORD_BCD_TO_DINT
 DWORD_BCD_TO_UINT
 DWORD_BCD_TO_UDINT

■STRINGからの変換 *

STRING_TO_INT
 STRING_TO_DINT
 STRING_TO_LINT
 STRING_TO_UINT
 STRING_TO_UDINT
 STRING_TO_ULINT
 STRING_TO_WORD
 STRING_TO_DWORD
 STRING_TO_LWORD

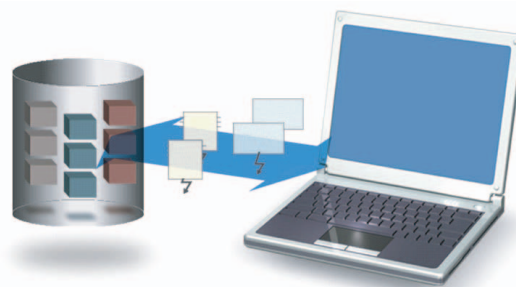
* STRING/BCD_WORD/BCD_DWORDデータ型はCS/CJユニットVer.4.0以降で使用できます。

Webオンラインサービス

ダウンロードサービス

以下のURLより、本サンプルプログラム集と本書で紹介しているSTサンプルプログラムをダウンロードできます。STサンプルプログラムを使ってSTプログラミングの理解にお役立てください。

<http://www.omron-cxone.com/seminar/>



本誌には主に機種のご選定に必要な内容を掲載し、ご使用上の注意事項等は掲載しておりません。ご使用上の注意事項等、ご使用の際に必要な内容につきましては、必ずユーザーズマニュアルをお読みください。

- 本誌に記載の標準価格はあくまで参考であり、確定されたユーザ購入価格を表示したものではありません。
- 本誌に記載の標準価格には消費税が含まれておりません。
- 本誌に記載されているアプリケーション事例は参考用ですので、ご採用に際しては機器・装置の機能や安全性をご確認の上、ご使用ください。
- 本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途への使用をご検討の場合は、定格・性能に対し余裕を持った使い方やフェールセーフ等の安全対策へのご配慮をいただくとともに、当社営業担当者までご相談いただき仕様書等による確認をお願いします。
- 本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物（又は技術）に該当するものを輸出（又は非居住者に提供）する場合は同法に基づく輸出許可、承認（又は役務取引許可）が必要です。

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

●お問い合わせ先

カスタマサポートセンタ

クイック オムロン
0120-919-066

携帯電話・PHSなどではご利用いただけませんので、その場合は下記電話番号へおかけください。

電話 **055-982-5015**（通話料がかかります）

【技術のお問い合わせ時間】

■営業時間：8:00～21:00 ■営業日：365日

■上記フリーコール以外のFAシステム機器の技術窓口：

電話 **055-977-6389**（通話料がかかります）

【営業のお問い合わせ時間】

■営業時間：9:00～12:00/13:00～17:30（土・日・祝祭日は休業）

■営業日：土・日・祝祭日/春期・夏期・年末年始休暇を除く

●FAXによるお問い合わせは下記をご利用ください。

カスタマサポートセンタ お客様相談室 FAX 055-982-5051

●その他のお問い合わせ先

納期・価格・修理・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン営業員にご相談ください。

オムロン制御機器の最新情報をご覧ください。

www.fa.omron.co.jp

緊急時のご購入にもご利用ください。

オムロン商品のご用命は