MITSUBISHI ELECTRIC

三菱电机可编程控制器

MELSEC iQ-R series MELSEC iQ-F series

# MELSEC iQ-R/MELSEC iQ-F 结构化文本(ST)编程指南

# 安全注意事项

(使用之前请务必阅读)

使用本产品之前，应仔细阅读本手册及本手册中所介绍的关联手册，同时在充分注意安全的前提下正确操作。

本手册中记载的注意事项仅与本产品相关。关于可编程控制器系统方面的安全注意事项，请参照所使用CPU模块的用户手册。

在“安全注意事项”中，安全注意事项分为“⚠警告”和“⚠注意”两个等级。

| | |
|---|---|
| ⚠警告 | 表示错误操作可能造成危险后果，导致死亡或重伤事故。 |

| | |
|---|---|
| ⚠注意 | 表示错误操作可能造成危险后果，导致中度伤害、轻伤及设备损失。 |

此外，根据情况不同，即使“⚠注意”这一级别的事项也有可能引发严重后果。

两级注意事项记载的都是重要内容，请务必遵照执行。

请妥善保管本手册以备需要时阅读，并将本手册交给最终用户。

## [设计注意事项]

### ⚠警告

- 应在可编程控制器系统的外部设置互锁电路，以便在通过计算机对运行中的CPU模块进行数据更改、程序更改、状态控制时，能够确保整个系统始终安全运行。此外，通过计算机对CPU模块进行在线操作时，应预先确定由于电缆连接不良等导致发生通信异常时的系统处理方法。

## [网络安全注意事项]

### ⚠警告

- 为了保证系统的网络安全 （可用性、完整性、机密性），对于来自外部设备通过网络的非法访问、拒绝服务攻击(DoS攻击)以及电脑病毒等其他网络攻击，应采取设置防火墙与虚拟专用网络 （VPN），以及在电脑上安装杀毒软件等对策。

# 关于产品的应用

(1) 在使用三菱电机可编程控制器时，应该符合以下条件：即使在可编程控制器设备出现问题或故障时也不会导致重大事故，并且应在设备外部系统地配备能应付任何问题或故障的备用设备及失效安全功能。

(2) 三菱电机可编程控制器是以一般工业用途等为对象设计和制造的通用产品。
因此，三菱电机可编程控制器不应用于以下设备・系统等特殊用途。如果用于以下特殊用途，对于三菱电机可编程控制器的质量、性能、安全等所有相关责任（包括但不限于债务未履行责任、瑕疵担保责任、质量保证责任、违法行为责任、制造物责任），三菱电机将不负责。
- 面向各电力公司的核电站以及其它发电厂等对公众有较大影响的用途。
- 用于各铁路公司或公用设施目的等有特殊质量保证体系要求的用途。
- 航空航天、医疗、铁路、焚烧・燃料装置、载人移动设备、载人运输装置、娱乐设备、安全设备等预计对人身财产有较大影响的用途。

然而，对于上述应用，如果在限定于具体用途，无需特殊质量（超出一般规格的质量等）要求的条件下，经过三菱电机的判断也可以使用三菱电机可编程控制器，详细情况请与当地三菱电机代表机构协商。

(3) 因拒绝服务攻击（DoS攻击）、非法访问、计算机病毒以及其他网络攻击引发的系统方面的各种问题，三菱电机不承担责任。

# 前言

在此真诚感谢您购买了三菱电机可编程控制器MELSEC iQ-R系列与MELSEC iQ-F系列的产品。
本手册用于帮助理解如何使用GX Works3进行结构化文本 （ST）编程等内容。
使用之前应熟读本手册及关联手册，在充分理解MELSEC iQ-R系列与MELSEC iQ-F系列可编程控制器的功能、性能的基础上正确地使用本产品。
此外，将本手册中介绍的程序示例用于实际系统时，要对对象系统进行充分验证以保证在控制方面没有问题。
本手册主要记载了使用MELSEC iQ-R系列时的内容。
关于使用MELSEC iQ-F系列时的注意事项，请参阅以下内容。
☞98页 使用MELSEC iQ-F系列时的注意事项

# 目录

# 关联手册

| 手册名称[手册编号] | 内容 | 提供形式 |
|---|---|---|
| MELSEC iQ-R/MELSEC iQ-F 结构化文本(ST)编程指南 [SH-081465CHN](本手册) | 对在GX Works3中的结构化文本 （ST）编程进行说明。以下通过样本程序对基本操作方法及功能进行说明。 | e-Manual<br>PDF |
| GX Works2 入门指南(结构化工程篇) [SH-080936CHN] | 对在GX Works2中的结构化文本 （ST）编程进行说明。 | PDF |
| 结构化文本 （ST）编程参考手册 [SH-080665CHN] | 对在GX Developer中的结构化文本 （ST）编程进行说明。 | PDF |

本手册未记载ST语言规格的详细内容。

关于详细说明，请参阅以下手册。

MELSEC iQ-R 编程手册(程序设计篇)

MELSEC iQ-F FX5编程手册(程序设计篇)

**要点**

e-Manual是可以使用专用工具进行浏览的三菱电机FA电子书籍手册。

e-Manual具有以下特点。

- 可以从多本手册同时搜索需要的信息 （跨手册搜索）
- 可以通过手册内的链接浏览其他手册
- 可以通过产品插图的各部分浏览想要了解的硬件规格
- 可以将需要频繁浏览的信息登录到收藏夹

# 术语

除特别注明的情况外，本手册中使用以下术语进行说明。

| 术语 | 内容 |
|---|---|
| FB | 可作为程序部件使用的函数。可通过内部变量保持值。要创建实例后使用。 |
| 工程工具 | 用于进行可编程控制器的设置、编程、调试乃至维护的工具。 |
| 程序部件 | 按功能分别定义的程序单位。通过将程序部件化，在分层程序的下位处理中，可按处理内容或功能分成若干单位，并按单位编程。 |
| 函数 | 可作为程序部件使用的函数。对同一输入输出同一结果。 |
| 实例 | 对定义的内部变量分配软元件，并使其为可进行处理、执行的状态的FB的实体。针对同一FB定义，可生成多个实例。 |
| 执行程序 | 指经过转换的程序。可在CPU模块中执行的程序。 |
| 标签 | 用户声明的变量。 |
| 软元件 | 系统对名称、型号、使用方法进行定义的变量。 |

# 总称/简称

在本手册中，除非特别指明，将使用下述总称/简称进行说明。

| 总称/简称 | 内容 |
|---|---|
| FB | 功能块的简称。 |
| FUN | 函数的简称。 |
| GX Developer | MELSEC可编程控制器软件包的产品名。<br>产品型号SW8D5C-GPPW的产品名总称。 |
| GX Works2 | MELSEC可编程控制器软件包的产品名。<br>产品型号SWnDND-GXW2及SWnDNC-GXW2的产品名总称。（n代表版本。） |
| GX Works3 | MELSEC可编程控制器软件包的产品名。<br>产品型号SWnDND-GXW3的产品名总称。（n代表版本。） |
| LD | 梯形图（Ladder Diagram）的简称。 |
| ST | 结构化文本的简称。 |

# 第1部分 ST编程

本部分对MELSEC iQ-R系列与MELSEC iQ-F系列中的ST编程进行说明。

# 1 什么是ST

## 1.1 国际标准IEC 61131-3

IEC 61131是由国际标准团体之一IEC(International Electrotechnical Commission: 国际电工委员会）制定的可编程控制器（PLC：Programmable Logic Controller)系统相关的国际标准。

IEC 61131-3标准规定了5种编程语言 （IL/LD/ST/FBD/SFC），并提出了程序编制的指导方针。

## 1.2 ST语言的特点

ST语言是一种文本格式自由的表述方法，可以像C语言等计算机用编程语言一样进行编程。因为便于对运算处理、数据处理进行表述，因此程序的可视性会更高。

### 算式的运算处理

算术运算和比较运算等可以像一般的表达式那样进行表述。

**■多个运算可写在同1行中**

可以使用运算符 （+、-等）简洁地进行表述，因此程序比梯形图更易于理解。

#### 程序示例

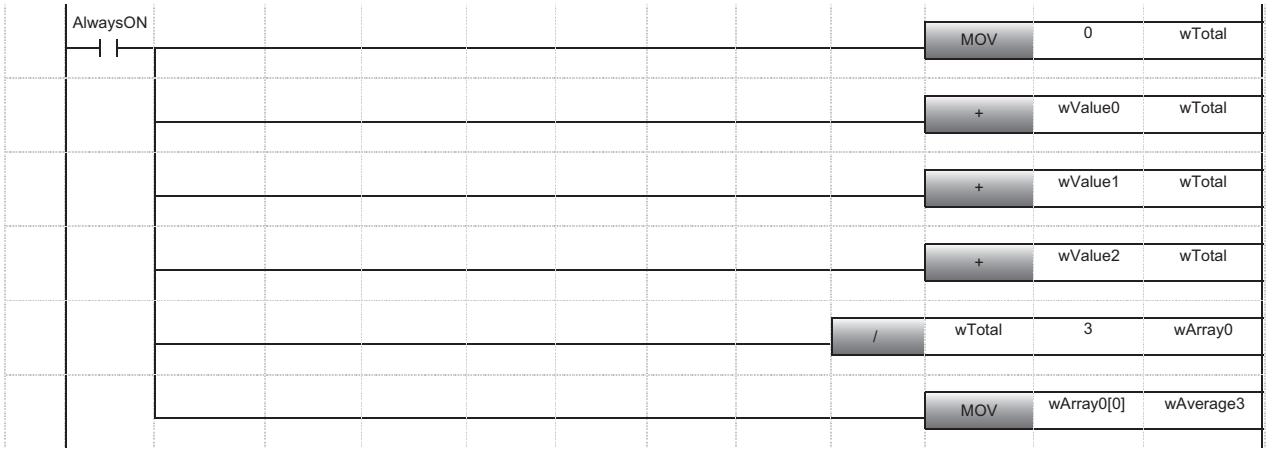
在wAverage3中代入wValue0～wValue2的平均值。

wAverage3 = (wValue0 + wValue1 + wValue2) ÷ 3

**ST**

```
wAverage3 := ( wValue0 + wValue1 + wValue2) / 3;
```

**LD**




**要点**

关于使用ST语言的表述方法，请参照以下内容。

☞18页 运算表达式

### ■无需考虑数据类型的表达式表述

需要考虑小数点以下部分时，其表达式的表述方法相同。实数型数据的复杂运算也可以简洁地进行表述。

#### 程序示例

换算模拟数据。

换算值 = (转换后.上限 - 转换后.下限) ÷ (转换前.上限 - 转换前.下限) × (测定值 - 转换前.下限) + 转换后.下限

**ST**

```
eScalingValue := ( stAfter.eUpperLimit - stAfter.eLowerLimit ) / (stBefore.eUpperLimit - stBefore.eLowerLimit) * (eMeasurements - stBefore.eLowerLimit) + stAfter.eLowerLimit;
```

**要点**

关于使用ST语言的表述方法，请参照以下内容。

☞28页 整数和实数

## 复杂的信息处理

可以通过选择语句或循环语句等语法编写控制程序。与梯形图相比，能够更简洁明了地表述根据不同条件对执行内容进行复杂分支处理或循环处理等。

#### 程序示例

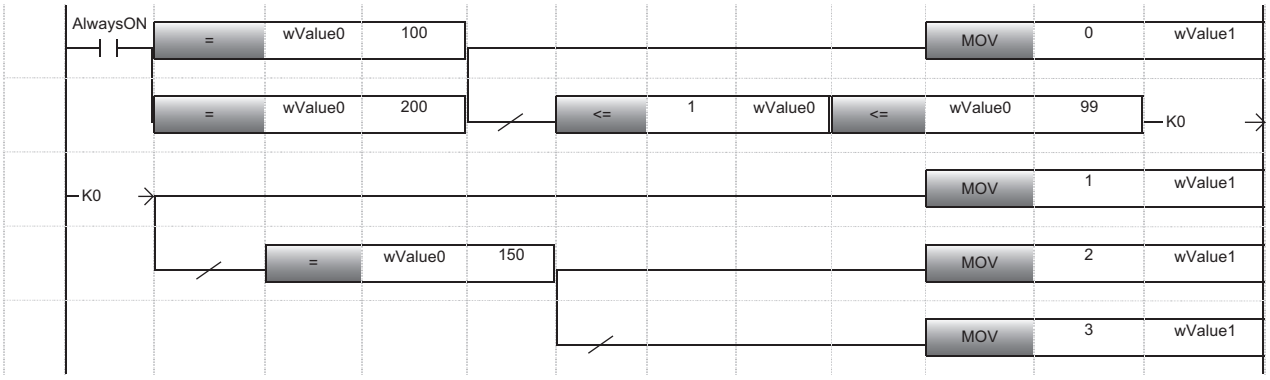
根据wValue0的值在wValue1中设置0～3。

- 100或200时： 0
- 1～99时： 1
- 150时： 2
- 上述以外时： 3

**ST**

```
CASE wValue0 OF
    100, 200: wValue1 := 0;
    1..99: wValue1 := 1;
    150: wValue1 := 2;
    ELSE wValue1 := 3;
END_CASE;
```

**LD**




**要点**

关于使用ST语言的表述方法，请参照以下内容。

☞22页 条件分支，25页 循环处理

# 1.3 与其他语言区分使用、混合使用

IEC 61131-3中所记载的5种编程语言各自的特点如下所示。

| 编程语言 | | 特点 |
|---|---|---|
| IL | 指令列表 | 适用于有内存限制或需要高速处理的情况。可编写紧凑的程序。 |
| LD | 梯形图 | 适用于单纯的继电器顺控处理。 |
| ST | 结构化文本 | 适用于算式的运算处理及复杂的信息处理。是熟悉计算机编程语言的技术人员容易掌握的一种语言。 |
| FBD | FB图 | 适用于连续的模拟信号处理。 |
| SFC | 顺序功能图 | 适用于基于状态转移的顺序控制。 |

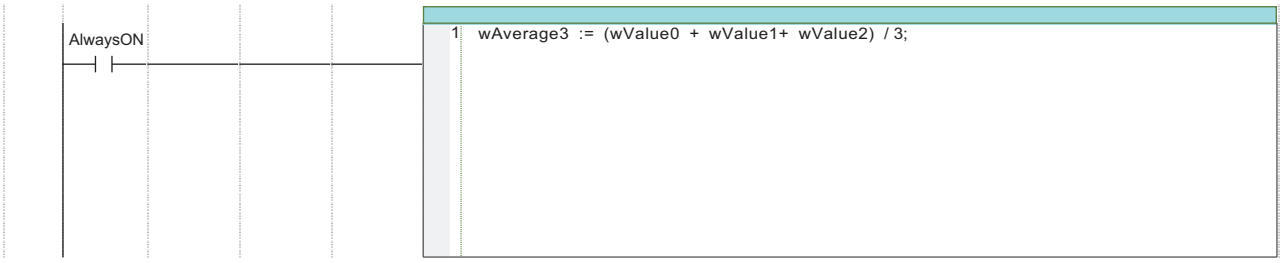
根据不同处理组合使用不同语言来编写程序，可发挥各种编程语言的优势。

在GX Works3中，以下功能中可将其他语言和ST组合使用。

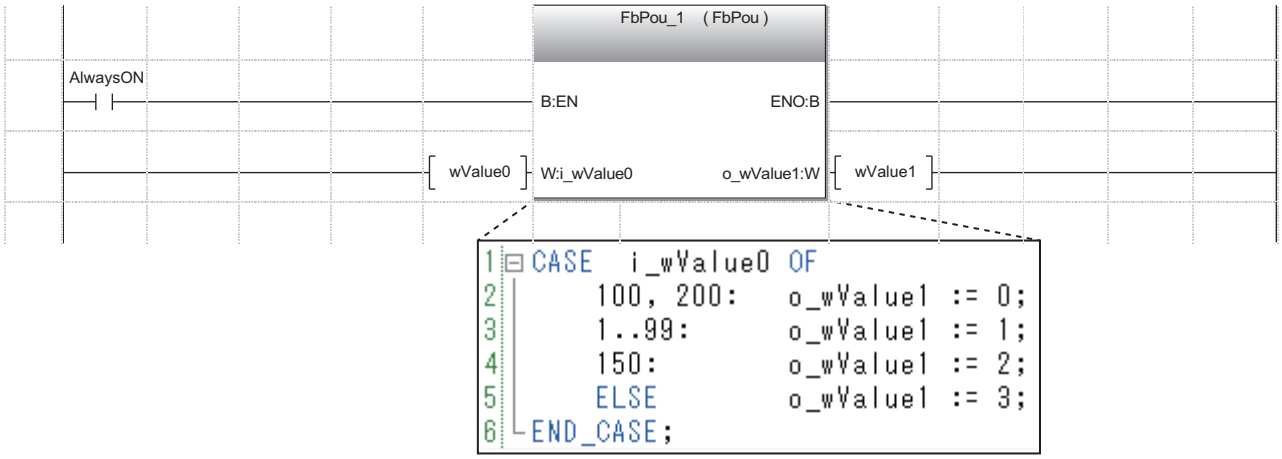
| 功能 | 可组合使用的语言 | 内容 |
|---|---|---|
| 内嵌ST | LD | 继电器顺控处理中的一部分处理使用ST表述。 |
| FUN/FB | LD<br>FBD | 将使用ST表述的子程序定义为程序部件，在LD等的其他程序中调用。 |

将单纯的继电器时序处理或需要对动作进行直观确认的处理使用梯形图进行表述，将部分复杂处理定义成部件使用ST进行表述，可使程序更易于理解。

**使用了内嵌ST的LD**




**使用了FUN/FB的LD**



# 2 编写的基本规则

本章对使用基本ST语言编写程序的方法进行说明。



## 2.1 可使用的字符

ST语言是文本格式的编程语言。本节对使用的字符及符号进行说明。

### 字符编码

GX Works3支持Unicode(UTF-16)的基本多语言面。
日语、英语、中文等多种语言的基本字符及大部分符号除了可以用于注释外，还可以用于标签名或数据名中。

**限制事项**

关于标签名或数据名等中不能使用的字符串 （保留字），请参照以下手册。
GX Works3操作手册

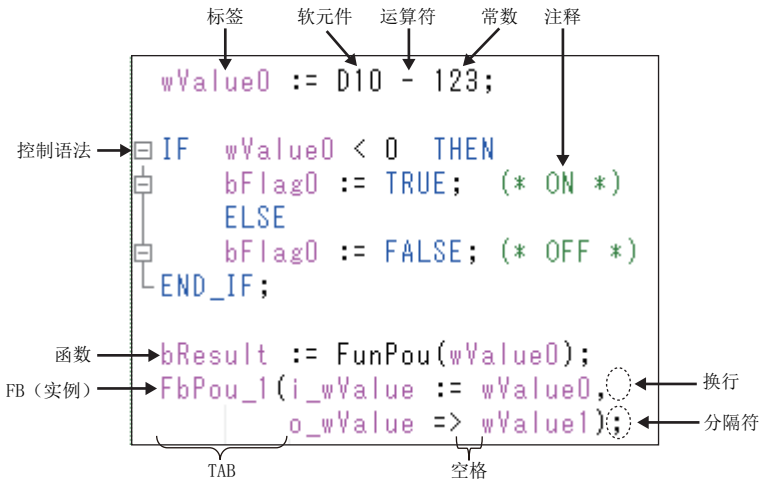
### 构成单位 （标记）

构成程序的单词或符号的最小单位称为标记 （Token）。
ST语言使用以下记号的组合表述程序。

| 类型 | | 示例 | 参照 |
|---|---|---|---|
| 运算符 | | +、-、<、>、=、&、NOT | 88页 运算符 |
| 控制语法的关键字 （定义的标准识别符） | | IF、CASE、WHILE、RETURN | 87页 语句 |
| 识别符 | 变量<br>（标签、软元件等） | X0、Y10、M100、D10、ZR0、<br>bSwitch_A （任意的名称） | 89页 软元件<br>90页 标签 |
| | 程序部件<br>（函数、FB） | BMOV、FunPou(任意的名称)、<br>COUNTER_FB_M_1、FbPou_1(任意的名称) | 93页 函数和FB |
| 常数 | | 123、'字符串'、TRUE、FALSE | 91页 常数 |
| 分隔符 | | ;、(、) | — |

各记号之间可以自由插入空白、换行及注释。

| 类型 | 示例 | 参照 |
|---|---|---|
| 空白 | 空格(全角/半角)、TAB | — |
| 换行 | 换行代码 | — |
| 注释 | (**)、/**/、// | 88页 注释 |



# 2.2 可使用指令和函数

在ST语言中，将梯形图所使用的指令作为函数处理。
可以像C语言等中的函数调用一样使用。


GX Works3中可使用以下函数、FB。

| 类型 | MELSEC-Q/L系列中相当的指令、函数 | |
|---|---|---|
| | GX Works2 | GX Developer |
| CPU模块用指令<br>模块专用指令 | 公共指令 | MELSEC函数 |
| 通用函数<br>通用FB | 应用函数 | IEC函数 |
| 模块FB | MELSOFT Library | — |
| MELSOFT Library （样本库） | | — |
| 任意创建的函数、FB | — | — |

**限制事项**

ST语言不支持一部分并非必要的指令 （触点指令）。
☞95页 ST中无法使用的指令

**要点**

关于指令的详细内容，请参照以下手册。
📖MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)
📖MELSEC iQ-R 编程手册(模块专用指令篇)
📖MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇)
📖各模块的FB参照

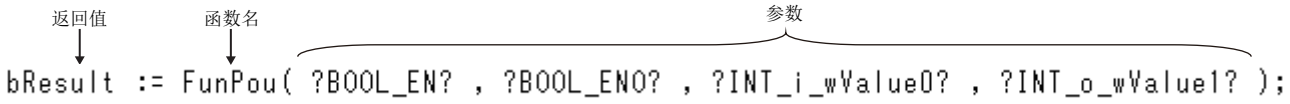
## 什么是函数和FB

函数和FB是对程序中调用的子程序进行了定义的程序部件。

### ■函数

对同一输入，输出同一结果的程序部件。用于将单纯的独立处理定义为部件。

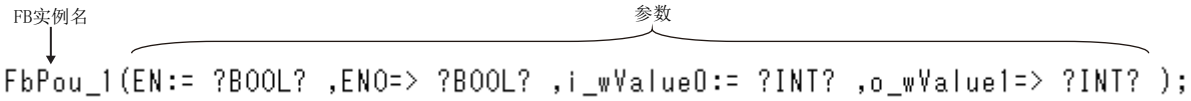
在ST程序中使用时，函数名和参数按如下所述表述。




### ■FB

可将内部保有的值用于运算的程序部件。根据各实例（实体）所保有的值，可对同一输入输出不同结果。用于将比函数更为复杂的处理或需要利用保有的值反复执行的处理等定义部件。

在ST程序中使用时，实例名和参数按如下所述表述。




**要点**

关于使用ST语言的表述方法，请参照以下内容。

☞93页 函数和FB

关于函数和FB的详细内容，请参照以下手册。

📖MELSEC iQ-R 编程手册(程序设计篇)

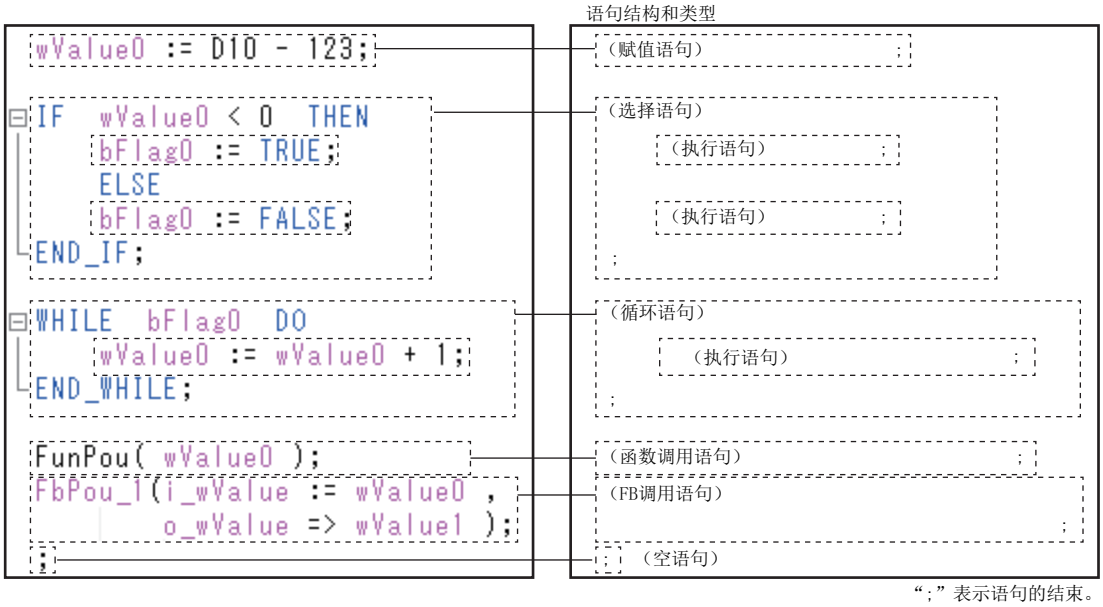
📖MELSEC iQ-F FX5编程手册(程序设计篇)

# 2.3 语句和表达式

本节对ST语言中编写程序时使用的“语句”和“表达式”进行说明。

## 语句

ST语言中，将执行处理的集合称为“语句”。
程序以“语句”为单位编写。
“语句”的结尾处，必须带有“;”（分号）。

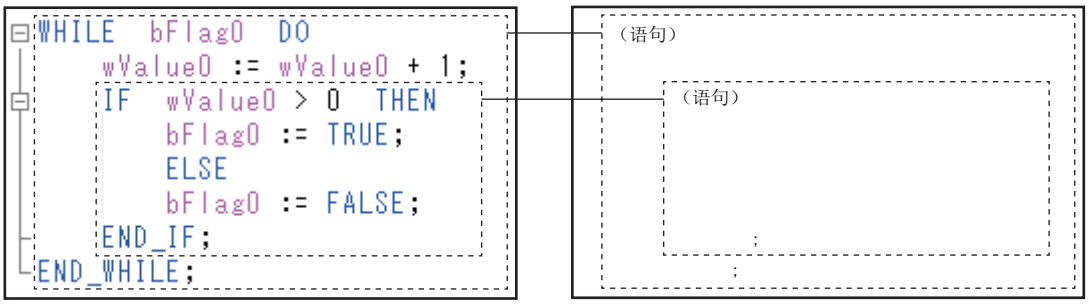



“;”表示语句的结束。

以下所示为语句的类型。

| 类型 | | 内容 | 示例 |
|---|---|---|---|
| 赋值语句 | | 将右边的评价结果代入左边的变量。 | 18页 代入(:=) |
| 控制语法 | 选择语句（IF、CASE） | 根据条件，选择执行语句。 | 22页 条件分支 |
| | 循环语句（FOR、WHILE、REPEAT） | 根据结束条件，多次执行执行语句。 | 25页 循环处理 |
| | 循环语句的中断（EXIT） | 中断循环语句。 | EXIT; |
| 子程序控制语句 | 调用语句 | 调用函数、FB等。 | 93页 函数和FB |
| | RETURN语句 | 中途结束程序。 | RETURN; |
| 空语句 | | 不执行任何处理。 | ; |

