T9可编程控制器软件手册

版本02 • 201203　　著：杨洪仁

软件手册

北京腾控科技有限公司

前言

用途

该手册叙述了如何使用 MULTIPROG Express 编程软件。
用户根据本手册可在腾控 T9 系列 PLC 上编写自己的程序。

基础知识要求

用户要熟悉本手册，需要具有自动化技术的一般知识，了解 Windows 操作系统，以及已经阅读了腾控《MULTIPROG 编程软件快速入门》。

MULTIPROG Express 的特点

支持 IEC 61131-3 编程语言——FBD、LD、IL、ST 和 SFC。
工程结构清晰，编程语言直观。
支持 FBD、LD 和 IL 之间的交叉编译，支持混合编程。
支持多用户编程，缩短工程编程时间。
用户界面可以使用德语、英语、法语、西班牙语、日语和汉语。
通过向导、交叉参考等资源可进行高效编程。
版本兼容可进行统一版本管理。

计算机硬件要求

设备	最低	推荐
CPU	500MHz	1GHz
内存	256MB	1GB
硬盘	500MB	1GB
监视器	1024×768	
操作系统	Windows XP Pro、2000、Vista，IE 5.0 以上	
通讯	TCP/IP、RS232	

MULTIPROG Express 支持

内容	数量
工程树中的节点	8000
工程树中的配置/资源	1/1
每个资源的程序实例	1000
每个资源的任务	1
每个任务的程序实例	500
每个 POU 的全局变量/局部变量	15000/15000
包含的库	32
一个工程中的 POU 数量（包括库中的多个 POU）	2000
一个工程支持的 I/O 数量	256 Byte
I/O 组	200

目录

1. 如何开始

本章叙述如何连接 PLC，如何用 MULTIPROG Express 创建一个程序，如何下载并运行这个程序。

本章分成以下四节：

1.1 连接 PLC

1.2 创建一个工程

1.3 创建一个程序

1.1 连接 PLC

连接 PLC 需要做两个工作，一是连接电源到 PLC，二是用通讯线连接电脑与 PLC。

连接电源

PLC 的供电电源是 220VAC，根据用户要求，也可定做成 24VDC 供电，在接线时要注意 PLC 的电源，不要接错，否则会损坏 PLC。

连接通讯线

MULTIPROG 与 PLC 之间通讯使用 Modbus TCP 协议，将一根以太网线分别连接计算机和 PLC 的以太网口。Modscan 与 PLC 之间通讯使用 Modbus TCP 或 Modbus RTU 协议，将一根 RS232 直连串行通讯线分别连接计算机的 COM1 口和 PLC 的 COM2 口。

1.1.1 MULTIPROG 连接计算机

使用以太网线连接 PLC 时，PLC 的出厂默认 IP 地址是：

IP 地址：192.168.1.**99**

子网掩码：255.255.255.0

网关：192.168.1.254

使用以太网线时，需要做以下工作：

第 1 步 把一根以太网线的一端插入 PLC 的以太网口，另一端插入计算机的以太网口；

第 2 步 修改计算机的 IP 地址，使其与 PLC 在同一个网段内，如

IP 地址：192.168.1.**100**

子网掩码：255.255.255.0

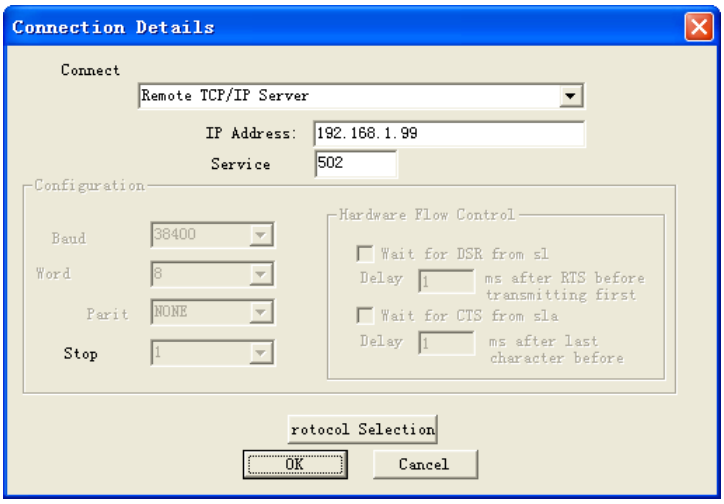
网关：192.168.1.254。

这样，PLC 就能连接到计算机了。如果计算机有无线网卡时，最好关掉，否则影响通讯。

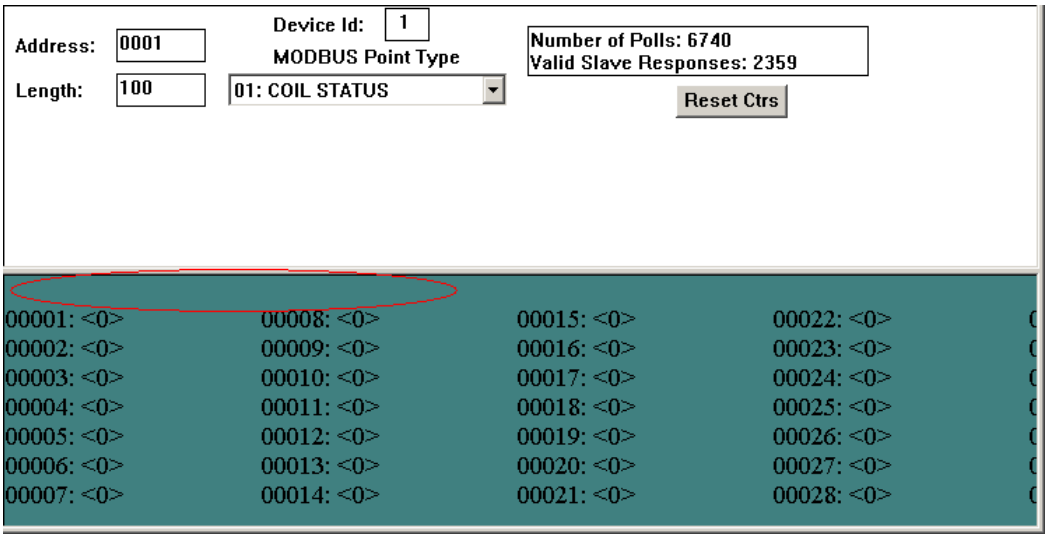
第 3 步 在计算机上运行 Modscan 软件修改 PLC 的 IP 地址，如果用户不希望修改，可不执行第 3 步。

1.1.2 Modscan 连接计算机

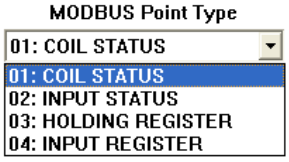
使用以太网线连接 PLC 时，打开 Modscan 文件夹，双击文件夹中的“ModScan32.exe”，在菜单栏选择“Connection->Connect”，在弹出对话框中选择 Remote TCP/IP Server、并输入 PLC 的 IP 地址（默认 192.168.1.99）、端口号默认 502，如下图，点击 OK 就能连接上了。



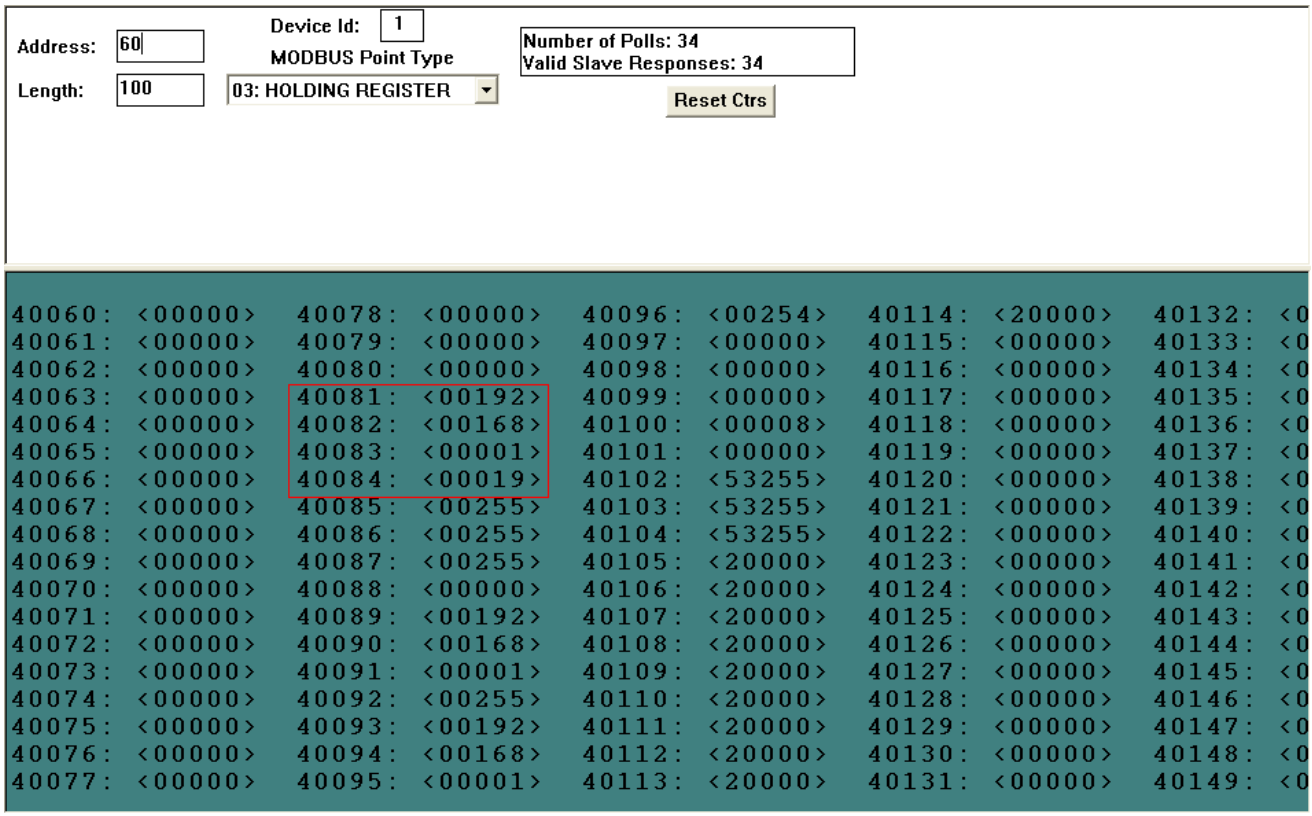
连接后的 ModScan 主界面如下图，如果没有连接上，在主界面中会有报警提示（红色字体）。