控制语法可分层。（可以使用其他的语句作为选择语句或循环语句的执行语句。）

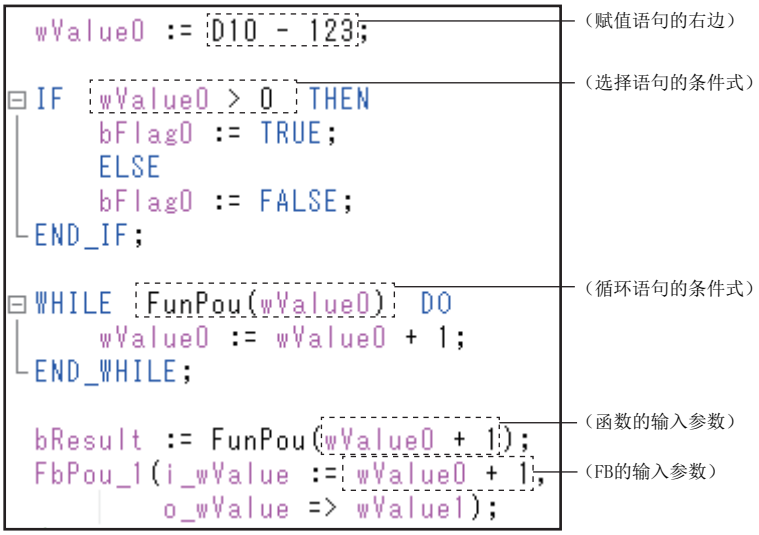


# 表达式

对处理语句时所需的值进行的表述称为“表达式”。
表达式由变量、运算符等构成。表达式的值在程序执行中获得。
表达式用于文中以下所示的位置。

- 赋值语句的右边
- 函数及FB的执行条件 （EN）或输入参数
- 选择语句及循环语句中指定的条件

表达式的使用位置




表达式的数据类型在编译 （转换）时自动判断。表达式的值在程序执行中获得。
算术、比较等运算式可与一般的表达式一样用常数、变量和运算符的组合来表述。
ST语言中，变量和常数也作为“表达式”处理。（基本表达式）
以下所示为表达式的类型。

| 类型 | | 表达式 （运算结果）的数据类型 | 示例 |
|---|---|---|---|
| 运算表达式 | 算术表达式 | 整数、实数等 （依据运算对象） | wValue0 + wValue1 |
| | 逻辑表达式 | BOOL值(TRUE/FALSE) | bFlag0 OR bFlag1 |
| | 比较表达式 | BOOL值(TRUE/FALSE) | wValue0 > 0 |
| 基本表达式 | 变量、常数 | 定义的数据类型 | X0, wValue0, 123, TRUE |
| | 函数调用表达式 | 返回值的数据类型 | FunPou(wValue0, wValue1) |

**要点**

没有返回值的函数不能作为表达式处理。

# 3 使用ST使程序更易于理解

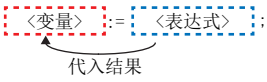
本章以示例对使用ST语言表述可使处理更加简单易读进行说明。

## 3.1 运算表达式

ST语言中，含小数点或指数的复杂运算也能像一般的算术表达式那样简洁地进行表述。

### 代入(:=)

使用赋值语句可以将表达式的结果存入变量。
赋值语句通过:=表述。将右边的表达式的结果代入左边的变量。




**要点**

一般的表达式使用=，但ST语言使用带有:(冒号)的运算符。

### 四则运算(+、-、*、/)

四则运算使用与一般的算术符号相同的运算符(+、-、*、/)表述。
对于使用梯形图指令无法一次性表述完整的运算，可以通过单行表达式简洁地表述出来。

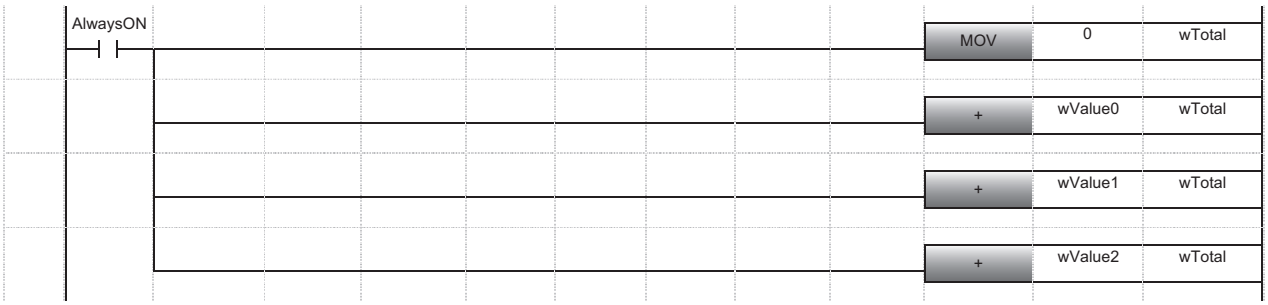
#### 程序示例

在wTotal中代入wValue0～wValue2的和。
wTotal = wValue0 + wValue1 + wValue2

**ST**

```
wTotal := wValue0 + wValue1 + wValue2;
```

**LD**




**要点**

用1个语句汇总表述多个运算表达式时，将从优先级最高的运算符开始处理。
• 四则运算符的优先级 （从高到低）：乘除运算(*、/)，加减运算(+、-)
有多个优先级相同的运算符时，从最左边的运算符开始运算。

小括号()内的运算表达式将优先进行运算。
在复杂的四则运算中，使用小括号()可使优先级清楚明了。

## 程序示例

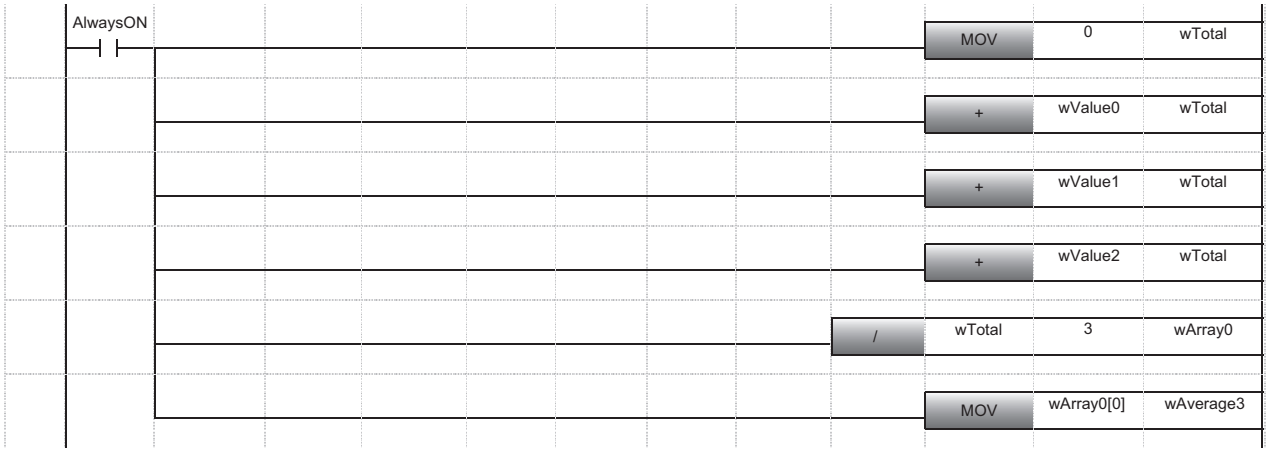
在wAverage3中代入wValue0～wValue2的平均值。

wAverage3 = (wValue0 + wValue1 + wValue2) ÷ 3

**ST**

```
wAverage3 := ( wValue0 + wValue1 + wValue2) / 3;
```

**LD**



# 高级运算（指数函数、三角函数）

指数运算或三角函数运算使用通用函数。

| 类型 | | 函数名 | 示例 | |
|---|---|---|---|---|
| | | | 一般算式表述 | ST |
| 绝对值 | | ABS | B = \|A\| | eValueB := ABS( eValueA ); |
| 平方根 | | SQRT | B = $\sqrt{A}$ | eValueB := SQRT( eValueA ); |
| 自然对数 | | LN | B = log $e^A$ | eValueB := LN( eValueA ); |
| 常用对数 | | LOG | B = log $10^A$ | eValueB := LOG( eValueA ); |
| 指数 | | EXP | B = $e^A$ | eValueB := EXP( eValueA ); |
| 三角函数 | 正弦、反正弦 | SIN、ASIN | B = SIN A | eValueB := SIN( eValueA ); |
| | 余弦、反余弦 | COS、ACOS | B = COS A | eValueB := COS( eValueA ); |
| | 正切、反正切 | TAN、ATAN | B = TAN A | eValueB := TAN( eValueA ); |

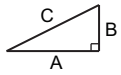
幂乘可使用“**”进行表述。

| 类型 | 运算符 | 示例 | |
|---|---|---|---|
| | | 一般算式表述 | ST |
| 幂乘 | ** | B = $C^A$ | eValueB := eValueC ** eValueA; |

## 程序示例

求直角三角形的斜边长。

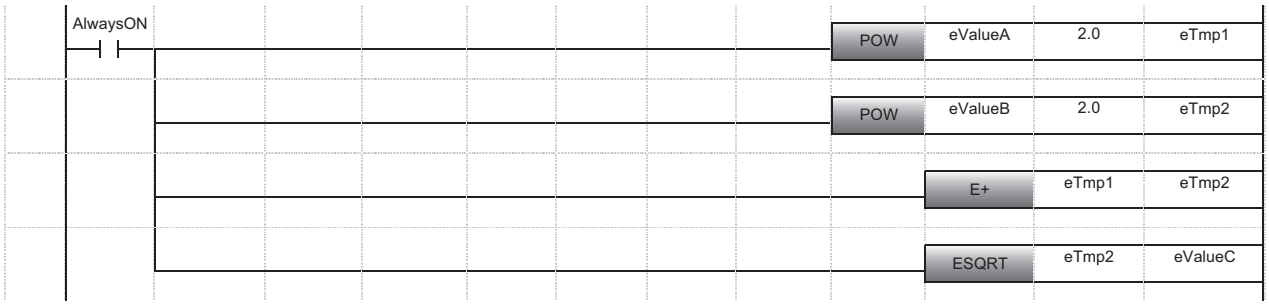
$C = \sqrt{(A^2 + B^2)}$




**ST**

```
eValueC := SQRT((eValueA ** 2.0)+( eValueB ** 2.0));
```

**LD**



# 逻辑运算(AND、OR、XOR、NOT)

逻辑运算不使用符号 （∧、 ∨、 ∀等），而是使用容易输入容易理解的运算符来表述。

## 程序示例

在bResult中代入bFlag0和bFlag1的逻辑与(AND)。

**ST**

```
bResult := bFlag0 AND bFlag1;
```

**LD**


**要点**

逻辑与也可使用运算符“&”表述。

用1个语句汇总表述多个运算表达式时，将从优先级最高的运算符开始处理。

• 逻辑运算符的优先级 （从高到低）：逻辑非 （NOT）、逻辑与 （AND、&）、逻辑异或 （XOR）、逻辑或 （OR）

有多个优先级相同的运算符时，从最左边的运算符开始运算。

# 比较(<, >, <=, >=)、一致/不一致(=, <>)

比较运算使用与一般算术符号相同的等号、不等号的运算符进行表述。

## 程序示例

在bResult中代入wValue0和wValue1的比较结果 （一致时： TRUE、不一致时： FALSE）。

**ST**

```
bResult := wValue0 = wValue1;
```

**LD**


**要点**

“=”号在ST语言中作为对右边和左边的值比较是否一致的运算符使用。

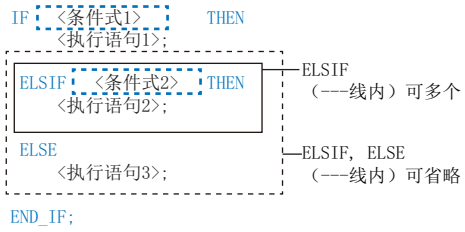
赋值语句应使用带分号的“:=”表述。

# 3.2 条件分支

ST语言中可像C语言等高级编程语言一样使用根据条件进行的分支处理。
使用选择语句 （IF、CASE），可清楚明确地表述不同场合下的执行动作。

## 基于BOOL值的条件分支 （IF）

基于某个条件的真 （TRUE）或假 （FALSE）的分支处理可使用IF语句表述。
IF语句执行以下处理。




**1.** IF的判断
条件式1为真 （TRUE）时，执行执行语句1。

**2.** ELSIF的判断
此前的条件式为假 （FALSE）时，对条件进行判断。
条件式2为真 （TRUE）时，执行执行语句2。

**3.** ELSE的判断
“IF”及“ELSIF”的所有条件式均为假 （FALSE）时，执行“ELSE”后的执行语句3。

**要点**

在IF语句的条件式中，可作以下指定。
- 结果为BOOL值的运算表达式
- BOOL型的变量
- 返回值为BOOL型的函数调用式

可设置多项基于ELSIF的条件分支 （ELSIF<条件式>THEN<执行语句>;）。
根据实际需要使用基于ELSIF、ELSE的条件分支。（可省略）

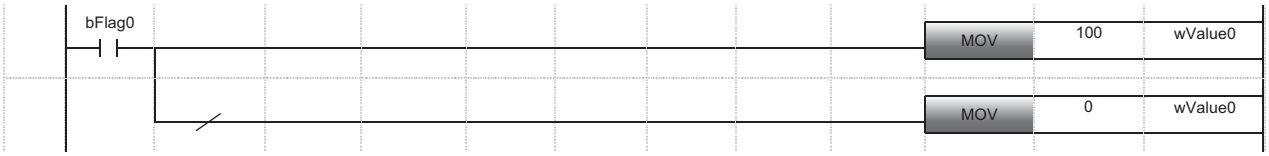
### 程序示例

根据bFlag0的条件，在wValue0中设置不同的值。
- ON时 （bFlag0为TRUE）： wValue0 = 100
- OFF时 （bFlag0为FALSE）： wValue0 = 0

**ST**

```
IF bFlag0 THEN
    wValue0 := 100;
    ELSE
    wValue0 := 0;
END_IF;
```

**LD**



通过将逻辑运算 （AND、OR等）和比较 （<、>、 =等）的组合表达式用于选择语句的条件式，可表述按复杂的条件进行分支的程序。

## 程序示例

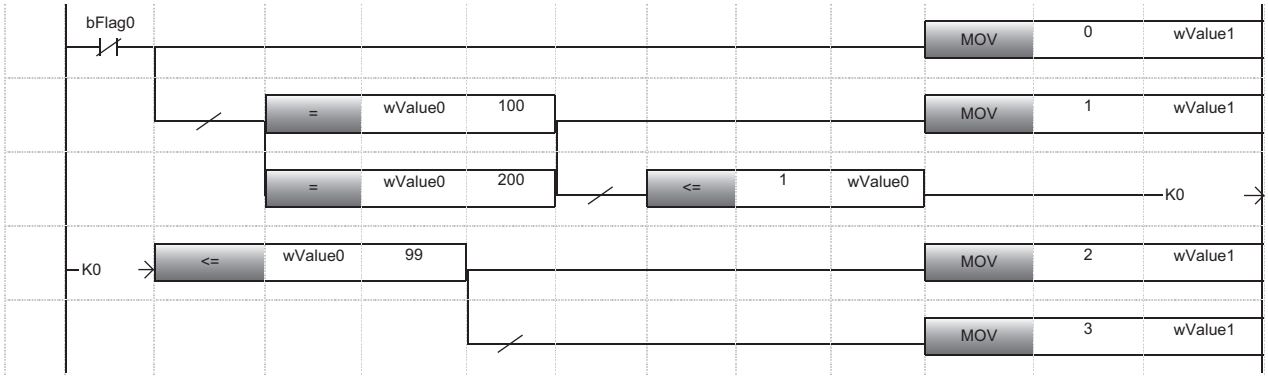
根据bFlag0和wValue0的值在wValue1中设置0～3。

- bFlag0为FALSE时： 0
- bFlag0为TRUE，wValue0为100或200时： 1
- bFlag0为TRUE，wValue0为1～99时： 2
- 上述以外时： 3

**ST**

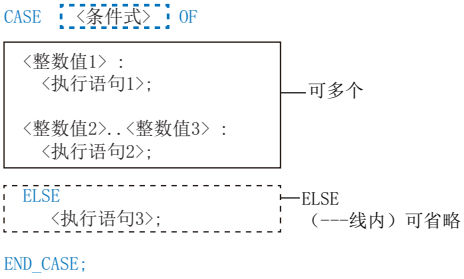
```
IF NOT bFlag0 THEN
    wValue1 := 0;
    ELSIF (wValue0 = 100) OR (wValue0 = 200) THEN
    wValue1 := 1;
    ELSIF (1 <= wValue0) AND (wValue0<= 99) THEN
    wValue1 := 2;
    ELSE
    wValue1 := 3;
END_IF;
```

**LD**



# 基于整数值的条件分支 （CASE）

基于整数值的分支处理可以使用CASE语句简洁明了地进行表述。
CASE语句执行以下处理。




根据整数值的条件，选择执行语句。

**1.** 整数值1的判断

条件式的结果与整数值1一致时，执行执行语句1。

**2.** 对指定了范围的整数值的判断

通过范围指定要判断的整数值时，使用“..”表述。
条件式结果的值在整数值2～整数值3的范围内时，执行执行语句2.

**3.** ELSE的判断

与所有的整数值或范围都不一致时，执行“ELSE”后的执行语句3。

## 要点

在CASE语句的条件式中，可作以下指定。

- 结果为整数 （INT、DINT）值的运算表达式
- 整数 （INT、DINT）型的变量
- 返回值为整数 （INT、DINT）型的函数调用式

可设置多项基于整数值的条件分支 （<整数值>:〈执行语句>;）。
根据实际需要使用基于ELSE的条件分支 （ELSE<执行语句>;）。（可省略）
可对要判断的整数值设置数据类型为字[有符号]或双字[有符号]且类为VAR_CONSTANT的标签。

## 程序示例

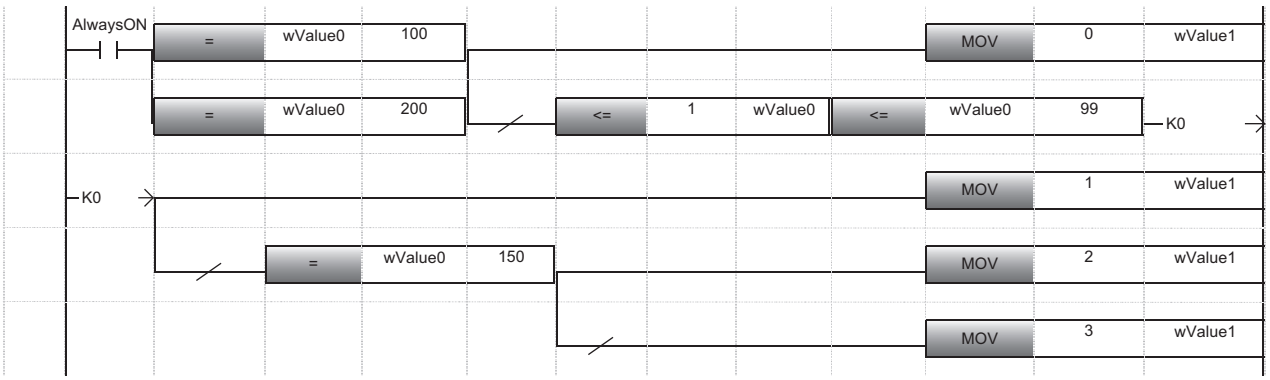
根据wValue0的值在wValue1中设置0～3。

- 100或200时： 0
- 1～99时： 1
- 150时： 2
- 上述以外时： 3

**ST**

```
CASE wValue0 OF
    100, 200: wValue1 := 0;
    1..99: wValue1 := 1;
    150: wValue1 := 2;
    ELSE wValue1 := 3;
END_CASE;
```

**LD**



# 3.3 循环处理

使用循环语句 （WHILE、REPEAT、 FOR）时，可以指定结束条件，重复多次执行处理。

## 通过BOOL条件循环 （WHILE、REPEAT）

基于某个条件的真 （TRUE）或假 （FALSE）的循环处理，可使用WHILE语句或REPEAT语句进行表述。

WHILE语句执行以下处理。

```
WHILE  〈条件式〉
  DO
    〈执行语句〉;
END_WHILE;
```

循环直至变为假（FALSE）

根据BOOL值的结束条件，多次执行执行语句。
反复执行直到条件式的结果变为假 （FALSE）为止。

**1.** 条件的判断

条件式为假 （FALSE）时，结束处理。

**2.** 处理的执行

条件式为真 （TRUE）时，执行执行语句并重复处理。

REPEAT语句执行以下处理。

```
REPEAT
  〈执行语句〉;
  UNTIL  〈条件式〉
END_REPEAT;
```

循环直至变为真（TRUE）

根据BOOL值的结束条件，多次执行执行语句。
反复执行直到条件式的结果变为真 （TRUE）为止。

**1.** 处理的执行

执行执行语句。

**2.** 条件的判断

条件式为真 （TRUE）时，结束处理。
条件式为假 （FALSE）时，重复处理。

**要点**

在WHILE语句、REPEAT语句的条件式中，可作以下指定。

- 结果为BOOL值的运算表达式
- BOOL型的变量
- 返回值为BOOL型的函数调用式

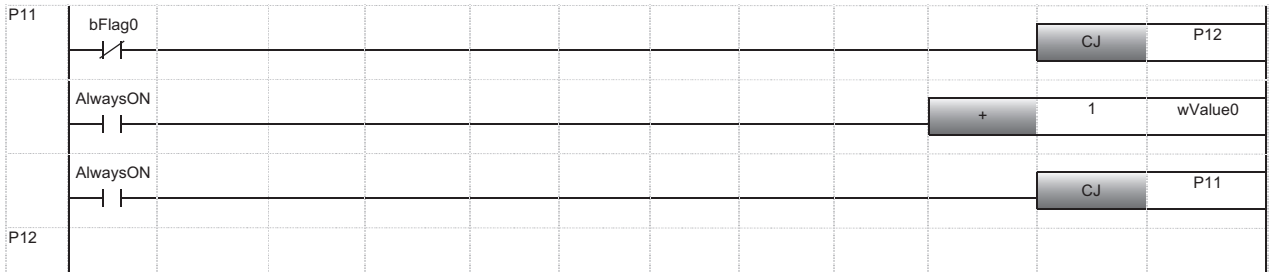
### 程序示例

bFlag0为TRUE时，wValue0加1。

**ST**

```
WHILE bFlag0 DO
    wValue0 := wValue0 + 1;
END_WHILE;
```

**LD**



通过将逻辑运算 （AND、OR等）和比较 （<、>、 =等）的组合表达式用于循环语句的条件式，可表述复杂的结束条件的程序。

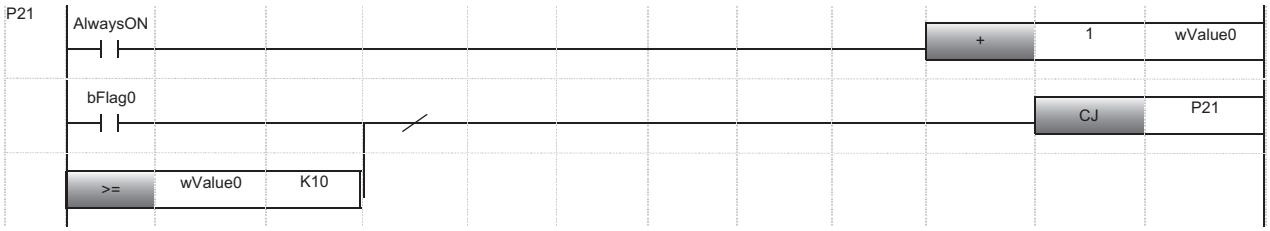
## 程序示例

bFlag0为TRUE或wValue0小于10时，wValue0加1。

**ST**

```
REPEAT
    wValue0 := wValue0 + 1;
    UNTIL bFlag0 OR (wValue0 >= 10)
END_REPEAT;
```

**LD**



# 重复执行直到变量达到设定值 （FOR）

整数型变量在达到条件值之前重复执行的处理可以使用FOR语句进行表述。

FOR语句执行以下处理。

```
FOR  〈变量〉 :=〈初始值（表达式）〉
  TO 〈最终值（表达式）〉
  BY 〈增加值（表达式）〉      —BY（---线内）可省略
  DO 〈执行语句〉;
END_FOR;
```

根据整数值的结束条件，多次执行执行语句。

设置了初始值的整数型变量在达到最终值之前，重复执行。

**1.** 数据的初始化

对作为条件的变量设置初始值。

**2.** 条件的判断

变量为最终值时，结束处理。

**3.** 处理的执行

执行执行语句。

**4.** 增加值的加法运算

在变量上加上增加值，重复处理。

## 要点

FOR语句的初始值、最终值及增加值可指定如下。

- 结果为整数 （INT、DINT）值的运算表达式
- 整数 （INT、DINT）型的变量
- 返回值为整数 （INT、DINT）型的函数调用式

应转换数据类型以使值变为整数型。

增加值的加法运算处理 （BY〈增加值 （表达式）〉;）在增加值为1时，表述可以省略。

设置为条件的变量在FOR语句结束后，将保持结束时的值。

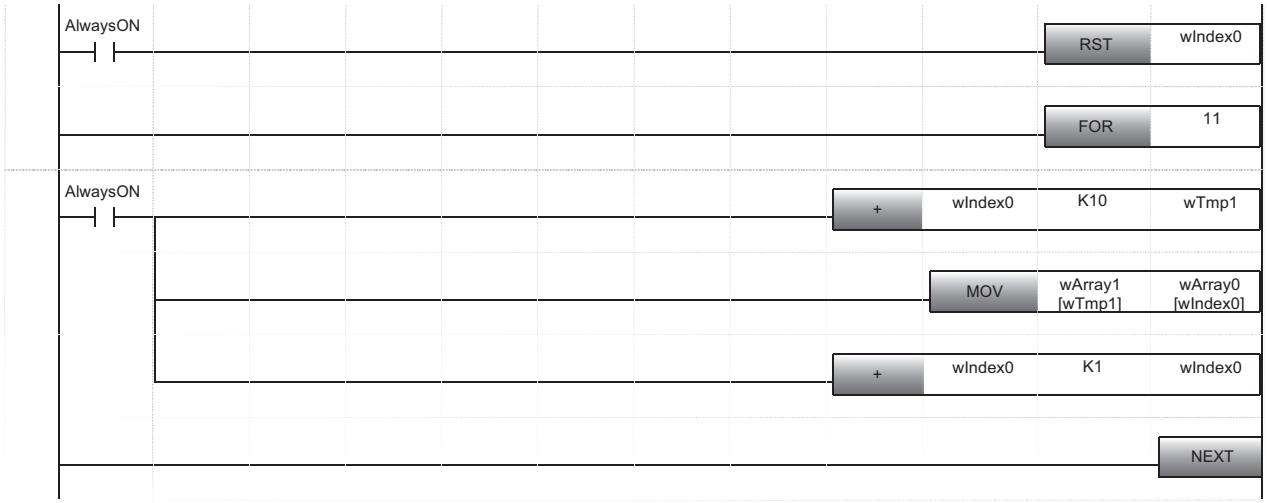
## 程序示例

在数组wArray0的元素0～10中设置数组wArray1的元素10～20的值。

**ST**

```
FOR wIndex0 := 0 TO 10 BY 1 DO
    wArray0[ wIndex0 ] := wArray1[ wIndex0 + 10 ];
END_FOR;
```

**LD**




## 要点

将数组型数据和循环语句结合，可以对数组的多个元素执行同一数据处理。

# 4 多种数据类型的处理

本章对在ST语言中处理各数据类型的变量时的注意事项、指定/转换数据类型进行表述的方法进行说明。
标签的数据类型在登录标签时定义。软元件的数据类型视软元件类型而定。

**要点**

要以不同于定义的数据类型使用值时，应使用通用函数的类型转换函数。

## 4.1 BOOL值

BOOL是表述0和1的1位数据类型。
用于表示位软元件的ON/OFF、执行结果的真 （TRUE）/假 （FALSE）。
位软元件、设置了“位”的数据类型的标签按BOOL值的变量处理。

## 4.2 整数和实数

字软元件、设置了整数型或实数型的数据类型的标签按整数或实数的变量处理。
各处理中使用的值的基本数据类型为整数型。要处理小数点以下的值时，使用实数型的变量。

### 值的范围

根据变量的不同类型，其有效位数也不同。应根据要执行的运算，指定合适的数据类型的变量。
以下所示为各数据类型的变量在运算中可处理的值的范围。

| 数据类型 | | | 值的范围 |
|---|---|---|---|
| 整数 | 字[无符号]/位列[16位] | WORD | 0～65535 |
| | 双字[无符号]/位列[32位] | DWORD | 0～4294967295 |
| | 字[有符号] | INT | -32768～32767 |
| | 双字[有符号] | DINT | -2147483648～2147483647 |
| 实数 | 单精度实数 | REAL | $-2^{128}$～$-2^{-126}$、0、$2^{-126}$～$2^{128}$ |
| | 双精度实数 | LREAL | $-2^{1024}$～$-2^{-1022}$、0、$2^{-1022}$～$2^{1024}$ |

# 自动进行的类型转换

在GX Works3的ST语言中，整数或实数型的情况下，在以下处理中即使使用了不同数据类型的变量或常数，也会自动进行类型转换。

- 赋值语句
- 向函数或FB传递输入参数
- 算术运算式

向大于可处理值的范围的数据类型转换时，执行自动类型转换。

例

代入字[有符号]（INT型）变量和双字[有符号]（DINT型）变量时

| ST | 运算结果 | |
|---|---|---|
| dValue0 := wValue1; | ○ | INT向DINT的类型转换会自动进行。 |
| wValue1 := dValue0; | × | DINT向INT的转换可能会造成数据丢失，因此转换时会出现错误。 |
| wValue1 := DINT_TO_INT(dValue0); | ○ | DINT向INT的转换需使用类型转换函数。<br>转换前的值超出INT型的范围时，会出现运算错误。 |

**要点**

以下情况时，不会自动进行类型转换。应使用类型转换函数。

- 数据大小相同、符号不同的整数型之间的类型转换
- 向数据丢失类型的类型转换
- 与整数型、实数型以外的类型转换

## 可自动转换的数据类型

自动类型转换的数据类型组合如下所示。

| 转换前的数据类型 | 转换后的数据类型 | 备注 |
|---|---|---|
| 字[有符号] | 双字[有符号] | 自动转换为符号扩展的值。 |
| | 单精度实数 | 自动转换为与转换前相同的值。 |
| | 双精度实数 | |
| 字[无符号]/位列[16位] | 双字[有符号] | 自动转换为零扩展的值。 |
| | 双字[无符号]/位列[32位] | |
| | 单精度实数 | 自动转换为与转换前相同的值。 |
| | 双精度实数 | |
| 双字[有符号]<br>双字[无符号]/位列[32位] | 双精度实数 | |
| 单精度实数 | | |

在向通用函数、通用FB及指令传递输入参数时，也会自动进行类型转换。

输入参数被定义为泛型数据类型时的自动类型转换数据类型的组合如下所示。

| 参数所指定的变量（转换前）的数据类型 | 输入参数的数据类型定义 | 转换后的数据类型 |
|---|---|---|
| 字[有符号] | ANY32、ANY32_S | 双字[有符号] |
| | ANY_REAL、ANY_REAL_32 | 单精度实数 |
| | ANY_REAL_64 | 双精度实数 |
| 字[无符号]/位列[16位] | ANY32、ANY32_U | 双字[无符号]/位列[32位] |
| | ANY_REAL、ANY_REAL_32 | 单精度实数 |
| | ANY_REAL_64 | 双精度实数 |
| 双字[有符号]<br>双字[无符号]/位列[32位] | ANY_REAL、 ANY_REAL_64 | 双精度实数 |
| 单精度实数 | ANY_REAL_64 | |

# 算术表达式的运算结果的数据类型

ST语言的算术运算表达式（四则运算）的运算结果与运算对象的变量或常数的数量类型相同。（幂乘（**）时，运算结果为实数型。）

**要点**

运算符的左边和右边的数据类型不同时，运算结果按容量较大一侧的数据类型处理。

• 运算结果的数据类型的优先级（从高到低）：双精度实数、单精度实数、双字、字

整数的二元运算中，[有符号]和[无符号]不能同时存在。

## 注意事项

运算结果超出了数据类型所能处理的值的范围时，在以后的处理中将无法反映正确的结果（数值）。应事先将运算对象变量的数据类型转换成可对运算结果进行处理的范围的数据类型。

例

字[有符号]（INT型）的算术运算时

| ST | 运算结果 | |
|---|---|---|
| dValue0 := wValue1 * 10; | × | 运算结果超出INT型的范围（-32768～32767）时，会代入上溢或下溢的运算结果。 |
| dValue1 := INT_TO_DINT(wValue1);<br>dValue0 := dValue1 * 10; | ○ | 运算是以DINT型进行的，因此不会发生上溢或下溢。 |
| dValue1 := INT_TO_DINT(wValue1) * 10; | ○ | |

# 整数和实数的除法运算

根据变量的类型，除法运算的运算结果可能会有所不同。

整数型变量的除法运算会舍去小数点以下的部分。

实数型变量的除法运算会求取至小数点以下的有效位为止。

- 单精度实数： 有效位数7位（小数点以下6位）
- 双精度实数： 有效位数15位（小数点以下14位）

例

将表达式“(2 ÷ 10) × 10”的运算结果代入D0时

| 数据类型 | | ST | 运算结果 |
|---|---|---|---|
| 整数 | 字[有符号] | D0 := (2 / 10) * 10; | 0 |
| 实数 | 单精度实数 | D0:E := (2.0 / 10.0) * 10.0; | 2.0 |

**要点**

ST语言中，对于字软元件，可在“:E”等的类型指定中明确指定数据类型进行使用。

（☞89页 字软元件的类型指定）

## 除法运算的余数（MOD）

整数除法运算的余数可通过余数运算（MOD）运算符求取。

例

求5位整数的后2位时

| 数据类型 | | ST | 运算结果 |
|---|---|---|---|
| 整数 | 字[有符号] | D0 := (-32368 MOD 100); | -68 |

# 4.3 字符串

字符串型的变量的处理使用字符串处理指令和通用函数（字符串函数）。

### 程序示例

求字符串的长度。

**ST**

```
wLen0 := LEN(sString0);
```

## 字符串的代入

字符串型变量可以使用赋值语句进行表述。

### 程序示例

在字符串型变量sString0和字符串[Unicode]型变量wsString1中代入字符串“ABC”。

**ST**

```
sString0 := 'ABC';
wsString1 := "ABC";
```

**要点**

ASCII字符串常数使用单引号（'）括起来。
Unicode字符串常数使用双引号（"）括起来。

## 字符串的比较

字符串型变量的比较运算（比较、一致、不一致）可以使用运算符进行表述。

### 程序示例

比较字符串，不一致的情况下将sString1代入sString0。

**ST**

```
IF sString0 <> sString1 THEN
    sString0 := sString1;
END_IF;
```

**要点**

字符串型变量的情况下，比较运算符（<、>、<=、>=）会以ASCII或Unicode的编码值进行比较。
按第一个不同的字符编码的大小，确定字符串的大小。
比较条件的详细内容与字符串比较指令（LD$<等）相同。

# 4.4 时间

时间的数据按时间型变量或时钟数据 （CPU模块用指令所使用的数组数据）处理。

## 时间型 （TIME）变量

使用时间型变量，能够以ms单位简洁明了地用时间数据进行编程。
时间型的常数按以下格式进行表述。

- T#23d23h59m59s999ms

| 时间单位 | d(日) | h(时) | m(分) | s(秒) | ms(毫秒) |
|---|---|---|---|---|---|
| 范围 | 0～24 | 0～23 | 0～59 | 0～59 | 0～999 |

时间型变量可在T#-24d20h31m23s648ms～T#24d20h31m23s647ms的范围内进行设置。
关于时间长度的记述方法的详细内容，请参阅以下内容。
☞92页 时间长度的记述方法

### 时间的代入

时间型变量可以使用赋值语句进行表述。

#### 程序示例

将1小时30分的值代入时间型变量tmData0中。

ST

```
tmData0 := T#1h30m;
```

### 时间的比较

时间型变量的比较运算 （比较、一致、不一致）可以使用运算符进行表述。

#### 程序示例

时间为1天以上时，结束处理。

ST

```
IF tmData0 >= T#1d THEN
    RETURN;
END_IF;
```

### 时间的四则运算

时间型变量的乘法运算、除法运算使用通用函数 （时间数据型函数）进行运算。
加减法运算可用运算符进行表述。

#### 程序示例

在时间型变量tmData0上加上10ms，乘以2并将所得值代入tmData0中。

ST

```
tmData0 := MUL_TIME(tmData0 + T#10ms, 2);
```

# 时钟数据 （日期数据、时间数据）

时钟数据是CPU模块用指令所使用的数组数据。

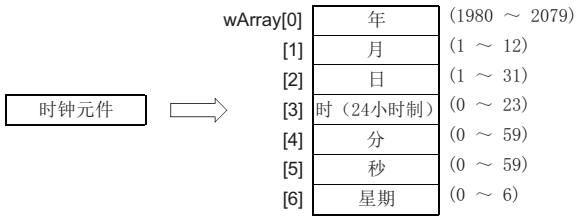
- 时钟数据： 存储年、月、日、时、分、秒及星期的数组
- 扩展时钟数据： 在时钟数据中增加了毫秒单位的数组
- 日期数据： 仅时钟数据的年、月、日的数组
- 时间数据： 仅时钟数据的时、分、秒的数组

## 时钟用指令和时钟数据

使用应用指令的时钟用指令，可对时钟数据 （字[有符号]数据的数组）进行处理。

### 程序示例

从CPU模块的时钟元件中读取“年、月、日、时、分、秒、星期”。




**ST**

```
DATERD( TRUE, wArray);
```

## 日期数据、时间数据的比较

ST中不能使用日期数据、时间数据的比较指令。（☞95页 ST中无法使用的指令）

要对日期数据、时间数据进行比较时，应对数组的各元素进行比较或转换为时间型变量后进行比较。

# 4.5 数组和结构体

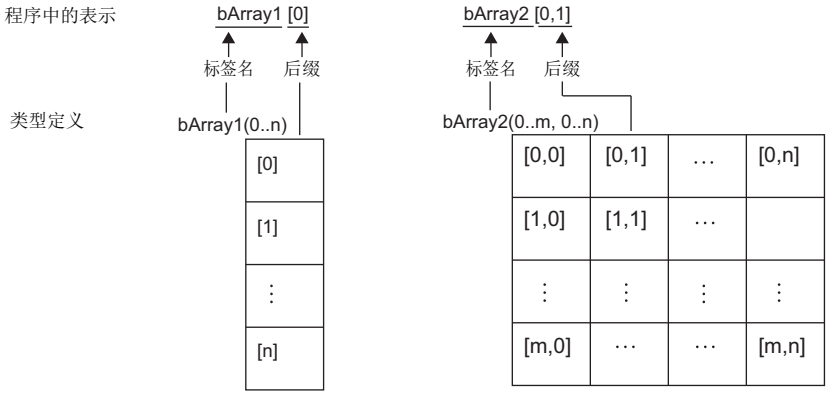
数组和结构体是将多个数据作为一个整体处理的数据格式。

- 数组： 同一数据类型的变量的连续集合。
- 结构体： 不同数据类型的变量的集合。

## 数组

表示数组型变量整体时，仅使用标签名。

要表示数组的各元素时，在标签名后面附上“[ ]”，并在“[ ]”内填上要指定的元素编号。




### 数组的代入

将数组型变量记述在赋值语句中时，值将会代入附带元素编号所指定的元素中。

如果没有附带元素编号，则可以对数组整体进行赋值 （数组的复制）。

#### 程序示例

在字[有符号]数据的数组型变量wArray1中代入以下内容。

- 元素0： 10
- 元素1： 二维数组wArray2的元素[0,1]

ST

```
wArray1[0] := 10;
wArray1[1] := wArray2[0,1];
```

#### 程序示例

将wArray0的所有元素代入字[有符号]数据的数组型变量wArray1的所有元素中。

（只有在右边和左边的数组的数据类型及数据数完全一致时才可实施。）

ST

```
wArray1 := wArray0;
```

### 数组的表达式

数组的各元素可以作为定义了数据类型的变量由运算表达式进行指定。（可以进行比较或四则运算等。）

没有附带元素编号的数组变量不能用运算表达式进行指定。

ST语言中，可以用表达式来表述数组元素。

#### 程序示例

在字[有符号]数据的数组型变量wArray1中代入以下内容。

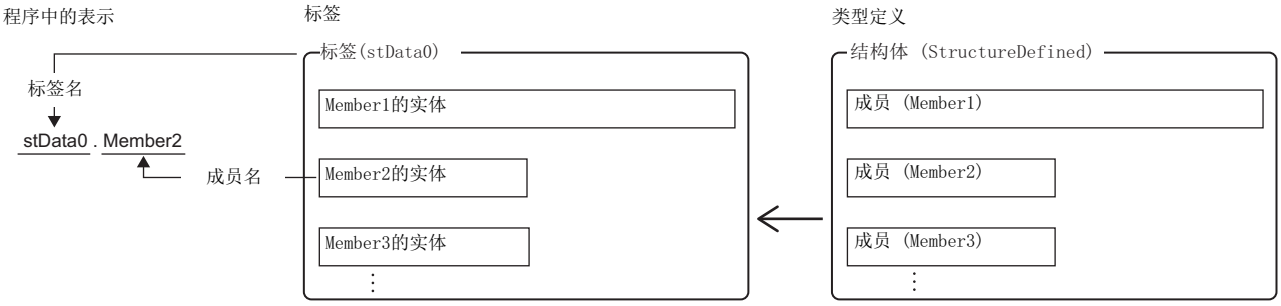
- 元素wIndex0+1：wArray0的元素0和元素1的和

ST

```
wArray1[wIndex0 +1] := wArray0[0] + wArray0[1];
```

# 结构体

结构体是将多个不同数据类型汇聚起来，作为1个数据类型定义的集合。
结构体的类型定义的名称及各成员的名称均可以任意设置，因此非常有利于数据管理。
表示结构体型变量整体时，仅使用标签名。
要表示结构体的各成员时，在标签名后面加上“.”再加上成员名。




## 结构体的代入

将结构体型变量记述在赋值语句中时，值将会代入指定成员中。
如果没有指定成员，则可以对结构体整体进行赋值 （结构体的复制）。

### 程序示例

在结构体型变量stData0的成员中代入以下内容。

- Member1(单精度实数)：10.5
- Member2(位)：TRUE

ST

```
stData0.Member1 := 10.5;
stData0.Member2 := TRUE;
```

### 程序示例

在结构体型变量stData1的所有元素中代入stData0的所有元素。
（仅在右边和左边的结构体的数据类型 （结构体型定义）完全一致时才可实施。）

ST

```
stData1 := stData0;
```

# 组合了结构体和数组的数据类型

ST程序中也可以使用成员中包含了数组的结构体或结构体型的数组。

### 程序示例

在结构体型变量stData0的数组型成员wArray中代入数组型变量wArray1。（均为字[有符号]、元素数3的数组）

ST

```
stData0.wArray := wArray1;
```

### 程序示例

对结构体数组型变量stArray0的元素编号1的成员和元素编号0的成员的值进行比较。

ST

```
bResult := stArray0[1].Member1 > stArray0[0].Member1;
```

**要点**

GX Works3还可进行结构体数组的代入。

# 5 使用ST表述梯形图

本章对使用ST语言表述梯形图程序的梯形图符号或顺序指令等的方法进行说明。

## 5.1 表述触点、线圈

在ST语言中，对使用了梯形图的触点、线圈的回路可以用逻辑与 （AND）、逻辑或 （OR）、逻辑非 （NOT）等逻辑运算和赋值语句进行表述。

### 常开触点和线圈

常开触点和线圈的程序可以用赋值语句进行表述。

#### 程序示例

根据bFlag0的ON/OFF，bResult0进行ON/OFF。

**ST**

```
bResult0 := bFlag0;
```

**LD**


**要点**

赋值语句通过:=表述。将右边的表达式的结果代入左边的变量。

### 常闭触点 （NOT）

常闭触点可以用逻辑非 （NOT）的运算符进行表述。

#### 程序示例

bFlag0为ON时，bResult0为OFF，bFlag0为OFF时，bResult0为ON。

**ST**

```
bResult0 := NOT bFlag0;
```

**LD**

# 串联连接、并联连接 （AND、OR）

梯形图中以串联连接、并联连接表示的条件，可以用逻辑与 （AND、&）、逻辑或 （OR）的运算符进行表述。

## 程序示例

以下任一条件成立时，bResult0为ON。

- 条件1： bFlag0为ON且bFlag1为ON
- 条件2： bFlag2为ON

**ST**

```
bResult0 := bFlag0 AND bFlag1 OR bFlag2;
```

**LD**


**要点**

用1个语句汇总表述多个运算表达式时，将从优先级最高的运算符开始处理。

- 逻辑运算符的优先级 （从高到低）：逻辑与(AND、&)、逻辑异或(XOR)、逻辑或(OR)

有多个优先级相同的运算符时，从最左边的运算符开始运算。

# 执行顺序比较复杂的触点、线圈

ST语言还可用于表述复杂的触点线圈的组合处理。执行顺序难以理解时，应使用小括号()，以使优先级简洁明了。

## 程序示例

以下条件1及条件2成立时，bResult0为ON。

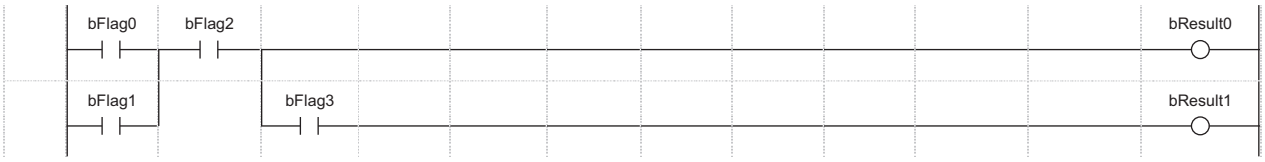
以下条件1、条件2及条件3都成立时，bResult1为ON。

- 条件1： bFlag0或bFlag1为ON
- 条件2： bFlag2为ON
- 条件3： bFlag3为ON

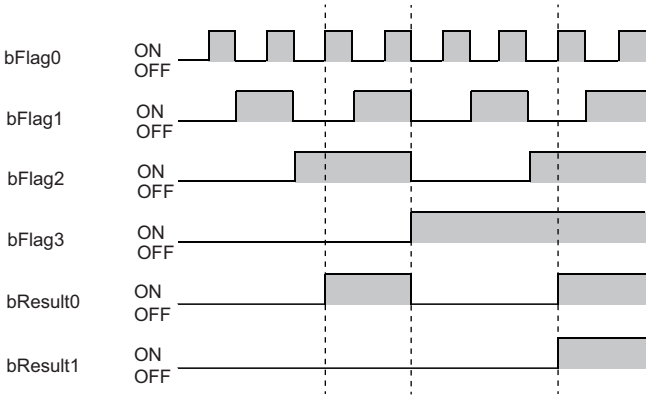
**ST**

```
bResult0 := (bFlag0 OR bFlag1) AND bFlag2;
bResult1 := bResult0 AND bFlag3;
```

**LD**




执行了上述程序时，各软元件的ON/OFF时序如下所示。



# 5.2 指令表述

ST语言中，将可以与梯形图通用的指令作为函数处理。
ST语言所不支持的部分指令（☞95页 ST中无法使用的指令）可以使用运算符等ST语言特有的格式进行表述。

## 梯形图和ST均可以使用的指令

原则上，各指令在梯形图和ST中均可以使用。（☞14页 可使用指令和函数）
根据各指令的定义，作为函数进行表述。

### 程序示例

bFlag0为OFF→ON时，bResult0为1次扫描ON。

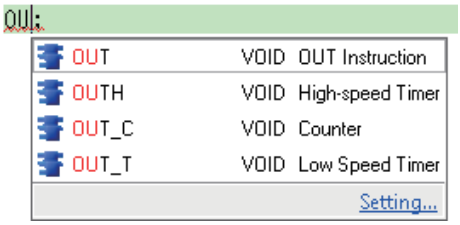
**ST**

```
PLS(bFlag0, bResult0);
```

**LD**

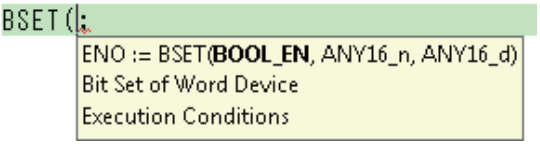

### 输入指令名时的注意事项

在梯形图和ST中，有些指令虽然功能相同，但指令名却不同。（输出指令的低速定时器指令“OUT_T”等）
在ST编辑器中输入指令名后，会一览显示以该字符开始的指令。
应确认后输入可在ST中使用的指令名。




### 输入参数时的注意事项

相同的指令在梯形图和在ST中的参数的顺序可能会不同。
在ST编辑器中输入参数时，会通过工具提示的方式显示指令格式。
应根据工具提示，输入参数。




关于指令的详细内容，请参照编程手册。光标停留在指令上时按F1后，会显示该指令的页面。

# 赋值可使用的指令

通用函数的类型转换函数和赋值语句可使用数据转换指令。
字符串传送指令 （$MOV）等可以使用字符串型标签的赋值语句进行表述。
☞95页 赋值可使用的指令

## 程序示例

将单精度实数eValue0转换成有符号BIN16位数据wValue1。

**ST**

```
wValue1 := REAL_TO_INT( eValue0 );
```

**LD**


## 程序示例

将字符串型变量sString0传送 （代入）至sString1。

**ST**

```
sString1 := sString0;
```

**LD**


# 可以使用运算符表述的指令

基本指令的比较运算指令、算术运算指令可以使用运算符进行表述。
☞95页 可以使用运算符表述的指令

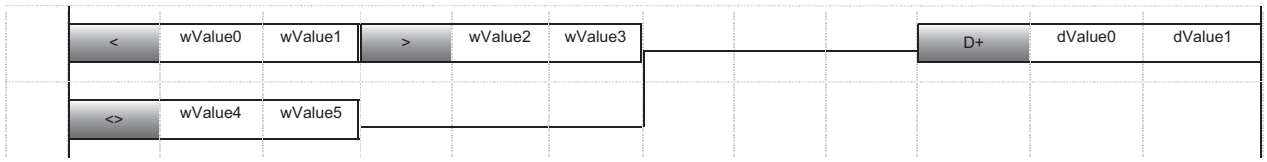
## 程序示例

根据字[有符号]（INT型）变量的比较结果，加上双字[有符号]（DINT型）变量的2个值。

**ST**

```
IF (wValue0 < wValue1 ) AND (wValue2 > wValue3 ) OR (wValue4 <> wValue5 ) THEN
    dValue1 := dValue0 + dValue1;
END_IF;
```

**LD**



## 可以使用控制语法、FUN/FB表述的指令

循环语句 （☞25页 循环处理）可以使用结构化指令FOR～NEXT进行表述。
不能通过子程序指令 （CALL等）或指针分支指令 （CJ、SCJ、JMP）对指定了指针的程序进行分支。在ST语言中，应通过选择语句、函数或FB以使程序结构化。
☞97页 控制语句、函数等可使用的指令