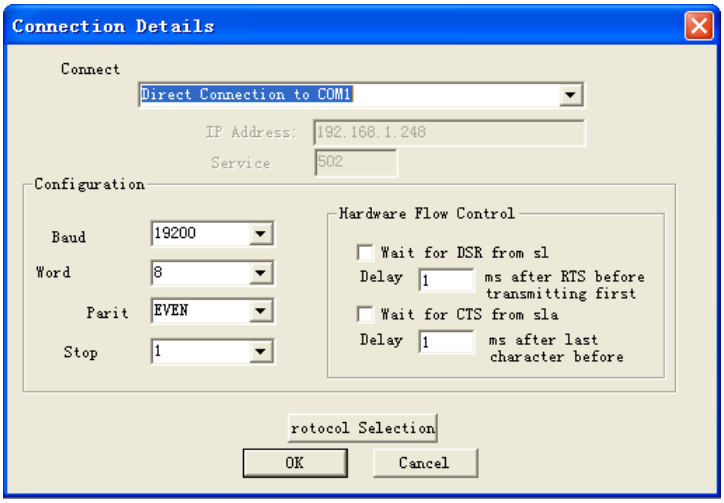
上图中，上部白色背景部分为输入栏，下部绿色背景部分为输出栏。在输入栏中，“Address”是要访问的 Modbus 的起始地址，“Length”是要访问的 Modbus 地址的数量（长度），“Device Id”是站号，“MODBUS Point Type”分为四个功能码分别是：01 线圈状态（数字量输出）、02 输入状态（数字量输入）、03 保持寄存器（PLC 的各保持寄存器、中间变量、模拟量输出等）和 04 输入寄存器（模拟量输入），从它的下拉列表中可以选择，如下图



如果需要修改 PLC 的 IP 地址，MODBUS Point Type 下拉框中，选择 03：HOLDING REGISTER，找到地址 40081~40084，即可修改 4 位 IP 地址，如下图



使用 RS232 直连串行通讯线时，一端连接电脑的 COM1 口，一端连接 PLC 的 COM2 口，打开 Modscan 文件夹，双击文件夹中的“ModScan32.exe”，在菜单栏选择“Connection->Connect”，在弹出对话框中选择 COM1 口，波特率 19200，8 位数据，一个停止位，偶校验，如下图，点击 OK 就能连接上了。