**要点**

ST语言中，无需END指令。（☞97页 不需要的指令）

# 5.3 声明、注解的表述

梯形图的声明、注解作为注释进行表述。
ST语言的注释可以在任意位置进行表述，因此在使用上比声明、注解更具灵活性。

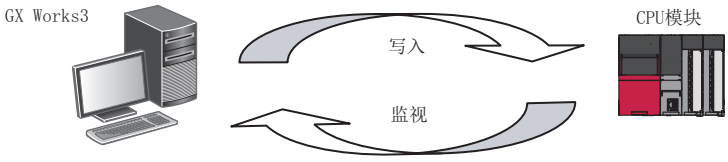
```
(* The processing is performed depending on the ten-key input. *)
IF G_wTenKey <> c_wNONE THEN
        CASE G_wTenKey OF
                0..9 : (* For number-key input (0 to 9) *)
                (* Add the input numeric value to the end of the display value. *)
                IF G_eDecimal = 0.0 THEN
                        (* For integer part *)
                        G_eDisplayValue := (G_eDisplayValue * 10) + G_wTenKey;
                        ELSE
                        (* For after decimal point *)
                        G_eDisplayValue := G_eDisplayValue + (G_eDecimal * G_wTenKey);
                        G_eDecimal := G_eDecimal * 0.1;
                END_IF;
                10: (* For input of decimal point key *)
                G_eDecimal := 0.1;
                11..14: (* For input of addition, subtraction, multiplication, or division key (11 to 14) *)
                (* Retain the operation type *)
                G_wOperation := G_wTenKey - 10;
                (* Move the display value to the previous operation value and then reset the displayed value *)
                G_eLastValue := G_eDisplayValue;
                G_eDisplayValue := 0.0;
                G_eDecimal := 0.0;
                15: (* For equal-key input *)
                (* Add, subtract, multiply, or divide the displayed value to/from the current value. *)
                G_eLastValue := Calculation(G_eLastValue, G_wOperation, G_eDisplayValue);
                (* Assign the rounding result to the display value. *)
                G_eDisplayValue := FractionProcessing(G_eLastValue, G_wSwitch1, G_wSwitch2);
                G_wOperation := 0; (* Clear the operation type *)
                G_eDecimal := 0.0;
        END_CASE;
        (* Clear the key input*)
        G_wTenKey := c_wNONE;
END_IF;
```

# 6 程序创建步骤

## 6.1 步骤概要

本节对程序的创建步骤进行说明。

1. 打开ST编辑器。
2. 编辑ST程序。
3. 转换程序，进行调试。
4. 确认机器的实际动作。




**要点**

关于GX Works3的操作方法的详细内容，请参照以下手册。

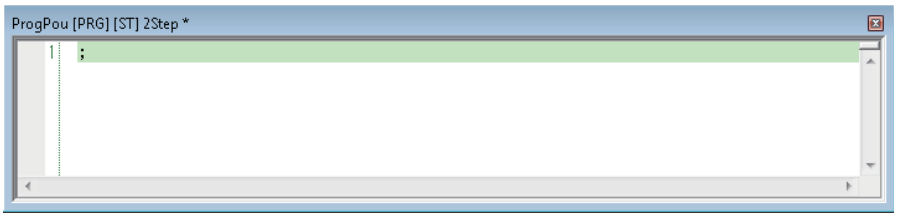
GX Works3操作手册

## 6.2 打开ST编辑器

ST程序使用工程工具（GX Works3）的编程功能进行创建。

应在程序语言中选择“ST”，创建工程或程序。

ST编辑器




## 6.3 编辑ST程序

对输入简单的程序并执行一系列操作的步骤进行说明。

例如：试着输入以下程序。

### 程序示例

```
wValue0 := D10 - 123;

IF  wValue0 < 0  THEN
    bFlag0 := TRUE;  (* ON *)
    ELSE
    bFlag0 := FALSE; (* OFF *)
END_IF;

bResult := FunPou(wValue0);
FbPou_1(i_wValue := wValue0,
        o_wValue => wValue1);
```

# 输入文本

在ST编辑器中，可通过与一般的文本编辑器相同的操作输入文本。

## 操作步骤

```
wValue0 := D10 - 123;
```

**1.** 输入“wValue0 := D10 - 123;”。

**要点**

按Ctrl+Shift+=，可输入赋值符号 （:=）。

除了通过[Edit （编辑）]菜单进行的操作外，还可通过输入以下标准键进行操作。

- Shift+Delete/Shift+Insert：剪切/粘贴
- Alt+Back space/Ctrl+Back space：撤销/恢复

## 编辑器间的复制

可以与一般的文本编辑器等之间进行文本数据的复制/粘贴，可进行如下应用。

- 从其他的文本编辑器或PDF等中复制文本，将其粘贴至ST编辑器中以便引用程序
- 将ST编辑器中创建的程序复制到其他的文本编辑器中，创建文件

**要点**

GX Works3中，在ST编辑器上按下Alt的同时拖动时会出现矩形选择框，可通过该矩形选择框进行选择。

# 输入控制语法

输入控制语法 （IF语句）。

## 操作步骤

```
IF
```

**1.** 输入“IF”。

**2.** 通过Ctrl+F1可显示语法模板。

```
IF  ?Condition?  THEN
    ?Statement? ;
ELSE
    ?Statement? ;
END_IF;
```

```
IF  wValue0 < 0  THEN
    ?Statement? ;
ELSE
    ?Statement? ;
END_IF;
```

**3.** 将光标移动到“?条件?”上。
（通过Ctrl+Alt+←可移动光标。）

**4.** 输入“wValue0 < 0”作为条件式。

```
IF  wValue0 < 0  THEN
    bFlag0 := TRUE;
ELSE
    bFlag0 := FALSE;
END_IF;
```

**5.** 将光标移动到“?执行语句?”上。
（通过Ctrl+Alt+→可移动光标。）

**6.** 输入“bFlag0 := TRUE;”作为执行语句。

- 即使输入小写的“true”，也会自动转换成大写的“TRUE”。

同样，输入“bFlag0 := FALSE;”。

**要点**

使用了以下功能。

- [Edit （编辑）]⇨[Display Template （模板显示）]（Ctrl+F1）
- [Edit （编辑）] ⇨[Mark Template(Left) （模板参数选择(左)）]/[Mark Template(Right) （模板参数选择(右)）]（Ctrl+Alt+←/→）

## 折叠显示

单击显示在控制语法左边的图标（ ⊟ ），可将控制语法折叠显示（ ⊞IF...END_IF;）。

# 输入注释

在ST程序中输入注释。

### 操作步骤

```
⊟IF  wValue0 < 0  THEN
     bFlag0 := TRUE;  ON
     ELSE
     bFlag0 := FALSE;
 END_IF;
```

**1.** 在程序的任意位置（例如：“bFlag0 := TRUE;”之后），输入注释文字。

• 可自由插入空格或TAB。

```
⊟IF  wValue0 < 0  THEN
⊟    bFlag0 := TRUE;  (* ON *)
     ELSE
⊟    bFlag0 := FALSE; (* OFF *)
 END_IF;
```

**2.** 在注释文字的前后，需输入注释的分割符号（“(*”、“*)”）。

被符号包围的内容，将作为注释处理。

同样，在“FALSE”之后输入“(* OFF *)”。

### 要点

GX Works3中，可使用与C语言相同的注释符号“/**/”和“//”。

行首加上“//”符号，整行即变为注释。

通过以下功能，可以对选择范围以行为单位进行注释或解除注释。

• [Edit（编辑）]⇨[Comment Out of Selected Range（选择范围的注释化）]（Ctrl+Shift+S）

• [Edit（编辑）]⇨[Disable Comment Out of Selected Range（选择范围的注释解除）]（Ctrl+Shift+D）

```
⊟IF  wValue0 < 0  THEN
⊟    bFlag0 := TRUE;  (* ON *)
 //  ELSE
 //  bFlag0 := FALSE; (* OFF *)
 END_IF;
```

以行为单位进行注释的范围

## 折叠显示

单击显示在注释左边的图标（ ⊟ ），可将注释折叠显示（(**)）。

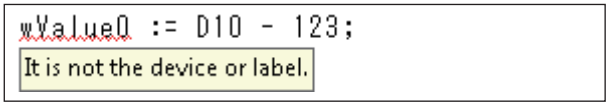
# 使用标签

登录标签并在ST程序中使用。

## 登录标签

从ST编辑器登录标签。

### 操作步骤

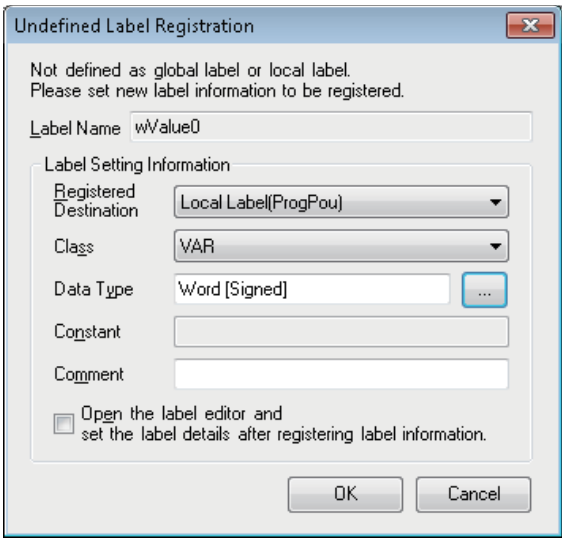


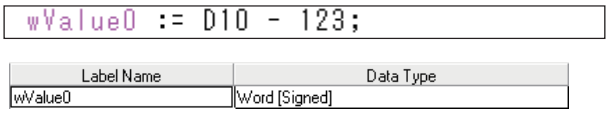

未登录的标签名将显示错误位置。

**1.** 光标移至标签名上的状态下按F2，可显示“Undefined Label Registration （未定义标签登录）”画面。

**2.** 按如下设置，登录标签wValue0。

- 登录目标标签： 局部标签
- 类： VAR
- 数据类型： 字[有符号]







登录的标签将以标签用的显示颜色显示。

**3.** 打开标签设置画面，登录的标签即被设置。

**要点**

使用了以下功能。

- [Edit （编辑）]⇨[Register Label （登录标签）] （F2）

ST编辑器上的各配置元素的显示颜色，可通过以下功能进行设置。

- [View （视图）]⇨[Color and Font （颜色及字体）]

## 通过标签编辑器设置

通过标签编辑器登录标签。

### 操作步骤

| Label Name | Data Type |
|---|---|
| wValue0 | Word [Signed] |
| bFlag0 | Bit |

1. 显示“Local Label Setting （局部标签设置）”画面。
2. 按如下设置，登录标签。
   - 标签名： bFlag0
   - 数据类型： 位
   - 类： VAR （自动设置。）
3. 打开ST编辑器后，登录的标签会以标签用显示颜色显示。

```
IF  wValue0 < 0  THEN
    bFlag0 := TRUE;  (* ON *)
    ELSE
    bFlag0 := FALSE; (* OFF *)
END_IF;
```

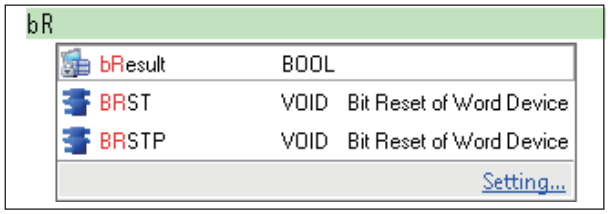
作为在程序例中使用的标签，对局部标签作如下设置。

| 标签名 | 数据类型 | 类 |
|---|---|---|
| wValue0 | 字[有符号] | VAR |
| wValue1 | 字[有符号] | VAR |
| bFlag0 | 位 | VAR |
| bResult | 位 | VAR |

## 在ST编辑器中使用已登录的标签

从已登录在标签编辑器的标签中选择，在ST编辑器中输入标签。

### 操作步骤





1. 事先登录位型标签bResult。
2. 输入起始字符“bR”后，会一览显示以该字符开始的已登录的标签名等。
3. 通过↓选择，按Enter后，选择的标签名即被输入。

# 创建函数、FB

新建在程序中使用的程序部件 （函数及FB）。

## 创建函数

在工程中新建函数的数据，对函数程序及其局部标签进行定义。

### ■新建数据

在工程中新建函数的数据。

#### 操作步骤

1. 选择[Project （工程）]⇨[Data Operation （数据操作）]⇨[New data （新建数据）]。
2. 在“New （新建数据）”画面中对以下设置项目进行设置。

| 数据类型 | 项目 | 内容 | 程序例中的设定值 |
|---|---|---|---|
| 函数 | 数据名 | 函数调用时的识别符。 | FunPou |
| | 程序语言 | 函数程序的表述语言。 | ST |
| | 返回值的类型 | 对执行完成后返回给调用原的返回值的数据类型进行设置。 | 位 |
| | 使用EN/ENO | 选择“Yes （是）”，参数会带上EN/ENO。<br>• EN：设置执行条件的BOOL型输入参数。<br>• ENO：返回执行结果的BOOL型输出参数。 | 否 |
| | 添加目标的FUN文件 | 设置用于存储创建的函数的文件名。 | FUNFILE |

**要点**

创建的函数的设置项目可以在“Properties （属性）”画面中确认。
通过以下操作，显示“Properties （属性）”画面。
• 在导航窗口中选择数据，右键单击，选择快捷菜单⇨[Properties （属性）]

### ■设置参数和内部变量

定义参数和在函数中使用的内部变量。
参数和内部变量通过函数的局部标签进行设置。应在标签设置画面中作如下设置。

| 标签名 | 数据类型 | 类 |
|---|---|---|
| i_wValue | 字[有符号] | VAR_INPUT |
| bFlag | 位 | VAR |

**要点**

类设置为“VAR_INPUT”的局部标签为输入参数。
类设置为“VAR_OUTPUT”的局部标签为输出参数。
函数调用时的参数顺序为局部标签设置时的定义顺序。

## 创建FB

在工程中新建FB的数据，对FB程序及其局部标签进行定义。

### ■新建数据

在工程中新建FB的数据。

#### 操作步骤

1. 选择[Project （工程）]⇨[Data Operation （数据操作）]⇨[New data （新建数据）]。
2. 在“New （新建数据）”画面中对以下设置项目进行设置。

| 数据类型 | 项目 | 内容 | 程序例中的设定值 |
|---|---|---|---|
| FB | 数据名 | 要定义的FB的数据类型名。 | FbPou |
| | 程序语言 | FB程序的表述语言。 | ST |
| | 使用EN/ENO | 选择“Yes （是）”，参数会带上EN/ENO。<br>• EN：设置执行条件的BOOL型输入参数。<br>• ENO：返回执行结果的BOOL型输出参数。 | 否 |
| | FB的类型 | 设置FB的程序本体的存储目标。可选择宏类型或子程序类型。 | 子程序类型 |
| | 添加目标的FB文件 | 设置用于存储创建的FB的文件名。 | FBFILE |

**要点**

创建的FB的设置项目可以在“Properties （属性）”画面中确认。
通过以下操作，显示“Properties （属性）”画面。
• 在导航窗口中选择数据，右键单击，选择快捷菜单⇨[Properties （属性）]

### ■设置参数和内部变量

定义参数和在FB中使用的内部变量。
参数和内部变量通过FB的局部标签进行设置。应在标签设置画面中作如下设置。

| 标签名 | 数据类型 | 类 |
|---|---|---|
| i_wValue | 字[有符号] | VAR_INPUT |
| o_wValue | 字[有符号] | VAR_OUTPUT |

**要点**

类设置为“VAR_INPUT”的局部标签为输入参数。
类设置为“VAR_OUTPUT”的局部标签为输出参数。

# 输入函数

输入函数。

**要点**

有返回值的函数可作为表达式处理。（函数调用表达式）
没有返回值的函数以调用语句表述。（函数调用语句）

```
FunPou(wValue0);  ←— 函数调用语句
```

## 将函数调用表达式输入赋值语句

### ■将返回值代入变量

将返回值代入变量时，函数调用写在赋值语句的右边。

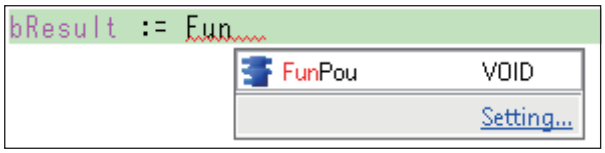
```
bResult :=
```

1. 输入要代入返回值的变量。
2. 输入赋值符号“:=”。

### ■输入函数名

从已定义的函数中选择并输入。

#### 操作步骤

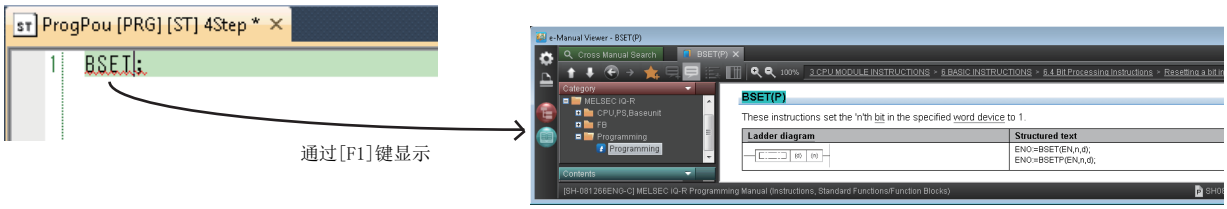



```
bResult := FunPou
```

1. 输入起始字符“Fun”后，会一览显示以该字符开始的函数。
2. 通过↓选择，按Enter后，选择的函数名即被输入。

**要点**

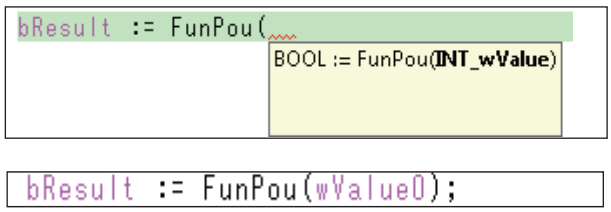
指令也可从一览显示输入。（☞ 39页 指令表述）
光标移至指令名上的状态下按F1，可在e-Manual Viewer中确认指令的详细内容。
（要确认指令，需要将相应编程手册的文件登录到e-Manual Viewer中。）



### ■输入参数

根据工具提示，输入参数。

#### 操作步骤




```
bResult := FunPou(wValue0);
```

1. 输入“(”后，会在工具提示中显示指令的格式。
2. 根据工具提示，输入参数。

有多个参数时，用逗号 （,）隔开。

3. 在参数的末尾输入“)”。
4. 输入表示调用语句结束的“;”。

**要点**

已定义的函数还可以输入使用了模板显示的参数。

光标移至函数名上时，通过Ctrl+F1可显示模板。

```
bResult := FunPou( ?INT_wValue? );
```

还可以使用参数选择功能。

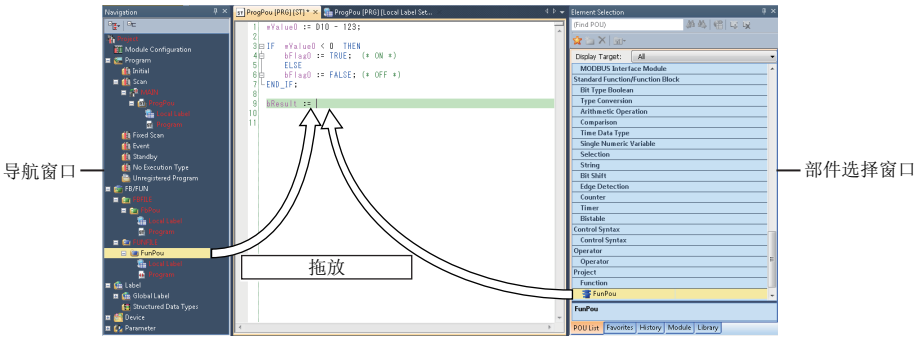
- [Edit （编辑）]⇨[Display Template （模板显示）] （Ctrl+F1）
- [Edit （编辑）] ⇨[Mark Template(Left) （模板参数选择(左)）]/[Mark Template(Right) （模板参数选择(右)）] （Ctrl+Alt+←/→）

## 从导航窗口、部件选择窗口中选择

可从导航窗口或部件选择窗口中选择，输入函数。

**要点**

可以从导航窗口或部件选择窗口拖放至ST编辑器。



# 输入FB

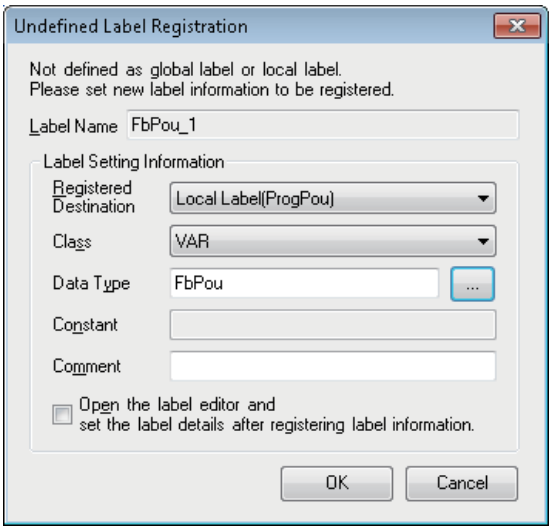
输入FB。

## 输入FB调用语句

### ■输入FB的实例名

输入已定义的FB。

#### 操作步骤



1. 输入实例名“FbPou_1”（任意）。
2. 光标移至实例名上的状态下按F2，可显示“Undefined Label Registration（未定义标签登录）”画面。
3. 进行如下设置，登录实例。
   - 登录目标标签： 局部标签
   - 类： VAR
   - 数据类型： FbPou

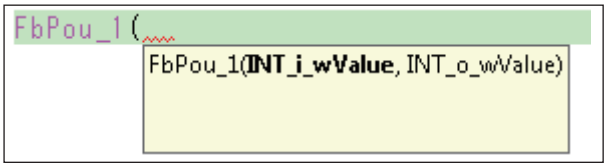
选择数据类型时，如果类型分类选择了“FB”，则可以选择已定义的FB。

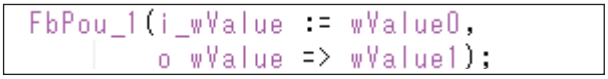
登录的实例名将以标签用的显示颜色显示。

### ■输入参数

根据工具提示，输入参数。

#### 操作步骤




1. 输入“(”后，会在工具提示中显示FB的格式。
2. 根据工具提示，输入参数。

有多个参数时，用逗号（,）隔开。

3. 在参数的末尾输入“)”。
4. 输入表示调用语句结束的“;”。

**要点**

已定义的FB还可以输入使用了模板显示的参数。
光标移至实例名上时，通过Ctrl+F1可显示模板。

```
FbPou_1(i_wValue:= ?INT? ,o_wValue=> ?INT? );
```

还可以使用参数选择功能。

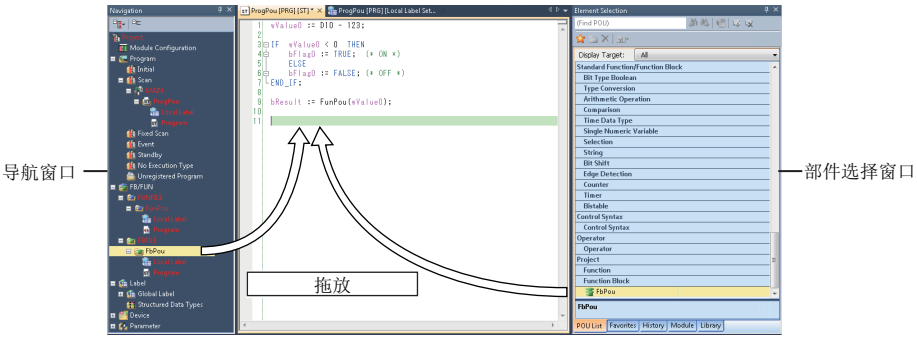
- [Edit（编辑）]⇨[Display Template（模板显示）]（Ctrl+F1）
- [Edit（编辑）]⇨[Mark Template(Left)（模板参数选择(左)）]/[Mark Template(Right)（模板参数选择(右)）]（Ctrl+Alt+←/→）

### 从导航窗口、部件选择窗口中选择

可从导航窗口或部件选择窗口中选择，输入函数FB。

**要点**

可以从导航窗口或部件选择窗口拖放至ST编辑器。




# 6.4 转换程序、调试

创建的程序必须转换成可在可编程控制器的CPU模块中执行的代码 （执行程序）。
如果程序中有不正确的表述，则会在转换时被检查出来。应根据显示的信息修改程序。

## 转换程序

将创建的程序转换成可执行代码

### 操作步骤

**1.** 执行[Convert （转换）]⇨[Convert （转换）] （ F4 ）。

**要点**

执行转换 （F4）时，仅对添加、更改的部分进行转换。
要对包括已转换的程序在内的所有程序进行转换时，应执行以下操作。
• [Convert （转换）]⇨[Rebuild All （全部转换）] （Shift+Alt+F4）

# 确认错误/警告

如果程序中有不正确的表述，则会在转换时被检查出来，并会显示错误/警告的信息。
例如：试着转换以下程序。

## 程序示例

```
1  (*wValue0 := D10 - 123;*)
2    wValue0 = D10 - 123;
3
4  IF  wValue0 < 0  THEN
5      bFlag0 := TRUE;  (* ON *)
6      ELSE
7      bFlag0 := FALSE; (* OFF *)
8  (*END_IF;*)
9  END_IF
10
11 (*bResult := FunPou( wValue0 );*)
12   FunPou( wValue0 );
13   FbPou_1(i_wValue := wValue0,
14     (*  o_wValue => wValue1);*)
15         o_wValue := wValue1);
```

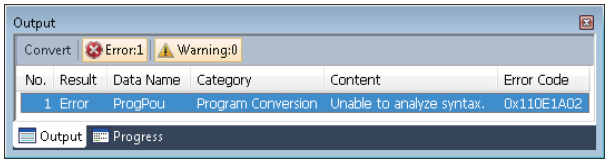
赋值符号不同

语句结尾没有“;”

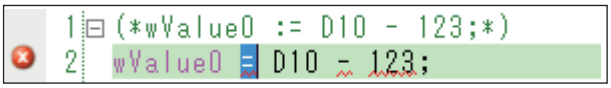
没有使用返回值

输出变量为“:=”

## 操作步骤




***1.*** 双击输出窗口中显示的错误/警告信息。




***2.*** 跳转到错误行。根据信息修改程序。

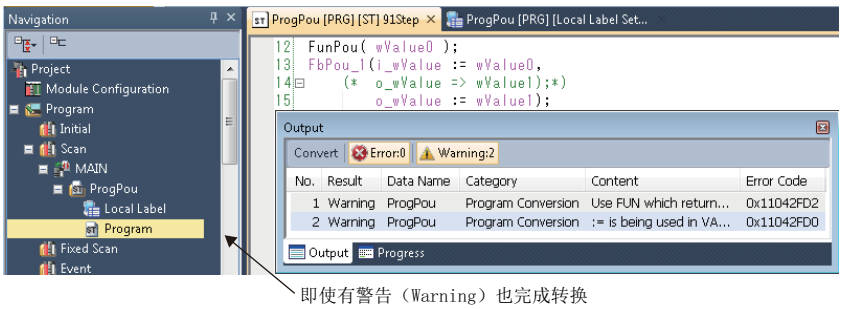
## 要点

如果无法对语法作出分析，应确认跳转行的前后语句。
错误处以下划线显示。

```
8  (*END_IF;*)
9  END_IF
10
11 (*bResult := FunPou( wValue0 );*)
12   FunPou( wValue0 );
```

语句结尾没有“;”

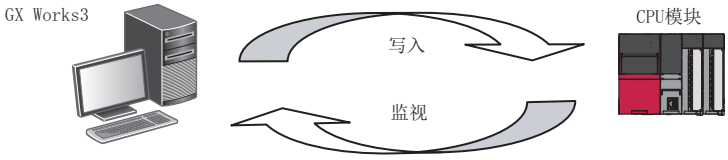
仅含警告（Warning）级别的信息时，会完成转换。




即使有警告（Warning）也完成转换

# 6.5 确认机器的实际动作

将转换后的执行程序写入到可编程控制器的CPU模块中后执行。监视执行中的程序，确认动作是否与预期一致。




## 在可编程控制器中执行程序

以下所示为在CPU模块中执行执行程序的步骤。

### 操作步骤

1. 将计算机连接至CPU模块，通过工程工具（GX Works3）设置连接目标。
2. 将CPU模块的动作状态置为STOP。
3. 选择[Online（在线）] ⇨[Write to PLC（写入至可编程控制器）]，写入“Program （程序）”。

- 应根据系统或软元件设置等的变更，写入“Parameter（参数）”。
- 使用了全局标签时，应写入“Global Label Setting（全局标签设置）”。
- 程序中使用了设置有初始值的标签/软元件时，应写入“Label Initial Value（标签初始值）”、“Global Label Initial Value（全局标签初始值）”或“Device Initial Value（软元件初始值）”。

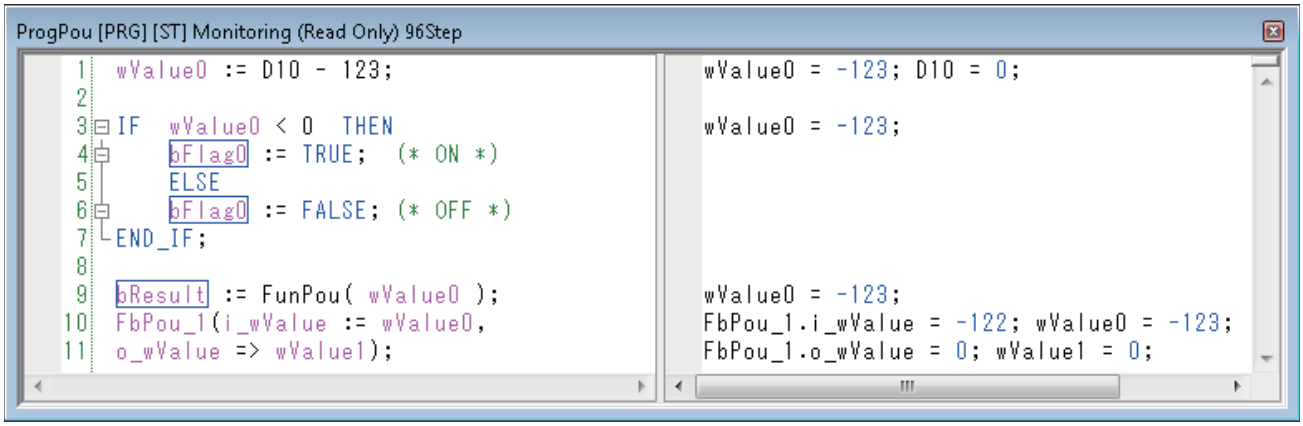
4. 复位CPU模块。
5. 将CPU模块的动作状态置为RUN。

## 确认执行中的程序

本项对在程序编辑器中确认执行中程序的状态的步骤进行说明。

### 操作步骤

选择[Online（在线）]⇨ [Monitor（监视）] ⇨[Start Monitoring（监视开始）]（F3）/[Stop Monitoring（监视停止）]（Alt+F3）。



## 当前值的显示

以下所示为在ST编辑器上显示的监视值。

### ■位类型

位类型的监视值在程序中显示如下。

- TRUE: bFlag0
- FALSE:bFlag0

### ■位类型以外

位类型以外的监视值显示在分割窗口的右侧。

**要点**

将光标移至软元件/标签名上，工具提示中即显示监视值。

## 当前值的更改

可通过以下方法更改软元件或标签的当前值。

### 操作步骤

1. 在ST编辑器中选择要更改当前值的软元件/标签。
2. 按Shift+Enter。（也可通过Shift+双击执行。）

位类型以外时，登录至监看窗口。应在监看窗口中更改当前值。

### ■监看窗口中的当前值更改

可通过以下方法确认、更改登录在监看窗口中的软元件/标签的当前值。

### 操作步骤

1. 选择[Online （在线）]⇨ [Watch （监看）] ⇨[Start Watching （监看开始）] （Shift+F3）/[Stop Watching （监看停止）] （Shift+Alt+F3）。
2. 监看中，在“Current Value （当前值）”栏中直接输入要更改的值。

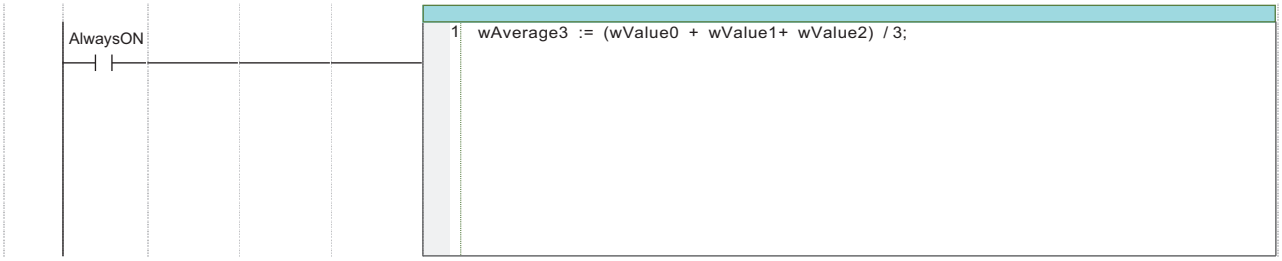
**要点**

以下任一操作也可以将软元件/标签登录至监看窗口。

- 从ST编辑器拖放到监看窗口
- [Tool （工具）] ⇨[Options （选项）]⇨“Monitor （监视）”⇨“ST Editor （ST编辑器）”⇨“Setting for Automatic Registration to Watch Window （自动登录至监看窗口的设置）”

# 6.6 使梯形图的一部分成为ST （内嵌ST）

要使梯形图程序的一部分成为ST时，可使用内嵌ST功能。使用了内嵌ST，即可取代梯形图的指令用ST程序进行表述。




## 操作步骤

**1.** 选择[Edit （编辑）]⇨[Inline Structured Text （内嵌ST）]⇨[Insert Inline Structured Text Box （插入内嵌ST框）] （Ctrl+B）。

**2.** 在内嵌ST框中编辑ST程序。

内嵌ST框中的编辑方法与在ST编辑器中的编辑方法相同。

**3.** 转换梯形图程序。

内嵌ST可作为梯形图程序的一部分进行编译 （转换）。

**要点**

在梯形图输入对话框中输入“STB”，可以插入内嵌ST框。

# 第2部分 程序示例

本部分记载了以单纯功能为模型的ST程序示例。

# 7 程序示例的概要

以下所示为程序示例的一览和使用方法。

## 7.1 程序示例一览

第2部分以单纯功能的应用程序为模型，记载了以下程序示例。

| 项目 | 示例内容 | | 参照 |
|---|---|---|---|
| 计算器处理 | 四则运算<br>条件分支 | 选择语句 （IF语句）<br>整数、实数及BOOL值的代入 | 61页 初始化程序： Initialization |
| | | 选择语句 （CASE语句）<br>四则运算 | 62页 四则运算 （FUN）：Calculation |
| | | 整数的舍去/进位/四舍五入 | 63页 化整处理 （FUN）：Rounding |
| | | 整数、实数的运算<br>整数、实数的类型转换<br>选择语句的分层 | 64页 尾数处理 （FUN）:FractionProcessing<br>65页 计算器程序： Calculator<br>67页 含税计算程序： IncludingTax |
| 定位处理 | 指数函数<br>三角函数<br>结构体 | 三角函数 （$TAN^{-1}$） | 69页 根据X、Y坐标计算旋转角度(FUN): GetAngle |
| | | 结构体型参数<br>结构体的运算<br>指数函数 （幂乘、平方根） | 70页 计算X、Y坐标的2点间距离 （FUN）：GetDistance |
| | | 三角函数 （COS、SIN）<br>结构体型返回值 | 71页 根据半径和角度计算X、Y坐标 （FUN）：GetXY |
| | | 实数、常数的运算 | 72页 电机的指令脉冲数的计算 （FB）：<br>PulseNumberCalculation |
| | | 结构体的代入<br>结构体的参数指定 | 73页 X、Y坐标的位置控制程序： PositionControl |
| 次品分类 | 数组<br>循环处理 | 二维数组<br>数组的运算<br>数组型参数<br>循环语句和选择语句的分层 | 76页 产品检查 （FB）：ProductCheck |
| | | 包含数组型成员的结构体<br>结构体数组 | 77页 产品数据的分类 （FB）：Assortment |
| | | 数组元素的初始化<br>数组的参数指定<br>循环语句 （FOR语句、WHILE语句、REPEAT语句） | 78页 产品数据管理程序： DataManagement |
| 运转时间计测 | 时间<br>字符串 | 时间型数据<br>定时器的触点、线圈、当前值 | 81页 运转时间管理程序： OperatingTime |
| | | 定时器<br>BOOL值的运算 | 82页 闪烁定时器 （FB）：FlickerTimer<br>83页 指示灯的点亮/熄灭程序： LampOnOff |
| | | 时钟数据 （INT型数组数据） | 84页 从秒至时、分、秒的转换程序： SecondsToTimeArray |
| | | 时间型数据<br>字符串数据 | 85页 从时间型至字符串的转换程序： TimeToString |

# 7.2 通过GX Works3创建程序示例时

本手册记载的是通过GX Works3创建的ST程序示例。

## 注意事项

本手册中记载的程序示例并不保证其实际的动作。
在实际创建程序时，要对应实际系统及要求的运行。
应根据需要，对应所使用的机器，对程序示例中使用的标签分配软元件。
应结合所使用的机器及软元件设置参数。

## 创建步骤

执行程序示例时，按如下步骤创建程序。

**1.** 在标签编辑器中设置全局标签。（☞ 46页 通过标签编辑器设置）

• 使用结构体时，应新建结构体定义并进行设置。

**2.** 创建程序部件。

• 设置局部标签。（☞ 46页 通过标签编辑器设置）

• 在程序本体中输入程序示例。（☞ 43页 编辑器间的复制）

程序内部使用了本手册中定义的程序部件时，应在同一工程内创建相应的函数、FB。（☞ 47页 新建数据）

**3.** 转换程序。（☞ 52页 转换程序、调试）

**要点**

以PDF或e-Manual格式使用本手册时，可以通过复制/粘贴的方式方便地引用程序示例。

# 8 计算器处理 （四则运算和条件分支）

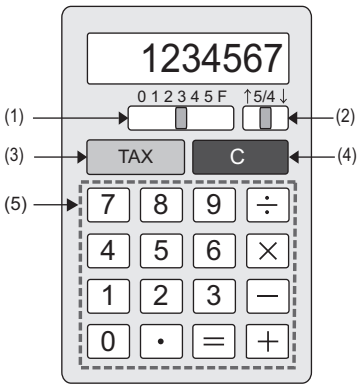
以下所示为在实现市售计算器功能的程序中进行四则运算或条件分支处理的示例。
在本程序示例中创建以下程序部件。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| Initialization | 程序块 | 有清除指示时，对变量进行初始化。 | 61页 初始化程序： Initialization |
| Calculation | 函数 | 对2值进行加减乘除运算。 | 62页 四则运算 （FUN）：Calculation |
| Rounding | 函数 | 对整数值进行舍去/进位/四舍五入。 | 63页 化整处理 （FUN）：Rounding |
| FractionProcessing | 函数 | 在指定小数位的下1位进行尾数处理。 | 64页 尾数处理 （FUN）:FractionProcessing |
| Calculator | 程序块 | 根据输入，对值继续运算。 | 65页 计算器程序： Calculator |
| IncludingTax | 程序块 | 有含税计算指示时，计算出含税价格和税额，显示含税价格。 | 67页 含税计算程序： IncludingTax |

## 功能的概要

执行以下处理。

- 输入了清除键时，将对当前值 （到上次为止的计算结果）、显示值等进行初始化。
- 根据数字键的输入，更新显示值。小数点以下的值，根据滑动开关的设置进行尾数处理。
- 输入了含税计算键时，显示含税价格。

| 设备示意图 | 编号 | 名称 | | 内容 |
|---|---|---|---|---|
|  | (1) | 小数位指定开关 | 0～5 | 指定要显示的小数点以下的位数。<br>指定位数以下的值将按化整处理开关指定的方法进行处理。 |
| | | | 浮点数 （F） | 不进行尾数处理直接显示。 |
| | (2) | 化整处理开关 | 进位 （↑） | 指定位的下1位为0以外时，指定位+1。 |
| | | | 四舍五入 （5/4） | 按指定位的下1位的值进行四舍五入。 |
| | | | 舍去 （↓） | 舍去指定位以后的值。 |
| | (3) | 含税计算键 | | 计算含税价格。 |
| | (4) | 清除键 | | 对当前值 （到上次为止的计算结果）、显示值进行初始化。 |
| | (5) | 数字键 | 0～9、小数点 （.） | 输入数值。 |
| | | | +、-、*、/ | 指定四则运算类型。 |
| | | | = | 执行指定的四则运算。 |

## 使用的全局标签

以下所示为本程序示例中使用的全局标签及结构体定义。
在GX Works3中作如下设置。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_eDisplayValue | 单精度实数 | VAR_GLOBAL | — | 显示值 |
| G_eLastValue | 单精度实数 | VAR_GLOBAL | — | 当前值 （到上次为止的计算结果） |
| G_wSwitch1 | 字[有符号] | VAR_GLOBAL | — | 开关的设置值 （0～5： 小数位数/6： 浮点数） |
| G_wSwitch2 | 字[有符号] | VAR_GLOBAL | — | 开关的设置值 （0： 舍去/1： 进位/2： 四舍五入） |
| G_bTax | 位 | VAR_GLOBAL | — | 输入含税计算键 |
| G_bClear | 位 | VAR_GLOBAL | — | 输入清除键 |
| G_wTenKey | 字[有符号] | VAR_GLOBAL | — | 输入数字键 （0～9： 数值/10： 小数点/11～14： 四则运算/15： =） |
| G_wOperation | 字[有符号] | VAR_GLOBAL | — | 运算类型 |
| G_eDecimal | 单精度实数 | VAR_GLOBAL | 初始值：0 | 小数运算用 |

### ■结构体

不使用。

# 8.1 初始化程序： Initialization

有清除指示时，对变量进行初始化。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | Initialization | ST | 初始化程序 |

## 程序示例

```
(*有清除指示时，对变量进行初始化。 *)
IF G_bClear THEN
    G_eDisplayValue := 0.0;
    G_eLastValue := 0.0;
    G_wOperation := 0;
    G_eDecimal := 0.0;
    G_bClear := FALSE;
END_IF;
```

**要点**

首次执行的各变量的值依标签设置或分配的软元件的设置而定。

全局标签可以在工程内所有程序中使用。

IF语句在条件式为真 （TRUE）时执行语句，为假 （FALSE）时不执行。

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_bClear | 位 | VAR_GLOBAL | — | 输入清除键 |
| G_eDisplayValue | 单精度实数 | VAR_GLOBAL | — | 显示值 |
| G_eLastValue | 单精度实数 | VAR_GLOBAL | — | 当前值 （到上次为止的计算结果） |
| G_wOperation | 字[有符号] | VAR_GLOBAL | — | 运算类型 |
| G_eDecimal | 单精度实数 | VAR_GLOBAL | 初始值：0 | 小数运算用 |

### ■局部标签

不使用。

## 程序部件

不使用。

# 8.2 四则运算（FUN）：Calculation

根据指定的运算类型，对2值进行四则运算。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 返回值的类型 | EN/ENO | 标题 |
|---|---|---|---|---|---|
| 函数 | Calculation | ST | 单精度实数 | 否 | 四则运算 |

## 程序示例

```
(*在值1上加减乘除值2。 *)
CASE i_wOperation OF
    1: (*运算类型为加法运算时： 计算结果=值1+值2*)
    Calculation := i_eValue1 + i_eValue2;
    2: (*减法运算时*)
    Calculation := i_eValue1 - i_eValue2;
    3: (*乘法运算时*)
    Calculation := i_eValue1 * i_eValue2;
    4: (*除法运算时*)
    IF i_eValue2 = 0.0 THEN(*值2为0时，不执行运行。*)
       Calculation := i_eValue1;
       ELSE
       Calculation := i_eValue1 / i_eValue2;
    END_IF;
END_CASE;
```

**要点**

CASE语句根据整数值的条件，选择执行语句。条件不一致的语句，不会执行。

选择语句（IF语句、CASE语句）可以分层。

## 变量

在GX Works3中作如下设置，定义标签。返回值的类型可在函数的属性中设置。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_eValue1 | 单精度实数 | VAR_INPUT | — | 值1 |
| i_wOperation | 字[有符号] | VAR_INPUT | — | 运算类型 |
| i_eValue2 | 单精度实数 | VAR_INPUT | — | 值2 |

### ■返回值的类型

| 识别符 | 数据类型 | 内容 |
|---|---|---|
| Calculation | 单精度实数 | 运算结果 |

## 程序部件

不使用。

# 8.3 化整处理 （FUN）：Rounding

对整数值的1的位进行舍去/进位/四舍五入。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 返回值的类型 | EN/ENO | 标题 |
|---|---|---|---|---|---|
| 函数 | Rounding | ST | 双字[有符号] | 否 | 化整处理 |

## 程序示例

```
(*按指定的化整处理方法，对1的位进行处理。 *)
CASE i_wType OF
    0: (* 舍去 *)
    Rounding := i_dValue / 10 * 10; (* 1的位舍去 *)
    1: (* 进位 *)
    Rounding := (i_dValue + 9) / 10 * 10; (* 1的位进位 *)
    2: (* 四舍五入 *)
    Rounding := (i_dValue + 5) / 10 * 10; (* 1的位四舍五入 *);
    ELSE (* 指定值以外时返回输入值。 *)
    Rounding := i_dValue;
END_CASE;
```

**要点**

整数的除法运算可以舍去小数点以下的位。

## 变量

在GX Works3中作如下设置，定义标签。返回值的类型可在函数的属性中设置。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_wType | 字[有符号] | VAR_INPUT | — | 处理方法 （0:舍去/1:进位/2:四舍五入） |
| i_dValue | 双字[有符号] | VAR_INPUT | — | 输入值 |

### ■返回值的类型

| 识别符 | 数据类型 | 内容 |
|---|---|---|
| Rounding | 双字[有符号] | 运算结果 |

## 程序部件

不使用。

# 8.4 尾数处理 （FUN）:FractionProcessing

在指定小数位的下1位进行尾数处理。
在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 返回值的类型 | EN/ENO | 标题 |
|---|---|---|---|---|---|
| 函数 | FractionProcessing | ST | 单精度实数 | 否 | 尾数处理 |

## 程序示例

```
(*在指定小数位的下1位进行尾数处理。 *)
(*位指定超出范围时，直接返回输入值。*)
IF (i_wDigits <= 0) OR (i_wDigits > c_wMAX) THEN
    FractionProcessing := i_eValue;
    RETURN; (* 结束处理 *)
END_IF;
(*移动小数点，使指定位变为1的位。 *)
wDigits := i_wDigits + 1;
eValue := i_eValue * (10.0 ** wDigits) - 0.5;
(*对整数的1的位进行指定的化整处理。 *)
dValue := Rounding(i_wType, REAL_TO_DINT(eValue));
(*将值转换为实数，恢复移动的小数点。 *)
FractionProcessing := DINT_TO_REAL(dValue)/(10.0 ** wDigits);
```

**要点**

将实数型转换为整数型时，使用REAL_TO_DINT等的类型转换函数。
运算表达式的项和参数可以使用类型转换函数的调用表达式。

## 变量

在GX Works3中作如下设置，定义标签。返回值的类型可在函数的属性中设置。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_eValue | 单精度实数 | VAR_INPUT | — | 输入值 |
| i_wDigits | 字[有符号] | VAR_INPUT | — | 显示的小数位 （小数部分0～5位） |
| i_wType | 字[有符号] | VAR_INPUT | — | 处理方法 （0:舍去/1:进位/2:四舍五入） |
| c_wMAX | 字[有符号] | VAR_CONSTANT | 常数： 5 | 小数位指定最大值 |
| wDigits | 字[有符号] | VAR | — | 处理对象的小数位 （小数部分1～6位） |
| eValue | 单精度实数 | VAR | — | 实数值 （内部计算用） |
| dValue | 双字[有符号] | VAR | — | 整数值 （内部计算用） |

### ■返回值的类型

| 识别符 | 数据类型 | 内容 |
|---|---|---|
| FractionProcessing | 单精度实数 | 运算结果 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| Rounding | 函数 | 对整数值进行舍去/进位/四舍五入。 | 63页 化整处理 （FUN）：Rounding |
| REAL_TO_DINT | 通用函数 | REAL型→DINT型转换<br>将REAL型数据转换成DINT型数据。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |
| DINT_TO_REAL | 通用函数 | DINT型→REAL型转换<br>将DINT型数据转换成REAL型数据。 | |

# 8.5 计算器程序：Calculator

根据数字键的输入，更新显示值。小数点以下的值，根据滑动开关的设置进行尾数处理。
在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | Calculator | ST | 计算器程序 |

## 程序示例

```
(*输入了数字键后，根据输入键区别处理。*)
IF G_wTenKey <> c_wNONE THEN
    CASE G_wTenKey OF
        0..9 : (* 输入数字键(0～9)时 *)
        (*在显示值的最后添加输入数值*)
        IF G_eDecimal = 0.0 THEN
            (* 整数部分时 *)
            G_eDisplayValue := (G_eDisplayValue * 10) + G_wTenKey;
            ELSE
            (* 小数点以下时 *)
            G_eDisplayValue := G_eDisplayValue + (G_eDecimal * G_wTenKey);
            G_eDecimal := G_eDecimal * 0.1;
        END_IF;
        10 : (* 输入小数点键时 *)
        G_eDecimal := 0.1;
        11..14 : (* 输入加减乘除键(11～14)时 *)
        (* 保存运算类型 *)
        G_wOperation := G_wTenKey - 10;
        (* 将显示值移动到上次运算值，复位显示值 *)
        G_eLastValue := G_eDisplayValue;
        G_eDisplayValue := 0.0;
        G_eDecimal := 0.0;
        15: (* 输入等号键时 *)
        (* 在当前值上加减乘除显示值。 *)
        G_eLastValue := Calculation(G_eLastValue, G_wOperation, G_eDisplayValue);
        (* 将尾数处理的结果代入显示值。 *)
        G_eDisplayValue := FractionProcessing(G_eLastValue, G_wSwitch1, G_wSwitch2);
        G_wOperation := 0; (* 清除运算类型 *)
        G_eDecimal := 0.0;
    END_CASE;
    (* 清除键输入 *)
    G_wTenKey := c_wNONE;
END_IF;
```

**要点**

运算符的左边和右边的数据类型不同时，运算结果按数据容量较大一侧的数据类型处理。（INT型 （字[有符号]）和REAL型 （单精度实数）的运算结果为REAL型 （单精度实数）。）

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_wTenKey | 字[有符号] | VAR_GLOBAL | — | 输入数字键 （0～9： 数值/10： 小数点/11～14： 四则运算/15： =） |
| G_eDecimal | 单精度实数 | VAR_GLOBAL | 初始值：0 | 小数运算用 |
| G_wOperation | 字[有符号] | VAR_GLOBAL | — | 运算类型 |
| G_eDisplayValue | 单精度实数 | VAR_GLOBAL | — | 显示值 |
| G_eLastValue | 单精度实数 | VAR_GLOBAL | — | 当前值 （到上次为止的计算结果） |
| G_wSwitch1 | 字[有符号] | VAR_GLOBAL | — | 开关的设置值 （0～5： 小数位数/6： 浮点数） |
| G_wSwitch2 | 字[有符号] | VAR_GLOBAL | — | 开关的设置值 （0：舍去/1：进位/2：四舍五入） |

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| c_wNONE | 字[有符号] | VAR_CONSTANT | 常数：-1 | 无数字键输入 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| Calculation | 函数 | 在当前值上加减乘除输入值。 | 62页 四则运算 （FUN）：Calculation |
| FractionProcessing | 函数 | 按指定的小数位数选择舍去/进位/四舍五入，求出值。 | 64页 尾数处理 （FUN）:FractionProcessing |

# 8.6 含税计算程序： IncludingTax

有含税计算指示时，计算出含税价格和税额，显示含税价格。
在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | IncludingTax | ST | 含税计算程序 |

## 程序示例

```
(*有含税计算指示时，显示含税价格。 *)
IF G_bTax THEN
    (* 计算税额。(同时计算小数点以下部分) *)
    eTaxAmount := G_eDisplayValue * c_eTaxRate / 100.0;
    (* 计算含税价格。(同时计算小数点以下部分) *)
    G_eLastValue := G_eDisplayValue + eTaxAmount;
    (* 将含税价格设为显示值。(小数点以下部分四舍五入) *)
    dPrice := REAL_TO_DINT(G_eLastValue);
    G_eDisplayValue := DINT_TO_REAL(dPrice);
    (* 清除键输入 *)
    G_bTax := FALSE;
END_IF;
```

**要点**

将实数型转换为整数型时，使用REAL_TO_DINT等的类型转换函数。
通过REAL_TO_DINT转换后的数据为将REAL型数据值的小数点以下第1位进行四舍五入后的值。

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_eDisplayValue | 单精度实数 | VAR_GLOBAL | — | 显示值 |
| G_eLastValue | 单精度实数 | VAR_GLOBAL | — | 当前值 （到上次为止的计算结果） |
| G_bTax | 位 | VAR_GLOBAL | — | 输入含税计算键 |

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| c_eTaxRate | 单精度实数 | VAR_CONSTANT | 常数： 8.0 | 税率 （%） |
| eTaxAmount | 单精度实数 | VAR | — | 税额 |
| dPrice | 双字[有符号] | VAR | — | 含税价格 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| REAL_TO_DINT | 通用函数 | REAL型→DINT型转换<br>将REAL型数据转换成DINT型数据。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |
| DINT_TO_REAL | 通用函数 | DINT型→REAL型转换<br>将DINT型数据转换成REAL型数据。 | |

# 9 定位处理 （指数函数、三角函数和结构体）

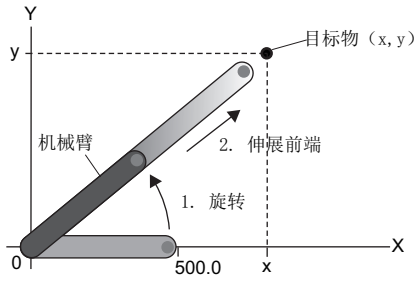
以下所示为在计算X、Y坐标上的位置或设备的旋转量的程序中，进行了指数函数、三角函数的算术运算及使用了结构体的数据处理的示例。

在本程序示例中创建以下程序部件。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| GetAngle | 函数 | 针对目标的X、Y坐标，计算旋转角度。 | 69页 根据X、Y坐标计算旋转角度(FUN)：GetAngle |
| GetDistance | 函数 | 根据X、Y坐标，计算2点间的距离。 | 70页 计算X、Y坐标的2点间距离 （FUN）：GetDistance |
| GetXY | 函数 | 根据半径和角度，计算X、Y坐标。 | 71页 根据半径和角度计算X、Y坐标 （FUN）：GetXY |
| PulseNumberCalculation | FB | 根据移动量计算发向电机的指令脉冲数。 | 72页 电机的指令脉冲数的计算 （FB）：PulseNumberCalculation |
| PositionControl | 程序块 | 根据想要移动的目标X、Y坐标，计算机械臂的移动角度和长度。 | 73页 X、Y坐标的位置控制程序：PositionControl |

## 功能的概要

根据以下步骤计算出机械臂的旋转角度和机械臂的长度调整用电机的指令脉冲数，以便将机械臂的前端移动到目标的X、Y坐标。




1. 将机械臂转至X、Y坐标所指定的目标物。
2. 通过步进电机伸展机械臂前端直至目标物。

## 使用的全局标签

以下所示为本程序示例中使用的全局标签及结构体定义。

在GX Works3中作如下设置。

### ■全局标签

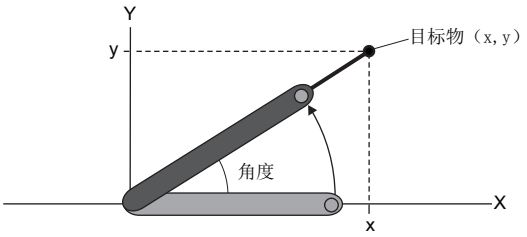
| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_stTarget | stPosition | VAR_GLOBAL | — | 目标物 （X、Y坐标） |
| G_stArm | stPosition | VAR_GLOBAL | — | 机械臂前端 （X、Y坐标） |
| G_eAngle | 单精度实数 | VAR_GLOBAL | — | 旋转角度 （度） |
| G_ePulses | 单精度实数 | VAR_GLOBAL | — | 电机的指令脉冲数 |
| G_bOneScanOnly | 位 | VAR_GLOBAL | 分配： SM402 | RUN后仅1次扫描ON |

### ■结构体

| 结构体名 | 标签名 | 数据类型 | 初始值 | 注释 |
|---|---|---|---|---|
| stPosition | eXcoordinate | 单精度实数 | 0.0 | X坐标 |
| | eYcoordinate | 单精度实数 | 0.0 | Y坐标 |

# 9.1 根据X、Y坐标计算旋转角度(FUN)：GetAngle

计算到目标X、Y坐标为止的旋转角度 （0～180度）。




角度（弧度）

$$\theta = \mathrm{ATAN}\left(\frac{y}{x}\right)$$

角度（度）

x = 0 时：　= 90

x > 0 时：　$= \theta \times \frac{180}{\pi}$

x < 0 时：　$= \theta \times \frac{180}{\pi} + 180$

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 返回值的类型 | EN/ENO | 标题 |
|---|---|---|---|---|---|
| 函数 | GetAngle | ST | 单精度实数 | 是 | 根据X、Y坐标计算旋转角度 |

## 程序示例

```
(* 根据X、Y坐标计算出弧度，并将弧度转换成角度。 *)
(* X坐标为0时，为90度 （不进行计算而结束处理）*)
IF i_eXcoordinate = 0.0 THEN
    GetAngle := 90.0;
    ENO := TRUE;
    RETURN; (* 结束处理 *)
END_IF;
(* 根据X、Y坐标计算出弧度。 *)
eAngleRad := ATAN(i_eYcoordinate / i_eXcoordinate); (* 角度 （弧度） θ = ATAN(Y坐标/X坐标) *)
(* 包含ATAN指令所无法处理的数据时，以错误结束。 *)
IF SD0 = H3402 THEN (* 错误代码： 3402H(运算异常) *)
    ENO := FALSE;
    RETURN; (* 结束处理 *)
    ELSE
    ENO := TRUE;
END_IF;
(* 将弧度转换为角度。 *)
GetAngle := eAngleRad * 180.0 / c_ePi; (* 角度 （度） = 角度 （弧度） θ ×180/π *)
(* X坐标为负值时+180度。 *)
IF i_eXcoordinate < 0.0 THEN
    GetAngle := GetAngle + 180.0;
END_IF;
```

**要点**

单精度实数弧度→角度转换可使用DEG指令。
SD0是错误检查用的软元件。存储最新错误代码。

## 变量

在GX Works3中作如下设置，定义标签。返回值的类型可在函数的属性中设置。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_eXcoordinate | 单精度实数 | VAR_INPUT | — | X坐标 |
| i_eYcoordinate | 单精度实数 | VAR_INPUT | — | Y坐标 |
| eAngleRad | 单精度实数 | VAR | — | 角度 （弧度） |
| c_ePi | 单精度实数 | VAR_CONSTANT | 常数：3.14159 | 圆周率 |

■返回值的类型

| 识别符 | 数据类型 | 内容 |
|---|---|---|
| GetAngle | 单精度实数 | 角度 （度）：0～180度 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| ATAN | 通用函数 | $TAN^{-1}$运算<br>输出输入值的$TAN^{-1}$<br>（反正切）值。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |

# 9.2 计算X、Y坐标的2点间距离 （FUN）：GetDistance

根据2点的X、Y坐标，计算2点间的距离。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 返回值的类型 | EN/ENO | 标题 |
|---|---|---|---|---|---|
| 函数 | GetDistance | ST | 单精度实数 | 否 | 2点间的距离计算 |

### 程序示例

```
(* 根据X、Y坐标计算2点间的距离。 *)
GetDistance := SQRT((i_stPosition0.eXcoordinate - i_stPosition1.eXcoordinate) ** 2.0
                  + (i_stPosition0.eYcoordinate - i_stPosition1.eYcoordinate) ** 2.0);
```

**要点**

函数/FB可以指定结构体型的参数。

可以对运算表达式的操作数 （运算对象的值）、赋值语句的左边和右边指定结构体成员。

“**”是幂乘的运算符。

## 变量

在GX Works3中作如下设置，定义标签。返回值的类型可在函数的属性中设置。

■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_stPosition0 | stPosition | VAR_INPUT | — | 位置0 （X、Y坐标） |
| i_stPosition1 | stPosition | VAR_INPUT | — | 位置1 （X、Y坐标） |

■返回值的类型

| 识别符 | 数据类型 | 内容 |
|---|---|---|
| GetDistance | 单精度实数 | 2点间的距离 |

■结构体

| 结构体名 | 标签名 | 数据类型 | 初始值 | 注释 |
|---|---|---|---|---|
| stPosition | eXcoordinate | 单精度实数 | 0.0 | X坐标 |
| | eYcoordinate | 单精度实数 | 0.0 | Y坐标 |

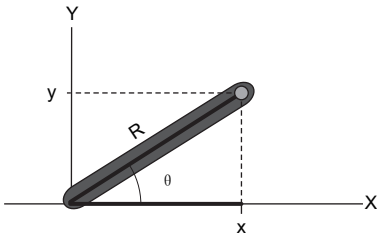
## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| SQRT | 通用函数 | 平方根<br>输出输入值的平方根。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |

# 9.3 根据半径和角度计算X、Y坐标 （FUN）：GetXY

根据指定的半径和角度，计算X、Y坐标。




角度（弧度）　　X、Y坐标

$\theta$ = 角度（度） × $\frac{\pi}{180}$

x = 半径 R × COS（θ）
y = 半径 R × SIN（θ）

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 返回值的类型 | EN/ENO | 标题 |
|---|---|---|---|---|---|
| 函数 | GetXY | ST | stPosition | 否 | X、Y坐标的计算 |

## 程序示例

```
(* 从度转换成弧度单位。 *)
eAngleRad := i_eAngle * c_ePi / 180.0; (* 角度 （弧度） θ= 角度 （度）×π/180 *)
(* 计算X、Y坐标。 *)
GetXY.eXcoordinate := i_eRadius * COS(eAngleRad); (* X坐标 = 半径 × COS(θ) *)
GetXY.eYcoordinate := i_eRadius * SIN(eAngleRad); (* Y坐标 = 半径 × SIN(θ) *)
```

**要点**

函数无法指定结构体型的返回值。
单精度实数角度→弧度转换可使用RAD指令。

## 变量

在GX Works3中作如下设置，定义标签。返回值的类型可在函数的属性中设置。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_eRadius | 单精度实数 | VAR_INPUT | — | 半径 （mm） |
| i_eAngle | 单精度实数 | VAR_INPUT | — | 角度 （度）：0～180度 |
| eAngleRad | 单精度实数 | VAR | — | 角度 （弧度） |
| c_ePi | 单精度实数 | VAR_CONSTANT | 常数：3.14159 | 圆周率 |

### ■返回值的类型

| 识别符 | 数据类型 | 内容 |
|---|---|---|
| GetXY | stPosition | X、Y坐标 |

### ■结构体

| 结构体名 | 标签名 | 数据类型 | 初始值 | 注释 |
|---|---|---|---|---|
| stPosition | eXcoordinate | 单精度实数 | 0.0 | X坐标 |
| | eYcoordinate | 单精度实数 | 0.0 | Y坐标 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| COS | 通用函数 | COS运算<br>输出输入值的COS （余弦）值。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |
| SIN | 通用函数 | SIN运算<br>输出输入值的SIN （正弦）值。 | |

# 9.4 电机的指令脉冲数的计算 （FB）：PulseNumberCalculation

根据移动量计算发向电机的指令脉冲数。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | EN/ENO | 标题 |
|---|---|---|---|---|
| FB | PulseNumberCalculation | ST | 否 | 指令脉冲数的计算 |

### 程序示例

```
(* 计算指令脉冲数。 *)
IF elround <> 0.0 THEN
    (* 指令脉冲数 = 电机分辨率 * ( 目标移动量 / 电机每转的移动量 ) *)
    o_ePulses := eResolution * ( i_eDistance / elround );
END_IF;
```

## 变量

在GX Works3中作如下设置，定义标签。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_eDistance | 单精度实数 | VAR_INPUT | — | 目标移动量 （mm） |
| o_ePulses | 单精度实数 | VAR_OUTPUT | — | 电机的指令脉冲数 （脉冲） |
| elround | 单精度实数 | VAR | — | 电机每转的移动量 （mm/rev） |
| eResolution | 单精度实数 | VAR | — | 电机分辨率 （脉冲/rev） |

**要点**

elround及eResolution在使用侧程序的局部标签设置中设置初始值。

☞74页 局部标签

使用MELSEC iQ-F系列时，由于无标签的初始值设置，应通过程序或工程工具的监看窗口等设置初始值。

## 程序部件

不使用。

# 9.5 X、Y坐标的位置控制程序： PositionControl



根据想要移动的目标X、Y坐标，计算移动的角度和长度，以控制机械臂。
在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | PositionControl | ST | X、Y坐标的位置控制程序 |

## 程序示例

```
(* 计算出机械臂的旋转角度和机械臂的长度调整用电机的指令脉冲数，以便将机械臂的前端移动到目标的X、Y坐标。 *)
(* RUN后仅在第1次扫描时为机械臂的坐标设置初始值。 *)
IF G_bOneScanOnly THEN
   G_stArm.eXcoordinate := 500.0;
   G_stArm.eYcoordinate := 0.0;
END_IF;
(* 目标物的位置由其他程序中的全局变量G_stTarget设置。[本程序示例中未记载]*)
(* 计算机械臂对目标X、Y坐标的移动角度。 *)
eTargetAngle := GetAngle( TRUE, bResult1, G_stTarget.eXcoordinate, G_stTarget.eYcoordinate );
eArmAngle := GetAngle( bResult1, bResult2, G_stArm.eXcoordinate, G_stArm.eYcoordinate );
IF bResult2 THEN
   G_eAngle := eTargetAngle - eArmAngle;
   ELSE
   RETURN; (* 错误结束 *)
END_IF;
(* 根据机械臂的前端X、Y坐标，计算当前机械臂的长度 （与原点之间的距离）。 *)
eDistance := GetDistance( G_stArm, stOrigin ); (* 输入变量为结构体型函数调用 *)
(* 计算旋转后的机械臂前端的X、Y坐标。 *)
G_stArm := GetXY( eDistance, G_eAngle); (* 返回值为结构体型函数调用 *)
(* 根据目标和机械臂前端的X、Y坐标，计算2点间的距离。 *)
eDistance := GetDistance( G_stTarget, G_stArm );
(* 根据移动量计算发向机械臂长度调整用电机的指令脉冲数。 *)
PulseNumberCalculation_1(i_eDistance := eDistance, o_ePulses => G_ePulses ); (* FB调用 *)
(* 指定角度，旋转机械臂。[本程序示例中未记载] *)
(* 指定电机的指令脉冲数，伸展机械臂前端。[本程序示例中未记载] *)
(* 更新机械臂坐标。 *)
G_stArm := G_stTarget;
```

**要点**

可选择EN/ENO的有无，创建函数或FB。
在EN中指定了FALSE时，不执行处理。ENO变为FALSE。
FB实例中，输出变量用“=>”指定。

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_bOneScanOnly | 位 | VAR_GLOBAL | 分配：SM402 | RUN后仅1次扫描ON |
| G_stTarget | stPosition | VAR_GLOBAL | — | 目标物（X、Y坐标） |
| G_stArm | stPosition | VAR_GLOBAL | — | 机械臂前端（X、Y坐标） |
| G_eAngle | 单精度实数 | VAR_GLOBAL | — | 旋转角度（度） |
| G_ePulses | 单精度实数 | VAR_GLOBAL | — | 电机的指令脉冲数（脉冲） |

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| eTargetAngle | 单精度实数 | VAR | — | 至目标物的角度（度）：0～180度 |
| eArmAngle | 单精度实数 | VAR | — | 机械臂的角度（度）：0～180度 |
| bResult1 | 位 | VAR | — | 运算结果1 |
| bResult2 | 位 | VAR | — | 运算结果2 |
| stOrigin | stPosition | VAR | — | 原点（X、Y坐标 = 0, 0） |
| eDistance | 单精度实数 | VAR | — | 距离 |
| PulseNumberCalculation_1 | PulseNumberCalculation | VAR | 初始值<br>• e1round: 0.2<br>• eResolution: 3000.0 | 指令脉冲数的计算处理 |

**要点**

使用MELSEC iQ-F系列时，由于无标签的初始值设置，应通过程序或工程工具的监看窗口等设置初始值。

### ■结构体

| 结构体名 | 标签名 | 数据类型 | 初始值 | 注释 |
|---|---|---|---|---|
| stPosition | eXcoordinate | 单精度实数 | 0.0 | X坐标 |
| | eYcoordinate | 单精度实数 | 0.0 | Y坐标 |

**要点**

在GX Works3中，可对每个FB实例的内部变量设置初始值。

应通过对结构体定义设置初始值、或通过程序初始化的方式，初始化结构体型变量。全局标签时，可对要分配的软元件设置初始值。

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| GetAngle | 函数 | 针对目标的X、Y坐标，计算角度。 | 69页 根据X、Y坐标计算旋转角度(FUN)：GetAngle |
| GetXY | 函数 | 根据半径和角度，计算X、Y坐标。 | 71页 根据半径和角度计算X、Y坐标（FUN）：GetXY |
| GetDistance | 函数 | 根据X、Y坐标，计算2点间的距离。 | 70页 计算X、Y坐标的2点间距离（FUN）：GetDistance |
| PulseNumberCalculation | FB | 根据移动量计算发向电机的指令脉冲数。 | 72页 电机的指令脉冲数的计算（FB）：PulseNumberCalculation |

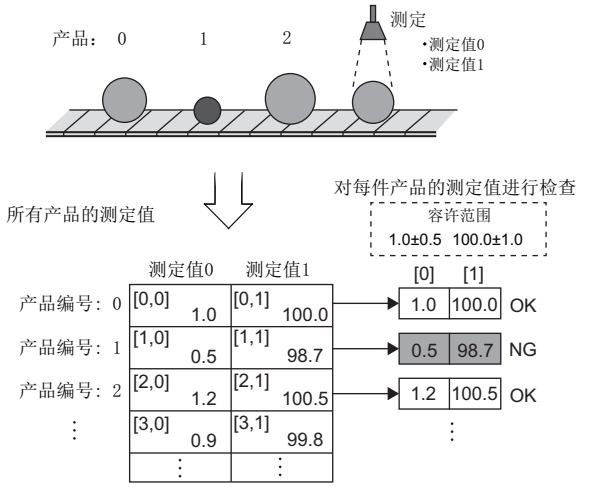
# 10 次品分类（数组和循环处理）

以下所示为在要整理多个产品的数据的程序中，使用数组循环处理的示例。
在本程序示例中创建以下程序部件。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| ProductCheck | FB | 根据产品数据，判断正品、次品。 | 76页 产品检查（FB）：ProductCheck |
| Assortment | FB | 将产品数据按正品、次品分类。 | 77页 产品数据的分类（FB）：Assortment |
| DataManagement | 程序块 | 整理产品数据。 | 78页 产品数据管理程序：DataManagement |

## 功能的概要

对测定了多个测定值（大小、重量等）的产品数据进行检查，划分正品、次品。

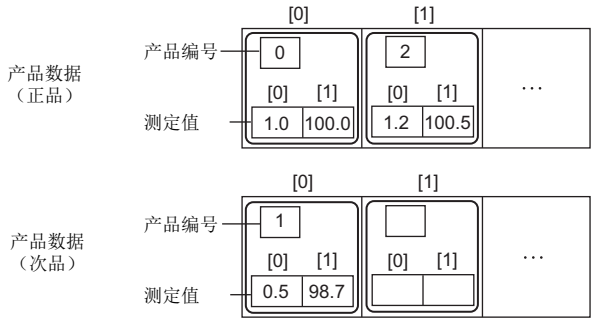



**1.** 测定产品。

在本程序示例中，将对8件产品的2种测定值进行测定，并对测得的数据进行处理。请在所有产品的测定值（G_eValueArray）中存储任意的值。

**2.** 对每件产品的测定值进行检查。

- 正品： 所有测定值均在容许范围内的产品
- 次品： 有测定值超出容许范围的产品




**3.** 将数据按正品、次品分开。

**4.** 完成所有产品的检查后，获取正品和次品的数据。

## 使用的全局标签

以下所示为本程序示例中使用的全局标签及结构体定义。
在GX Works3中作如下设置。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_eValueArray | 单精度实数（0..7,0..1） | VAR_GLOBAL | — | 所有产品的测定值（按产品编号顺序存储） |
| GC_wValueNumber | 字[有符号] | VAR_GLOBAL_CONSTANT | 常数：2 | 每件产品的测定值数 |
| GC_wTotalProduct | 字[有符号] | VAR_GLOBAL_CONSTANT | 常数：8 | 产品总数 |
| G_stProductArray | stProduct(0..7) | VAR_GLOBAL | — | 产品数据（正品） |
| G_stDefectiveArray | stProduct(0..7) | VAR_GLOBAL | — | 产品数据（次品） |

### ■结构体

| 结构体名 | 标签名 | 数据类型 | 初始值 | 注释 |
|---|---|---|---|---|
| stProduct | wProductNumber | 字[有符号] | — | 产品编号 |
| | eValueArray | 单精度实数（0..1） | — | 测定值 |

# 10.1 产品检查 （FB）：ProductCheck

判断值是否在容许范围内。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | EN/ENO | 标题 |
|---|---|---|---|---|
| FB | ProductCheck | ST | 否 | 产品检查 |

## 程序示例

```
(* 检查数组各元素的值是否在指定的容许范围内。 *)
FOR wIndex := 0 TO (i_wValueNumber - 1) BY 1 DO
    (* 根据基准值和容许差，求出上下限值。 *)
    eMaxValue := i_eAcceptableArray[wIndex, c_wBasicSize] + i_eAcceptableArray[wIndex, c_wTolerance];
    eMinValue := i_eAcceptableArray[wIndex, c_wBasicSize] - i_eAcceptableArray[wIndex, c_wTolerance];
    (* 检查上下限值。 *)
    IF (eMaxValue >= i_eValueArray[wIndex]) AND (eMinValue <= i_eValueArray[wIndex]) THEN
        o_bResult := TRUE;
        ELSE
        o_bResult := FALSE;
        EXIT; (* 如有值超出范围，则中断结束 *)
    END_IF;
END_FOR;
```

**要点**

将数组型数据和循环语句结合，可以对数组的多个元素执行同一数据处理。

数组的各元素可以作为定义了数据类型的变量由运算表达式进行指定。

选择语句 （IF语句、CASE语句）及循环语句 （FOR语句、WHILE语句、REPEAT语句）可以分层。

## 变量

在GX Works3中作如下设置，定义标签。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_eValueArray | 单精度实数 （0..1） | VAR_INPUT | — | 判断值 |
| i_eAcceptableArray | 单精度实数 （0..1,0..1） | VAR_INPUT | — | 容许范围 |
| i_wValueNumber | 字[有符号] | VAR_INPUT | — | 每件产品的测定值数 |
| o_bResult | 位 | VAR_OUTPUT | — | 检查结果 |
| c_wBasicSize | 字[有符号] | VAR_CONSTANT | 常数： 0 | 存储基准值的元素编号 |
| c_wTolerance | 字[有符号] | VAR_CONSTANT | 常数： 1 | 存储容许差的元素编号 |
| eMaxValue | 单精度实数 | VAR | — | 判断的上限值 |
| eMinValue | 单精度实数 | VAR | — | 判断的下限值 |
| wIndex | 字[有符号] | VAR | — | 元素编号 |

**要点**

FB内可使用全局数据。

可将数组型数据指定给输入变量。

## 程序部件

不使用。

# 10.2 产品数据的分类 （FB）：Assortment

将产品数据按正品、次品分类存储。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | EN/ENO | 标题 |
|---|---|---|---|---|
| FB | Assortment | ST | 否 | 产品数据的分类 |



## 程序示例

```
(* 根据检查结果划分正品、次品数据。*)
IF i_bCheck THEN
    (* 正品时 *)
    (* 在产品数据 （正品）中存储测定值 *)
    o_stProductArray[wTotal] := i_stProduct;
    (* 正品数+1 *)
    wTotal := wTotal + 1;
    ELSE
    (* 次品时 *)
    (* 在产品数据 （次品）中存储测定值 *)
    o_stDefectiveArray[wDefectiveTotal] := i_stProduct;
    (* 次品数+1 *)
    wDefectiveTotal := wDefectiveTotal + 1;
END_IF;
(* 更新成品率 *)
o_eYieldRatio := INT_TO_REAL(wTotal) / INT_TO_REAL(wTotal + wDefectiveTotal);
```

**要点**

ST程序中也可以使用成员中包含了数组的结构体或结构体型的数组。

## 变量

在GX Works3中作如下设置，定义标签。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| i_bCheck | 位 | VAR_INPUT | — | 检查结果 |
| i_stProduct | stProduct | VAR_INPUT | — | 检查完了产品数据 |
| o_stProductArray | stProduct(0..7) | VAR_OUTPUT | — | 产品数据 （正品） |
| o_stDefectiveArray | stProduct(0..7) | VAR_OUTPUT | — | 产品数据 （次品） |
| o_eYieldRatio | 单精度实数 | VAR_OUTPUT | — | 成品率 |
| wTotal | 字[有符号] | VAR | — | 正品数 |
| wDefectiveTotal | 字[有符号] | VAR | — | 次品数 |

### ■结构体

| 结构体名 | 标签名 | 数据类型 | 初始值 | 注释 |
|---|---|---|---|---|
| stProduct | wProductNumber | 字[有符号] | — | 产品编号 |
| | eValueArray | 单精度实数 （0..1） | — | 测定值 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| INT_TO_REAL | 通用函数 | INT型→REAL型转换<br>将INT型数据转换成REAL型数据。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |

# 10.3 产品数据管理程序： DataManagement

将产品按正品、次品分开，分别为每件产品存储测定值。
在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | DataManagement | ST | 产品数据管理程序 |

## 程序示例

```
(* 检查产品数据，划分正品和次品。 *)
(* 设置检查的容许范围。 *)
Check_1.i_eAcceptableArray[0,0] := 1.0;(* 测定值0的基准值 *)
Check_1.i_eAcceptableArray[0,1] := 0.5;(* 测定值0的容许差 *)
Check_1.i_eAcceptableArray[1,0] := 100.0; (* 测定值1的基准值 *)
Check_1.i_eAcceptableArray[1,1] := 1.0;(* 测定值1的容许差 *)
Check_1.i_wValueNumber := GC_wValueNumber; (* 每件产品的测定值数 *)
(* 通过循环处理检查所有产品的测定值。 *)
REPEAT
    (* 一次处理3件数据。 *)
    wDataEnd := wProductNumber + 3;
    (* 剩余数据不足3件时，处理到最后。*)
    IF wDataEnd > GC_wTotalProduct THEN
        wDataEnd := GC_wTotalProduct;
    END_IF;
    (* 重复是否为3件数据的判断、分类处理。 *)
    WHILE wProductNumber < wDataEnd DO
        (* 获取处理对象的测定值。 *)
        FOR wIndex := 0 TO (GC_wValueNumber - 1) BY 1 DO
            eValueArray[wIndex] := G_eValueArray[wProductNumber, wIndex];
        END_FOR;
        (* 根据测定值判断正品、次品。*)
        Check_1(i_eValueArray := eValueArray, o_bResult => bResult);
        (* 根据检查结果bResult划分正品、次品数据。*)
        stProductData.wProductNumber := wProductNumber;
        stProductData.eValueArray := eValueArray;
        Assortment_1(i_bCheck := bResult, i_stProduct := stProductData);
        (* 进行到下一个产品编号。 *)
        wProductNumber := wProductNumber + 1;
    END_WHILE;
    (* 以下情况时，结束处理。 *)
    UNTIL ((Assortment_1.o_eYieldRatio < c_eLimit) (* 成品率低于判断标准 *)
    OR(wProductNumber >= GC_wTotalProduct)) (* 或是产品编号超过产品总数 *)
END_REPEAT;
(* 获取按正品、次品划分的产品数据。 *)
G_stProductArray := Assortment_1.o_stProductArray;
G_stDefectiveArray := Assortment_1.o_stDefectiveArray;
```

**要点**

无法为数组各个元素设置不同的初始值。要设置不同值时，应通过程序进行设置。
FB时可在调用语句的前后指定参数。
选择语句（IF语句、CASE语句）及循环语句（FOR语句、WHILE语句、REPEAT语句）可以分层。

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 注释 |
|---|---|---|---|---|
| G_eValueArray | 单精度实数 （0..7,0..1） | VAR_GLOBAL | — | 所有产品的测定值 （按产品编号顺序存储） |
| GC_wValueNumber | 字[有符号] | VAR_GLOBAL_CONSTANT | 常数： 2 | 每件产品的测定值数 |
| GC_wTotalProduct | 字[有符号] | VAR_GLOBAL_CONSTANT | 常数： 8 | 产品总数 |
| G_stProductArray | stProduct(0..7) | VAR_GLOBAL | — | 产品数据 （正品） |
| G_stDefectiveArray | stProduct(0..7) | VAR_GLOBAL | — | 产品数据 （次品） |

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 注释 |
|---|---|---|---|---|
| wProductNumber | 字[有符号] | VAR | — | 产品编号 （内部循环处理用） |
| wDataEnd | 字[有符号] | VAR | — | 数据终端 （内部循环处理用） |
| wIndex | 字[有符号] | VAR | — | 元素编号 （内部循环处理用） |
| eValueArray | 单精度实数 （0..1） | VAR | — | 测定值 |
| bResult | 位 | VAR | — | 检查结果 （内部循环处理用） |
| stProductData | stProduct | VAR | — | 检查完了产品数据 |
| c_eLimit | 单精度实数 | VAR_CONSTANT | 常数： 0.8 | 成品容许率 |
| Check_1 | ProductCheck | VAR | — | 产品检查处理 |
| Assortment_1 | Assortment | VAR | — | 产品数据的分类处理 |

### ■结构体

| 结构体名 | 标签名 | 数据类型 | 初始值 | 注释 |
|---|---|---|---|---|
| stProduct | wProductNumber | 字[有符号] | — | 产品编号 |
| | eValueArray | 单精度实数 （0..1） | — | 测定值 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| ProductCheck | FB | 根据产品数据，判断正品、次品。 | 76页 产品检查 （FB）：ProductCheck |
| Assortment | FB | 将产品数据按正品、次品分类。 | 77页 产品数据的分类 （FB）：Assortment |

# 11 运转时间计测 （时间和字符串）

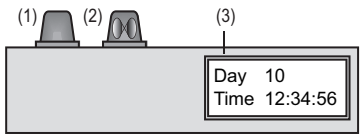
以下所示为在根据运转时间点亮指示灯以显示运转时间的设备的程序中，进行定时器及时间数据处理的示例。
在本程序示例中创建以下程序部件。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| OperatingTime | 程序块 | 以秒为单位对设备的运转时间进行计数。 | 81页 运转时间管理程序： OperatingTime |
| FlickerTimer | FB | 使输出信号交互闪烁。 | 82页 闪烁定时器 （FB）：FlickerTimer |
| LampOnOff | 程序块 | 根据设备的状态，点亮/熄灭指示灯。 | 83页 指示灯的点亮/熄灭程序： LampOnOff |
| SecondsToTimeArray | 程序块 | 根据以秒为单位的时间，求出时、分、秒。 | 84页 从秒至时、分、秒的转换程序：SecondsToTimeArray |
| TimeToString | 程序块 | 将运转时间 （时间型）转换为显示用字符串。 | 85页 从时间型至字符串的转换程序：TimeToString |

## 功能的概要

执行以下处理。

1. 以秒为单位对设备的运转时间进行计数。
2. 根据动作状态和运转时间，判断设备状态，点亮/熄灭指示灯。
3. 将运转时间转换为以时、分、秒为单位。
4. 将运转时间转换为画面显示用的字符串。

| 设备示意图 | 编号 | 名称 | 内容 | |
|---|---|---|---|---|
|  | (1) | 运转中指示灯 | 设备运转中点亮。<br>熄灭： 停止中，点亮： 运转中 | 运转时间超过3周以上时，交互闪烁。 |
| | (2) | 警告指示灯 | 运转时间超过1周以上时，点亮。<br>熄灭： 正常，点亮： 警告 | |
| | (3) | 显示画面 | 显示运转时间 （日、时、分、秒）。 | |

## 使用的全局标签

以下所示为本程序示例中使用的全局标签及结构体定义。
在GX Works3中作如下设置。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 内容 |
|---|---|---|---|---|
| G_bOperatingStatus | 位 | VAR_GLOBAL | — | 动作状态 （TRUE：运转中，FALSE：停止中） |
| G_tmTime | 时间 | VAR_GLOBAL | — | 总运转时间 （～T#24d20h31m23s647ms） |
| G_bOperationLamp | 位 | VAR_GLOBAL | — | 运转中指示灯 （TRUE：点亮，FALSE：熄灭） |
| G_bWarningLamp | 位 | VAR_GLOBAL | — | 警告指示灯 （TRUE：点亮，FALSE：熄灭） |
| G_dSeconds | 双字[有符号] | VAR_GLOBAL | — | 运转时间 （0～86399秒） |
| G_wTimeArray | 字[有符号](0..2) | VAR_GLOBAL | — | 运转时间 （[0]：时，[1]：分，[2]：秒） |
| G_sDisplayedCharacters | 字符串(32) | VAR_GLOBAL | — | 运转时间显示画面的显示字符 |
| G_bOneScanOnly | 位 | VAR_GLOBAL | 分配： SM402 | RUN后仅1次扫描ON |

### ■结构体

不使用。

# 11.1 运转时间管理程序： OperatingTime

以秒为单位对设备的运转时间进行计数。
在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | OperatingTime | ST | 运转时间管理程序 |



## 程序示例

```
(* 按秒对运转时间进行计数。 *)
bResult := OUT_T(G_bOperatingStatus, tdlsTimer, 10); (* 定时器的设定值1000ms *)
IF tdlsTimer.S THEN (* 1秒后 *)
    (* 运转时间 （TIMER型）计时 *)
    G_tmTime := G_tmTime + T#1000ms;
    IF G_tmTime < T#0ms THEN (* 发生溢出时 *)
        G_tmTime := T#0ms; (* 清零 *)
    END_IF;
    (* 运转时间 （以秒为单位）计时 *)
    G_dSeconds := G_dSeconds +1;
    IF G_dSeconds >= 86400 THEN
        G_dSeconds := 0; (* 第24小时清零 *)
    END_IF;
    (* 定时器的复位 （当前值：0，触点：OFF）*)
    RST(TRUE, tdlsTimer.N);
    RST(TRUE, tdlsTimer.S);
END_IF;
```

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 内容 |
|---|---|---|---|---|
| G_bOperatingStatus | 位 | VAR_GLOBAL | — | 运转状态 （TRUE：运转中，FALSE：停止中） |
| G_tmTime | 时间 | VAR_GLOBAL | — | 总运转时间 （～T#24d20h31m23s647ms） |
| G_dSeconds | 双字[有符号] | VAR_GLOBAL | — | 运转时间 （0～86399秒） |

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 内容 |
|---|---|---|---|---|
| bResult | 位 | VAR | — | 定时器执行的ENO |
| tdlsTimer | 定时器 | VAR | — | 1秒计测用定时器 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| OUT_T | 指令 | 低速定时器指令<br>到OUT指令为止的运算结果为ON时，线圈置ON，执行定时器的计测。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |

# 11.2 闪烁定时器 （FB）：FlickerTimer

输入信号为ON时，使输出信号0、1交互闪烁。

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | EN/ENO | 标题 |
|---|---|---|---|---|
| FB | FlickerTimer | ST | 是 | 闪烁定时器 |

## 程序示例

```
(* 使输出信号0、1交互闪烁。(触发器梯形图) *)
(* 求低速定时器的设定值。 *)
wInterval := TIME_TO_INT(i_tmInterval) / 100;
(* 定时器1为OFF时，定时器0在设置时间ms后置ON *)
bResult := OUT_T(NOT tdTimer1.S, tdTimer0, wInterval);
(* 定时器0从OFF变为ON时，定时器1在设置时间ms后置ON *)
bResult := OUT_T(tdTimer0.S, tdTimer1, wInterval);
(* 定时器0为ON时，输出信号0置ON *)
o_bOutputSignal0 := tdTimer0.S;
(* 定时器0为OFF时，输出信号1置ON *)
o_bOutputSignal1 := NOT tdTimer0.S;
```

**要点**

应将定时器的设定值设置为大于等于扫描时间+定时器的时限设置的值。定时器的时限设置因系列而异：

- MELSEC iQ-R：可在工程工具的参数设置中设置。（默认：100ms）
- MELSEC iQ-F：OUT_T指令将作为100ms的定时器运行，而OUTH指令为10ms、OUTHS指令为1ms。

## 变量

在GX Works3中作如下设置，定义标签。

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 内容 |
|---|---|---|---|---|
| i_tmInterval | 时间 | VAR_INPUT | — | 切换间隔 |
| o_bOutputSignal0 | 位 | VAR_OUTPUT | 初始值：TRUE | 输出信号0 |
| o_bOutputSignal1 | 位 | VAR_OUTPUT | 初始值：TRUE | 输出信号1 |
| bResult | 位 | VAR | — | 定时器执行的ENO |
| wInterval | 字[有符号] | VAR | — | 低速定时器设定值 |
| tdTimer0 | 定时器 | VAR | — | 定时器0 |
| tdTimer1 | 定时器 | VAR | — | 定时器1 |

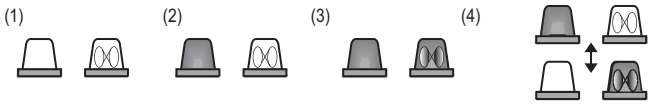
## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| OUT_T | 指令 | 低速定时器指令<br>到OUT指令为止的运算结果为ON时，线圈置ON，执行定时器的计测。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |
| TIME_TO_INT | 通用函数 | TIME型→INT型转换<br>将TIME型数据转换成INT型数据。 | |

# 11.3 指示灯的点亮/熄灭程序：LampOnOff

根据设备的状态，点亮/熄灭指示灯。




| 编号 | 运转中指示灯 | 警告指示灯 | 设备的状态 | | 备注 |
|---|---|---|---|---|---|
| (1) | 熄灭 | 熄灭 | 正常 | 停止中 | 运转前 |
| (2) | 点亮 | 熄灭 | | 运转中 | 运转时间0～7天 |
| (3) | 点亮 | 点亮 | 警告 | | 运转时间1周以上 |
| (4) | 交互闪烁 | | 异常 | | 运转时间3周以上 |

在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | LampOnOff | ST | 指示灯的点亮/熄灭程序 |

## 程序示例

```
(* 根据运转状态，对运转中指示灯进行ON/OFF。*)
G_bOperationLamp := G_bOperatingStatus;
(* 总运转时间为1周 （7天）以上时，警告指示灯ON。 *)
G_bWarningLamp := G_tmTime >= T#7d;
(* 总运转时间为3周 （21天）以上时， *)
(* 运转中指示灯和警告指示灯交互闪烁。 *)
FlickerTimer_1(EN := G_tmTime >= T#21d, i_tmInterval := T#1000ms);
IF FlickerTimer_1.ENO THEN
    G_bOperationLamp := FlickerTimer_1.o_bOutputSignal0;
    G_bWarningLamp := FlickerTimer_1.o_bOutputSignal1;
END_IF;
```

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 内容 |
|---|---|---|---|---|
| G_bOperatingStatus | 位 | VAR_GLOBAL | — | 动作状态 （TRUE：运转中，FALSE：停止中） |
| G_bOperationLamp | 位 | VAR_GLOBAL | — | 运转中指示灯 （TRUE：点亮，FALSE：熄灭） |
| G_bWarningLamp | 位 | VAR_GLOBAL | — | 警告指示灯 （TRUE：点亮，FALSE：熄灭） |
| G_tmTime | 时间 | VAR_GLOBAL | — | 总运转时间 （～T#24d20h31m23s647ms） |

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 内容 |
|---|---|---|---|---|
| FlickerTimer_1 | FlickerTimer | VAR | — | 闪烁定时器 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| FlickerTimer | FB | 使输出信号交互闪烁。 | 82页 闪烁定时器 （FB）：FlickerTimer |

# 11.4 从秒至时、分、秒的转换程序：SecondsToTimeArray

根据以秒为单位的时间，求出时、分、秒。时、分、秒的值存储在数组型数据（与时钟用指令SEC2TIME的参数的数据格式相同）中。


在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | SecondsToTimeArray | ST | 从秒至时、分、秒的转换程序 |

### 程序示例

```
(* 根据以秒为单位的时间，求出时、分、秒。 *)
G_wTimeArray[0] := GET_INT_ADDR( G_dSeconds / 3600); (* 小时 *)
G_wTimeArray[1] := GET_INT_ADDR((G_dSeconds MOD 3600) / 60); (* 分 *)
G_wTimeArray[2] := GET_INT_ADDR((G_dSeconds MOD 3600) MOD 60); (* 秒 *)
```

**要点**

MOD是余数运算的运算符。

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 内容 |
|---|---|---|---|---|
| G_dSeconds | 双字[有符号] | VAR_GLOBAL | — | 运转时间（0～86399秒） |
| G_wTimeArray | 字[有符号](0..2) | VAR_GLOBAL | — | 运转时间（[0]：时，[1]：分，[2]：秒） |

### ■局部标签

不使用。

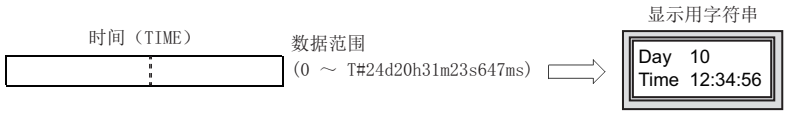
## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| GET_INT_ADDR | 通用函数 | 无需类型转换<br>输入变量按INT型输出。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |

# 11.5 从时间型至字符串的转换程序： TimeToString

将运转时间 （时间型）转换为显示用字符串。




在GX Works3中作如下设置，创建程序部件。

| 数据类型 | 数据名 | 程序语言 | 标题 |
|---|---|---|---|
| 程序块 | TimeToString | ST | 从时间型至字符串的转换程序 |

## 程序示例

```
(* 将运转时间转换为显示用字符串。*)
(* 初始化内部变量。*)
IF G_bOneScanOnly THEN (* RUN后仅在第1次扫描时执行*)
(* 初始化时间的单位 （日、时、分、秒的ms数）。 *)
   dUnitArray[0] := TIME_TO_DINT(T#1d); (* 日 *)
   dUnitArray[1] := TIME_TO_DINT(T#1h); (* 小时 *)
   dUnitArray[2] := TIME_TO_DINT(T#1m); (* 分 *)
   dUnitArray[3] := TIME_TO_DINT(T#1s); (* 秒 *)
(* 指定要转换为字符串的数值的数位。 *)
   wDigitArray[0] := 2; (* 按2位转换为字符串 *)
   wDigitArray[1] := 0; (* 无小数点 *)
END_IF;
sTimeString := ''; (* 初始化字符串 *)
(* 将时间 （TIME）型数据转换为ms数。 *)
dTime := TIME_TO_DINT(G_tmTime);
(* 按日、时、分、秒的顺序处理。 *)
FOR wIndex := 0 TO 3 BY 1 DO
(* 将时间 （ms单位）转换为日/时/分/秒。 *)
   wTime := DINT_TO_INT(dTime /dUnitArray[wIndex]);
   dTime := dTime MOD dUnitArray[wIndex];
(* 将日/时/分/秒的值转换为2位的字符串。 *)
   STR(TRUE, wDigitArray, wTime, sItemString);
   (* 设置要添加的字符串。 *)
   CASE wIndex OF
      0 : sAdd := 'Day ';
      1 : sAdd := '$nTime ';
      2,3:sAdd := ':';
   END_CASE;
   (* 添加字符串。 *)
   sTimeString := CONCAT(sTimeString, sAdd, sItemString);
END_FOR;
(* 将创建的字符串设置为显示字符串。 *)
G_sDisplayedCharacters := sTimeString;
```

**要点**

使用通用函数的TIME_TO_STRING时，以ms为单位的值会转换成字符串。本程序中，先求出按日、时、分、秒分开的值，再转换为字符串。

“$n”是换行代码。

## 变量

在GX Works3中作如下设置，定义标签。

### ■全局标签

| 标签名 | 数据类型 | 类 | 分配/初始值/常数 | 内容 |
|---|---|---|---|---|
| G_tmTime | 时间 | VAR_GLOBAL | — | 总运转时间 （～T#24d20h31m23s647ms） |
| G_bOneScanOnly | 位 | VAR_GLOBAL | 分配： SM402 | RUN后仅1次扫描ON |
| G_sDisplayedCharacters | 字符串(32) | VAR_GLOBAL | — | 运转时间显示画面的显示字符 |

### ■局部标签

| 标签名 | 数据类型 | 类 | 初始值/常数 | 内容 |
|---|---|---|---|---|
| dUnitArray | 双字[有符号](0..3) | VAR | — | 单位 （日、时、分、秒的ms数） |
| wDigitArray | 字[有符号](0..1) | VAR | — | 数位指定 （[0]：所有位数，[1]：小数位数） |
| sTimeString | 字符串(32) | VAR | — | 创建的字符串 （日、时、分、秒） |
| dTime | 双字[有符号] | VAR | — | 时间 （ms单位） |
| wIndex | 字[有符号] | VAR | — | 元素编号 |
| wTime | 字[有符号] | VAR | — | 时间 （日/时/分/秒） |
| sItemString | 字符串(32) | VAR | — | 日/时/分/秒的2位数值字符串 |
| sAdd | 字符串(32) | VAR | — | 添加的字符串 |

## 程序部件

关于使用的程序部件，请参照以下内容。

| 数据名 | 数据类型 | 内容 | 参照 |
|---|---|---|---|
| TIME_TO_DINT | 通用函数 | TIME型→DINT型转换<br>将TIME型数据转换成DINT型数据。 | MELSEC iQ-R 编程手册(CPU模块用指令/通用FUN/通用FB篇)<br>MELSEC iQ-F FX5编程手册(指令/通用FUN/FB篇) |
| DINT_TO_INT | 通用函数 | DINT型→INT型转换<br>将DINT型数据转换成INT型数据。 | |
| STR | 指令 | 在BIN16位数据的指定位置添加小数点并将其转换为字符串。 | |
| CONCAT | 通用函数 | 字符串的合并<br>合并并输出字符串。 | |

# 附录

## 附1　ST语言规格

以下所示为ST语言中使用的构成单位的类型及内容。

### 语句

以下所示为语句的类型和表述方法。

| 类型 | | | 表述方法 | 内容 |
|---|---|---|---|---|
| 赋值语句 | | | 〈变量〉 :=〈表达式〉;<br>代入结果 | 将右边的评价结果代入左边的变量。 |
| 子程序控制语句 | 调用语句 | | 〈识别符〉(参数1, 参数2, ...); | 调用函数、FB等。 |
| | RETURN语句 | | RETURN; | 中途结束程序。 |
| 控制语法 | 选择语句 | IF | IF 〈条件式1〉 THEN<br>〈执行语句1〉;<br>ELSIF 〈条件式2〉 THEN<br>〈执行语句2〉;<br>—ELSIF（---线内）可多个<br>ELSE<br>〈执行语句3〉;<br>—ELSIF, ELSE（---线内）可省略<br>END_IF; | 根据BOOL值的条件，选择执行语句。<br>条件式1为真（TRUE）时，执行执行语句1。<br>条件式1为假（FALSE）时，对“ELSIF”的条件进行判断。<br>条件式2为真（TRUE）时，执行执行语句2。<br>“IF”及“ELSIF”的所有条件式均为假（FALSE）时，执行“ELSE”后的执行语句3。 |
| | | CASE | CASE 〈条件式〉 OF<br>〈整数值1〉 :<br>〈执行语句1〉;<br>〈整数值2〉..〈整数值3〉 :<br>〈执行语句2〉;<br>—可多个<br>ELSE<br>〈执行语句3〉;<br>—ELSE（---线内）可省略<br>END_CASE; | 根据整数值的条件，选择执行语句。<br>条件式的结果与整数值1一致时，执行执行语句1。<br>要通过范围指定要判断的整数值时，使用“..”表述。条件式结果的值在整数值2～整数值3的范围内时，执行执行语句2。<br>与所有的整数值或范围都不一致时，执行“ELSE”后的执行语句3。 |
| | 循环语句 | FOR | FOR 〈变量〉 :=〈初始值（表达式）〉<br>TO 〈最终值（表达式）〉<br>BY 〈增加值（表达式）〉—BY（---线内）可省略<br>DO 〈执行语句〉;<br>END_FOR; | 根据整数值的结束条件，多次执行执行语句。<br>执行执行语句直到设置了初始值的整数型变量变为最终值为止。每执行1次执行语句，即在变量上加上增加值。 |
| | | WHILE | WHILE 〈条件式〉<br>DO<br>〈执行语句〉;<br>循环直至变为假（FALSE）<br>END_WHILE; | 根据BOOL值的结束条件，多次执行执行语句。<br>条件式为真（TRUE）时，执行执行语句。反复执行直到条件式的结果变为假（FALSE）为止。 |
| | | REPEAT | REPEAT<br>〈执行语句〉;<br>循环直至变为真（TRUE）<br>UNTIL 〈条件式〉<br>END_REPEAT; | 根据BOOL值的结束条件，多次执行执行语句。<br>执行执行语句后，对条件进行判断。<br>反复执行直到条件式的结果变为真（TRUE）为止。 |
| | 循环语句的中断 | | EXIT; | 中断循环语句。 |
| 空语句 | | | ; | 不执行任何处理。 |

**要点**

函数和FB总计可以进行32次分层。

IF语句、CASE语句、FOR语句、WHILE语句、REPEAT语句总计可以进行128次分层。

附

# 运算符

以下所示为运算符的类型和表述方法。

| 类型 | | 运算符 | 优先顺序（从高到低的顺序） | 示例 | |
|---|---|---|---|---|---|
| | | | | 一般算式表述 | ST |
| 符号的反转 | | - | 1 | B = - A | eValueB := - eValueA; |
| 逻辑运算 | 逻辑非 | NOT | | $B = \overline{A}$ | bFlagB := NOT bFlagA; |
| 幂乘 | | ** | 2 | $B = C^A$ | eValueB := eValueC ** eValueA; |
| 四则运算 | 乘法运算 | * | 3 | A × B = C | eValueC := eValueA * eValueB; |
| | 除法运算 | / | | A ÷ B = C ... D | eValueC := eValueA / eValueB; |
| | 余数运算 | MOD | | | eValueC := eValueA MOD eValueB; |
| | 加法运算 | + | 4 | A + B = C | eValueC := eValueA + eValueB; |
| | 减法运算 | - | | A - B = C | eValueC := eValueA - eValueB; |
| 比较运算 | 大于、小于 | >、< | 5 | A > B | bFlag := eValueA > eValueB; |
| | 大于等于、小于等于 | >=、<= | | A ≤ B | bFlag := eValueA <= eValueB; |
| | 等于 | = | 6 | A = B | bFlag := eValueA = eValueB; |
| | 不等于 | <> | | A ≠ B | bFlag := eValueA <> eValueB; |
| 逻辑运算 | 逻辑与 | AND、& | 7 | A ∧ B | bFlag := eValueA AND eValueB; |
| | 逻辑异或 | XOR | 8 | A ∀ B | bFlag := eValueA XOR eValueB; |
| | 逻辑或 | OR | 9 | A ∨ B | bFlag := eValueA OR eValueB; |

**要点**

1个表达式中最多可使用1024个运算符。

## 优先顺序

将多个运算表达式表述在1个表达式中时，将从优先级高的运算符开始处理。
有多个优先级相同的运算符时，从最左边的运算符开始运算。
小括号()内的运算表达式将优先进行运算。

# 注释

以下所示为注释的类型和表述方法。

| 类型 | 符号 | 内容 | 示例 |
|---|---|---|---|
| 多行注释 | (* *) | 从开始符号到结束符号为止的内容作为注释处理。 | (* 注释 *) |
| | /* */ | | /* 注释 */ |
| 单行注释 | // | 从开始符号到行尾为止的内容作为注释处理。 | // 注释 |

**要点**

IEC 61131-3中定义的ST的注释符号仅有(**)，但在GX Works3中可以使用与C语言等相同的符号（/**/、//）来表述注释。
GX Works2中，仅可使用带有(**)的注释。

# 软元件

可与梯形图一样指定软元件。带#号时可以指定本地软元件。
可使用软元件的指定 （数位指定、位指定、间接指定）及变址修饰。

**限制事项**

ST程序中无法使用指针。

## 定时器、计数器的触点/线圈/当前值

在ST语言中，可指明触点/线圈/当前值以使用定时器、计数器等软元件。
没有指定时，会根据使用的指令自动判断触点/线圈/当前值。
触点、线圈按位型处理，当前值按以下数据类型处理。

| 软元件 | 表示方法 | | | | 当前值的数据类型 |
|---|---|---|---|---|---|
| | 无指定 | 触点 | 线圈 | 当前值 | |
| 定时器 | T | TS | TC | TN | 字[无符号]/位列[16位] |
| 累积定时器 | ST | STS | STC | STN | |
| 计数器 | C | CS | CC | CN | |
| 长定时器 | LT | LTS | LTC | LTN | 双字[无符号]/位列[32位] |
| 长累积定时器 | LST | LSTS | LSTC | LSTN | |
| 长计数器 | LC | LCS | LCC | LCN | |

## 字软元件的类型指定

字软元件可指明数据类型进行使用。

| 数据类型 | 软元件类型指定符 | 示例 | 说明 |
|---|---|---|---|
| 字[无符号]/位列[16位] | :U | D0:U | 将D0的值按WORD型16位的值处理。 |
| 双字[无符号]/位列[32位] | :UD | D0:UD | 将D0、D1的值按DWORD型32位的值处理。 |
| 字[有符号] | （无） | D0 | 将D0的值按INT型16位的值处理。 |
| 双字[有符号] | :D | D0:D | 将D0、D1的值按DINT型32位的值处理。 |
| 单精度实数 | :E | D0:E | 将D0、D1的值按REAL型32位的值处理。 |
| 双精度实数 | :ED | D0:ED | 将D0～D3的值按LREAL型64位的值处理。 |

对进行了数位指定或间接指定的软元件，不可添加软元件类型指定符。

### ■字软元件的类型转换

将没有指定类型的字软元件指定给了函数或FB的参数时，将按参数定义的数据类型处理。（通用函数、通用FB及指令时也一样。）
指定给输入参数时会自动进行类型转换，因此根据对参数的指定方法结果会有所不同。

例

BIN32位数据传送指令 （DMOV）时 （输入参数、输出参数为ANY32型的指令）

- D0 = 16#ABCD
- D1 = 16#1234
- 分配了D0的字[有符号]型标签G_wLabel

| ST | 内容 | 传送来的值 |
|---|---|---|
| bResult := DMOV(TRUE, D0, D10); | 将存储在D0、D1中的32位数据传送到D10、D11。 | 16#1234ABCD |
| bResult := DMOV(TRUE, D0:UD, D10:UD); | | |
| bResult := DMOV(TRUE, D0:U, D10); | 存储在D0中的字[有符号]/位列[16位]型的整数值会被自动转换为双字[有符号]型，零扩展的值会被传送到D10、D11。 | 16#0000ABCD |
| bResult := DMOV(TRUE, G_wLebel, D10); | 存储在D0中的字[有符号]型的整数值会被自动转换为双字[有符号]型的整数值，符号扩展的值会被传送到D10、D11。 | 16#FFFFABCD |
| bResult := DMOV(TRUE, D0:U, D10:U); | 输出变量无法自动转换，因此会出现D10:U转换错误。 | — |

在运算表达式中使用未指定类型的字软元件时，将会执行从字[有符号]型数据的自动转换。
☞ 29页 可自动转换的数据类型

# 标签

可与梯形图一样指定标签。
可以使用标签的位指定 （例： Lbl.3）、数位指定 （例： K4Lbl）。

**限制事项**

ST程序中无法使用指针型标签。

# 常数

以下所示为常数的表述方法。

| 类型 | | 表示方法 | 表示示例 | 带“_”的表示示例*1 |
|---|---|---|---|---|
| BOOL值 | | 以TRUE或FALSE表述。 | TRUE、FALSE | |
| | | 用1或0的值表述。可使用整数的各种表示方法。 | 2#0、8#1、0、H1 | |
| 整数 | 2进制数 | 2进制数前加上“2#”。 | 2#0010、2#01101010 | 2#111_1111_1111_1111 |
| | 8进制数 | 8进制数前加上“8#”。 | 8#2、8#152、8#377 | 8#7_7777 |
| | 10进制数 | 直接输入10进制数。 | 2、106、-1 | 32_767 |
| | | 10进制数前加上“K”。 | K2、K106、K-1 | —*2 |
| | 16进制数 | 16进制数前加上“16#”。 | 16#2、16#6A、16#FF | 16#7F_FF |
| | | 16进制数前加上“H”。 | H2、H6A、HFF | —*2 |
| 实数 | 小数表示 | 直接输入实数。 | 1200.0、0.012、-0.1 | 3.14_159 |
| | | 实数前加上“E”。 | E1200、E0.012、E-0.1 | —*2 |
| | 指数表示 | 尾数部和指数之间加上“E”。<br>“mEn”表示尾数部m乘以10的n次幂。 | 1.2E3、1.2E-2、-1.0E-1 | 2.99_792_458E8 |
| | | 尾数部前面加上“E”，尾数部和指数之间加上“+”/“-”。<br>“Em+n”表示尾数部m乘以10的n次幂。*3 | E1.2+3、E1.2-2、E-1.0-1 | —*2 |
| 字符串 | ASCII<br>Shift JIS | 字符串使用单引号（’）括起来。 | ’ABC’ | |
| | Unicode | 字符串使用双引号（″）括起来。 | ″ABC″ | |
| 时间 | | 前面加上“T#”或“TIME#”。 | T#1h、T#1d2h3m4s5ms、TIME#1h | |

*1 在整数、实数的表示中，使用下划线（_）区隔数值，可使程序更易读懂。在程序的处理中，通过下划线（_）进行的数值区隔将被忽略。

*2 前面加了K、H、E时，无法使用下划线（_）。

*3 在前面加有“E”的实数表示中，如果在值后面再加上“+”/“-”，则会作为指数处理。
要表述为算术运算时，应在数值和运算符之间插入空格或制表符。
（例：“E1.2+3”表示1200.0，“E1.2 +3”表示4.2。）

## 指定数据类型时

指定以下值时可以标明数据类型。不指定时，将自动判断数据类型。
不区分数据类型名的大写字母/小写字母。不能与使用了K、H、E的常数的表示同时使用。

| 常数 | 可指定的数据类型 | | 表示示例 | 带“_”的表示示例 |
|---|---|---|---|---|
| BOOL值 | 位 | BOOL | BOOL#TRUE、BOOL#FALSE、BOOL#0、BOOL#1 | |
| 整数 | 字[无符号]/位列[16位] | UINT | UINT#2#01101010、UINT#8#152、UINT#106、UINT#16#6A | UINT#2#0110_1010 |
| | | WORD*1 | WORD#2#01101010、WORD#8#152、WORD#106、WORD#16#6A | WORD#2#0110_1010 |
| | 双字[无符号]/位列[32位] | UDINT | UDINT#2#11111111111111、UDINT#8#77777、UDINT#32767、UDINT#16#7FFF | UDINT#16#7F_FF |
| | | DWORD*1 | DWORD#2#11111111111111、DWORD#8#77777、DWORD#32767、DWORD#16#7FFF | DWORD#16#7F_FF |
| | 字[有符号] | INT | INT#2#01101010、INT#8#152、INT#106、INT#16#6A | INT#2#0110_1010 |
| | 双字[有符号] | DINT | DINT#2#11111111111111、DINT#8#77777、DINT#32767、DINT#16#7FFF | DINT#16#7F_FF |
| 实数 | 单精度实数 | REAL | REAL#2.34、REAL#1.0E6 | REAL#3.14_159 |
| | 双精度实数 | LREAL | LREAL#-2.34、LREAL#1.001E16 | LREAL#1.00_1E16 |

*1 四则运算公式的操作数(运算对象值)及函数、FB的调用语句及函数调用表达式的ANY_NUM型的参数时，不能使用“WORD#”及“DWORD#”。应使用“UINT#”或“UDINT#”。

附

## 字符串型的常数中使用“$”时

在字符串型常数中要指定换行等时，使用“$”表示。

| 类型 | 表示 |
|---|---|
| 美元符号（$） | $$、$24 |
| 单引号（’） | $’、$27 |
| 双引号（″） | $″、$22 |
| 换行（Line feed） | $L、$l、$0A |
| 换行（Newline） | $N、$n、$0D $0A |
| 翻页 | $P、$p、$0C |
| 回车符 | $R、$r、$0D |
| 制表符 | $T、$t、$09 |
| ASCII码对应字符 | $[ASCII码(2数位的16进制数)] |

## 时间长度的记述方法

时间型的常数按以下格式进行表述。

- T#23d23h59m59s999ms或TIME#23d23h59m59s999ms

| 时间单位 | d(日) | h(时) | m(分) | s(秒) | ms(毫秒) |
|---|---|---|---|---|---|
| 范围 | 0～24 | 0～23 | 0～59 | 0～59 | 0～999 |

时间单位需按d、h、m、s、ms的顺序连续。

可以在#的后面附加符号(-)，作为有符号的值进行记述。

（例： T#-24d20h31m23s648ms）

### ■时间单位的省略

前后的时间单位可以省略。

（例： T#1m2s）

中途的时间单位不能省略。

(例: 不能记述为“T#1m2ms”。应记述为“T#1m0s2ms”。)

对于最后的时间单位，可以通过使用了小数点的表示(无符号实数的10进制数表示)进行记述。

(例: “T#1.234s”被处理为“T#1s234ms”。)

ms(毫秒)单位无法以小数点表示。小数点以下将被舍去。

(例: “T#1.234ms”被处理为“T#1ms”。)

### ■设置范围

使用符号(-)，省略了前面的时间单位时，可记述的值的范围如下所示。

| 时间单位 | d(日) | h(时) | m(分) | s(秒) | ms(毫秒) |
|---|---|---|---|---|---|
| 无省略 | -24～24 | 0～23 | 0～59 | 0～59 | 0～999 |
| 省略d | — | -596～596 | 0～59 | 0～59 | 0～999 |
| 省略d、h | — | — | -35791～35791 | 0～59 | 0～999 |
| 省略d、h、m | — | — | — | -2147483～2147483 | 0～999 |
| 省略d、h、m、s | — | — | — | — | -2147483648～2147483647 |

时间型变量(☞ 32页 时间型 （TIME）变量)中可以在以下范围内设置时间长度。

- T#-24d20h31m23s648ms～T#24d20h31m23s647ms
- T#-596h31m23s648ms～T#596h31m23s647ms
- T#-35791m23s648ms～T#35791m23s647ms
- T#-2147483s648ms～T#2147483s647ms
- T#-2147483648ms～T#2147483647ms

# 函数和FB

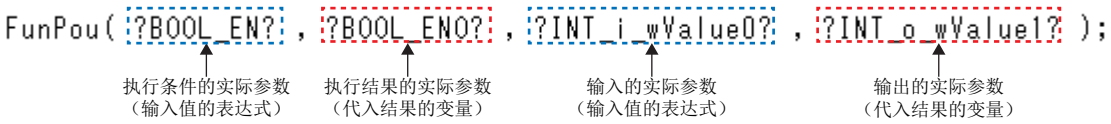
函数和FB是对程序中调用的子程序进行了定义的程序部件。
定义的函数可在程序块、FB及其他函数中使用。
定义的FB可在程序块及其他FB中通过创建实例来使用。

## 参数

本部分对调用语句中的参数的表述方法进行说明。

### ■实际参数的参数指定

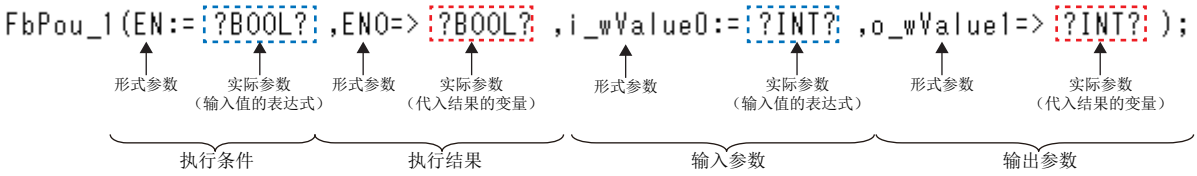
在调用语句的括号中，使用逗号（,）区隔列出参数。
函数的模板按此格式显示。（☞ 50页 输入参数）




参数的顺序是函数或FB定义的局部标签设置中的定义顺序。
如果创建时设置为使用EN/ENO，则第1参数指定为EN，第2参数指定为ENO。

### ■向形式参数中分配实际参数的参数指定

在调用语句的括号中，使用逗号（,）区隔列出形式参数和实际参数。
FB的模板按此格式显示。（☞ 51页 输入参数）

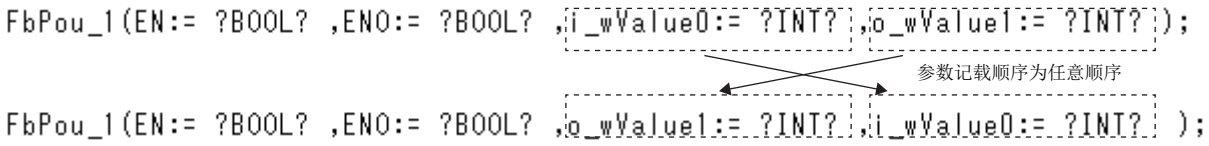



形式参数名是函数或FB定义的局部标签设置中指定的变量名。
EN/ENO指定“EN”、“ENO”作为形式参数。

**要点**

向形式参数中分配实际参数时，按以下格式指定。
- 输入参数、输入输出参数及EN时：“<形式参数名>:=<表达式>”
- 输出参数及ENO时：“<形式参数名>=><变量>”

向形式参数中分配实际参数时，可任意更改参数的表述顺序。

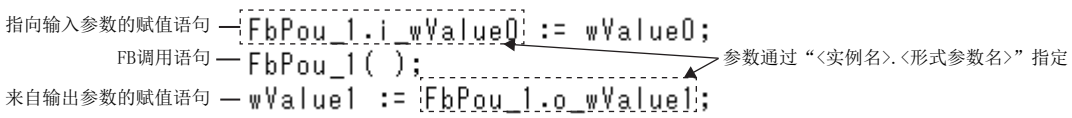



**要点**

向形式参数中分配实际参数并指定参数时，可省略参数的表述。

### ■调用语句前后的参数指定

FB可在调用语句的前后指定参数。表述时应将向输入参数的赋值语句写在调用语句之前，来自输出参数的赋值语句写在调用语句之后。




附

## 返回值

有返回值的函数会在执行结束后将值返回到调用源。

### ■返回值的数据类型

返回值的数据类型通过“New Data （新建数据）”画面或“Properties （属性）”画面设置。

没有设置数据类型时，将作为没有返回值的函数处理。

没有返回值的函数以调用语句表述。（函数调用语句）

### ■在返回值中设置值

在函数中，将作为返回值返回的值设置到将函数名作为识别符的变量中。

#### 程序示例

将输入参数i_wValue0～i_wValue2的和作为函数FunAdd的返回值返回。

**ST （函数FunAdd的程序）**

```
FunAdd := i_wValue0 + i_wValue1 + i_wValue2;
```

### ■在调用源使用返回值

有返回值的函数可作为表达式处理。（函数调用表达式）

在函数的调用源程序中，可将函数调用表达式作为返回值的数据类型的变量记述运算表达式或条件语句。

#### 程序示例

在Average3中代入wValue0～wValue2的平均值。

将输入参数wValue0～wValue2的和作为函数FunAdd的返回值返回。

**ST （调用源的程序）**

```
Average3 := FunAdd( wValue0, wValue1, wValue2) / 3;
```

## EN/ENO的思路

可选择EN/ENO的有无，创建函数及FB。

带上EN （使能输入）、ENO （使能输出），可对执行处理进行控制。

- EN：设置执行条件。
- ENO：输出执行结果。

EN/ENO为BOOL型。

带EN的函数及FB，仅在EN的执行条件为TRUE时执行。

以下所示为与EN和ENO状态所对应的输出变量及返回值。

| EN | ENO | 输出变量、返回值 | 备注 |
|---|---|---|---|
| TRUE | TRUE | 运算输出值 | 正常结束 |
| | FALSE | 不定值 | 执行处理中，ENO中代入了FALSE时。<br>输出的值视实际装置而定。 |
| FALSE | FALSE | • 函数：调用时的值<br>• FB：上次的结果 | 不执行处理，直接结束。<br>也不执行对输入变量、输出变量的赋值。 |

### ■在ENO中设置值

要在函数/FB内设置ENO的值时，应在变量“ENO”中代入BOOL值。

#### 程序示例

BCD指令中出现了运算错误时，将中断函数/FB的处理，并错误结束。

**ST （函数/FB的程序）**

```
ENO := BCD(EN, wValue0, DO);
IF ENO = FALSE THEN
    RETURN;
END_IF;
```

# 附2　ST中无法使用的指令

在ST中使用运算符或控制语句表述以下在梯形图中使用的指令。

## 赋值可使用的指令

| 类型 | 指令符号 | 对应的类型转换函数 |
|---|---|---|
| 单精度实数→有符号BIN16位数据 | FLT2INT(P) | REAL_TO_INT(_E) |
| 单精度实数→无符号BIN16位数据 | FLT2UINT(P) | —(REAL_TO_DINT(_E), DINT_TO_WORD(_E)) |
| 单精度实数→有符号BIN32位数据 | FLT2DINT(P) | REAL_TO_DINT(_E) |
| 单精度实数→无符号BIN32位数据 | FLT2UDINT(P) | —(REAL_TO_DINT(_E), DINT_TO_DWORD(_E)) |
| 双精度实数→有符号BIN16位数据 | DBL2INT(P) | LREAL_TO_INT(_E) |
| 双精度实数→无符号BIN16位数据 | DBL2UINT(P) | —(LREAL_TO_DINT(_E), DINT_TO_WORD(_E)) |
| 双精度实数→有符号BIN32位数据 | DBL2DINT(P) | LREAL_TO_DINT(_E) |
| 双精度实数→无符号BIN32位数据 | DBL2UDINT(P) | —(LREAL_TO_DINT(_E), DINT_TO_DWORD(_E)) |
| 有符号BIN16位数据→无符号BIN16位数据转换 | INT2UINT(P) | INT_TO_WORD(_E) |
| 有符号BIN16位数据→有符号BIN32位数据转换 | INT2DINT(P) | INT_TO_DINT(_E)*1 |
| 有符号BIN16位数据→无符号BIN32位数据转换 | INT2UDINT(P) | INT_TO_DWORD(_E) |
| 无符号BIN16位数据→有符号BIN16位数据转换 | UINT2INT(P) | WORD_TO_INT(_E) |
| 无符号BIN16位数据→有符号BIN32位数据转换 | UINT2DINT(P) | WORD_TO_DINT(_E)*1 |
| 无符号BIN16位数据→无符号BIN32位数据转换 | UINT2UDINT(P) | WORD_TO_DWORD(_E)*1 |
| 有符号BIN32位数据→有符号BIN16位数据转换 | DINT2INT(P) | DINT_TO_INT(_E) |
| 有符号BIN32位数据→无符号BIN16位数据转换 | DINT2UINT(P) | DINT_TO_WORD(_E) |
| 有符号BIN32位数据→无符号BIN32位数据转换 | DINT2UDINT(P) | DINT_TO_DWORD(_E) |
| 无符号BIN32位数据→有符号BIN16位数据转换 | UDINT2INT(P) | DWORD_TO_INT(_E) |
| 无符号BIN32位数据→无符号BIN16位数据转换 | UDINT2UINT(P) | DWORD_TO_WORD(_E) |
| 无符号BIN32位数据→有符号BIN32位数据转换 | UDINT2DINT(P) | DWORD_TO_DINT(_E) |
| 字符串传送 | $MOV(P) | —*1 |
| Unicode对应字符串传送 | $MOV(P)_WS | —*1 |
| 有符号BIN16位数据→单精度实数转换 | INT2FLT(P) | INT_TO_REAL(_E)*1 |
| 无符号BIN16位数据→单精度实数转换 | UINT2FLT(P) | —(WORD_TO_INT(_E), INT_TO_REAL(_E))*1 |
| 有符号BIN32位数据→单精度实数转换 | DINT2FLT(P) | DINT_TO_REAL(_E) |
| 无符号BIN32位数据→单精度实数转换 | UDINT2FLT(P) | —(DWORD_TO_DINT(_E), DINT_TO_REAL(_E)) |
| 双精度实数→单精度实数转换 | DBL2FLT(P) | LREAL_TO_REAL(_E) |
| 有符号BIN16位数据→双精度实数转换 | INT2DBL(P) | INT_TO_LREAL(_E)*1 |
| 无符号BIN16位数据→双精度实数转换 | UINT2DBL(P) | —(WORD_TO_DINT(_E), DINT_TO_LREAL(_E))*1 |
| 有符号BIN32位数据→双精度实数转换 | DINT2DBL(P) | DINT_TO_LREAL(_E)*1 |
| 无符号BIN32位数据→双精度实数转换 | UDINT2DBL(P) | —(DWORD_TO_DINT(_E), DINT_TO_LREAL(_E))*1 |
| 单精度实数→双精度实数转换 | FLT2DBL(P) | REAL_TO_LREAL(_E)*1 |

*1　在字符串的传送，或为可自动转换的数据类型 （☞ 29页 自动进行的类型转换）时，可以只通过代入进行表述。

## 可以使用运算符表述的指令

### 可以使用算术运算符表述的指令

| 类型 | 指令符号 |
|---|---|
| BIN16位加法运算 | +(P)(_U) [2个操作数时] |
| BIN16位减法运算 | -(P)(_U) [2个操作数时] |
| BIN32位加法运算 | D+(P)(_U) [2个操作数时] |
| BIN32位减法运算 | D-(P)(_U) [2个操作数时] |
| BIN16位乘法运算 | *(P)(_U) |
| BIN16位除法运算 | /(P)(_U) |
| BIN32位乘法运算 | D*(P)(_U) |
| BIN32位除法运算 | D/(P)(_U) |

| 类型 | 指令符号 |
| --- | --- |
| BCD4位加法运算 | B+(P) [2个操作数时] |
| BCD4位减法运算 | B-(P) [2个操作数时] |
| BCD8位加法运算 | DB+(P) [2个操作数时] |
| BCD8位减法运算 | DB-(P) [2个操作数时] |
| BCD4位乘法运算 | B*(P) |
| BCD4位除法运算 | B/(P) |
| BCD8位乘法运算 | DB*(P) |
| BCD8位除法运算 | DB/(P) |
| BIN16位块数据加法运算 | BK+(P)(_U) |
| BIN16位块数据减法运算 | BK-(P)(_U) |
| BIN32位块数据加法运算 | DBK+(P)(_U) |
| BIN32位块数据减法运算 | DBK-(P)(_U) |
| 字符串的合并 | $+(P) [2个操作数时] |
| 单精度实数加法运算 | E+(P) [2个操作数时] |
| 单精度实数减法运算 | E-(P) [2个操作数时] |
| 双精度实数加法运算 | ED+(P) [2个操作数时] |
| 双精度实数减法运算 | ED-(P) [2个操作数时] |
| 单精度实数乘法运算 | E*(P) |
| 单精度实数除法运算 | E/(P) |
| 双精度实数乘法运算 | ED*(P) |
| 双精度实数除法运算 | ED/(P) |
| 时钟数据的加法运算 | DATE+(P) |
| 时钟数据的减法运算 | DATE-(P) |
| 扩展时钟数据的加法运算 | S(P).DATE+ |
| 扩展时钟数据的减法运算 | S(P).DATE- |

## 可以使用逻辑运算符、比较运算符表述的指令

| 类型 | 指令符号 |
| --- | --- |
| 运算开始、串联连接、并列连接 | LD、LDI、AND、ANI、OR、ORI |
| 梯形图块串联连接、并列连接 | ANB、ORB |
| BIN16位数据比较 | LD□(_U)、AND□(_U)、OR□(_U) |
| BIN32位数据比较 | LDD□(_U)、ANDD□(_U)、ORD□(_U) |
| BIN16位块数据比较 | BKCMP□(P)(_U) |
| BIN32位块数据比较 | DBKCMP□(P)(_U) |
| 16位数据逻辑与 | WAND(P) [2个操作数时] |
| 32位数据逻辑与 | DAND(P) [2个操作数时] |
| 16位数据逻辑或 | WOR(P) [2个操作数时] |
| 32位数据逻辑或 | DOR(P) [2个操作数时] |
| 16位数据逻辑异或 | WXOR(P) [2个操作数时] |
| 32位数据逻辑异或 | DXOR(P) [2个操作数时] |
| 16位数据逻辑异或非 | WXNR(P) [2个操作数时] |
| 32位数据逻辑异或非 | DXNR(P) [2个操作数时] |
| 字符串比较 | LD$□、AND$□、OR$□ |
| 单精度实数比较 | LDE□、ANDE□、ORE□ |
| 双精度实数比较 | LDED□、ANDED□、ORED□ |
| 日期比较 | LDDT□、ANDDT□、ORDT□ |
| 时间比较 | LDTM□、ANDTM□、ORTM□ |

## 控制语句、函数等可使用的指令

| 类型 | 指令符号 |
| --- | --- |
| 结构化指令 | FOR、NEXT |
| 指针分支 | CJ、SCJ、JMP |
| 跳转至END | GOEND |
| 从中断程序返回 | IRET |
| FOR～NEXT强制结束 | BREAK(P) |
| 子程序调用 | CALL(P) |
| 从子程序返回 | RET |
| 子程序的输出OFF调用 | FCALL(P) |
| 程序文件间子程序调用 | ECALL(P) |
| 程序文件间子程序输出OFF调用 | EFCALL(P) |
| 子程序调用 | XCALL |

## 不需要的指令

| 类型 | 指令符号 |
| --- | --- |
| 顺控程序结束 | END |
| 无处理 （NOP） | NOP、NOPLF |

# 附3　使用MELSEC iQ-F系列时的注意事项

本手册主要记载了使用MELSEC iQ-R系列时的内容。
参考本手册使用MELSEC iQ-F系列创建ST程序时的注意事项如下所示。

## MELSEC iQ-F系列与MELSEC iQ-R系列的不同点

在MELSEC iQ-F系列中，以下规格与MELSEC iQ-R系列不同。
本手册记载的内容中，存在对MELSEC iQ-F系列产品创建ST程序时无法使用的软元件与数据类型，请加以注意。

| 项目 | | 不同点 | 参照 |
|---|---|---|---|
| 软元件 | | 可使用的软元件类型不同。<br>应确认使用的CPU模块的规格。 | MELSEC iQ-F FX5用户手册(应用篇) |
| 数据类型 | 双精度实数 （LREAL） | MELSEC iQ-F系列不支持。 | MELSEC iQ-F FX5编程手册(程序设计篇) |
| 常数 | 字符串[Unicode]型 | MELSEC iQ-F系列不支持。 | |

虽ST语言本身不存在差异，但除上述不同点外，MELSEC iQ-R系列与MELSEC iQ-F系列可能在指令等方面存在规格上的差异。
在创建ST程序时，应确认使用的模块的规格。

# 索引



索

# 备忘录

# 修订记录

*本手册编号在封底的左下角。

| 修订日期 | *手册编号 | 修订内容 |
|---|---|---|
| 2015年2月 | SH(NA)-081465CHN-A | 第一版 |
| 2016年8月 | SH(NA)-081465CHN-B | ■第二版<br>部分修改 |
| 2018年2月 | SH(NA)-081465CHN-C | ■第三版<br>部分修改 |
| 2020年11月 | SH(NA)-081465CHN-D | ■第四版<br>部分修改 |

日文手册编号： SH-081445-F

# 商标

Unicode is either a registered trademark or a trademark of Unicode, Inc. in the United States and other countries.

The company names, system names and product names mentioned in this manual are either registered trademarks or trademarks of their respective companies.

In some cases, trademark symbols such as '™' or '®' are not specified in this manual.

SH(NA)-081465CHN-D(2011)MEACH
MODEL: R-ST-GUIDE-C

三菱电机自动化(中国)有限公司
地址：上海市虹桥路1386号三菱电机自动化中心
邮编：200336
电话：021-23223030 传真：021-23223000
网址：http://cn.MitsubishiElectric.com/fa/zh/
技术支持热线 400-821-3030

扫描二维码,关注官方微博

扫描二维码,关注官方微信