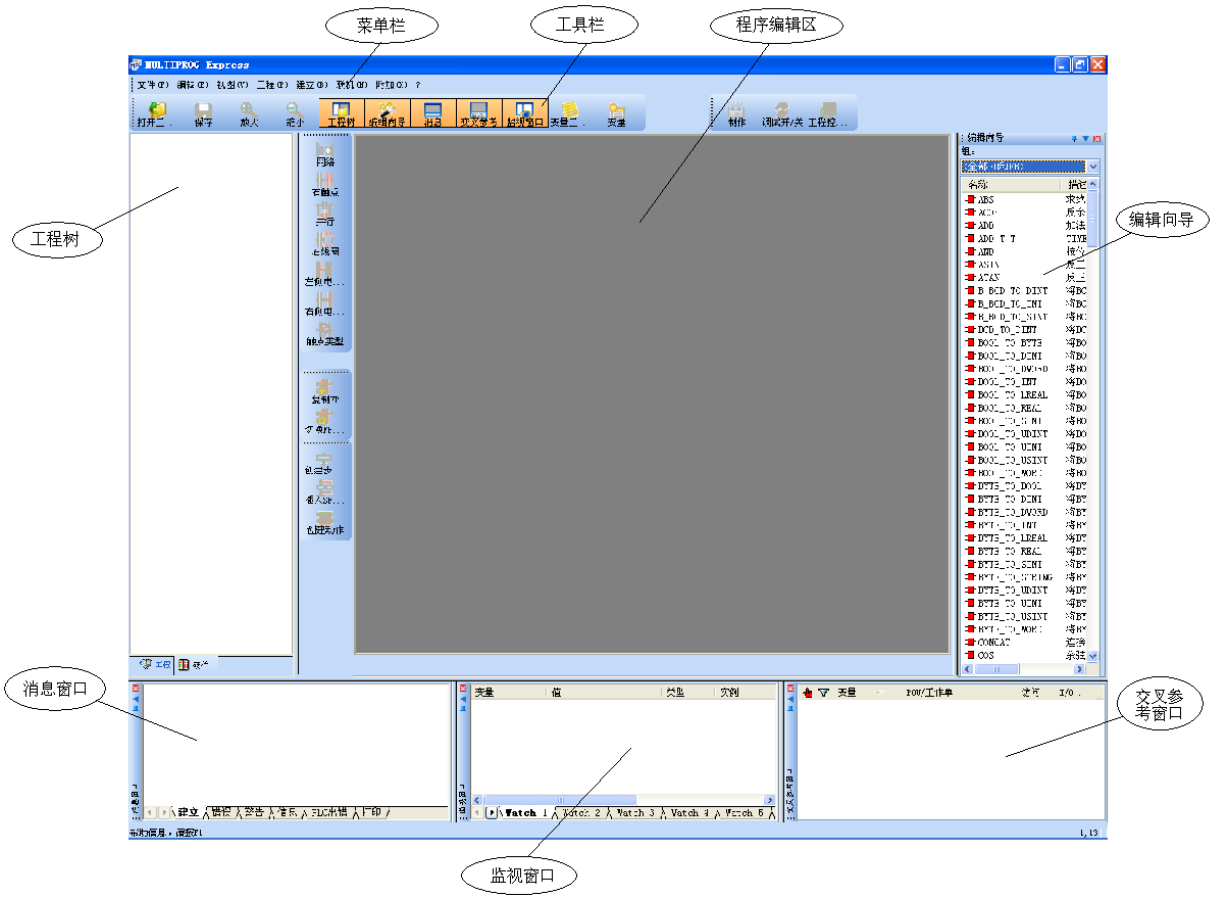


1.2 创建一个工程

第 1 步 启动 MULTIPROG 编程软件：

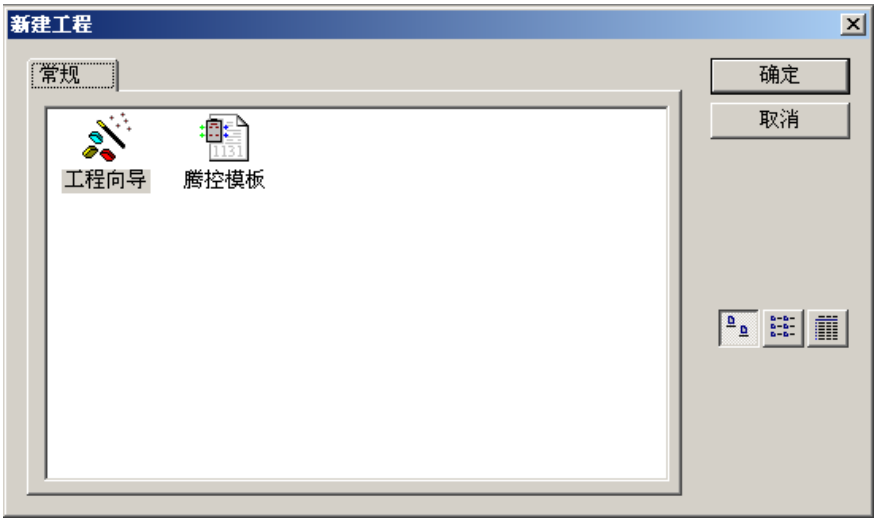
点击开始，所有程序→KW-Software→MULTIPROG 5.3 Express→ MULTIPROG 5.3 Express，或在桌面上双击 MULTIPROG 的快捷方式

MULTIPROG 的编程界面如下所示



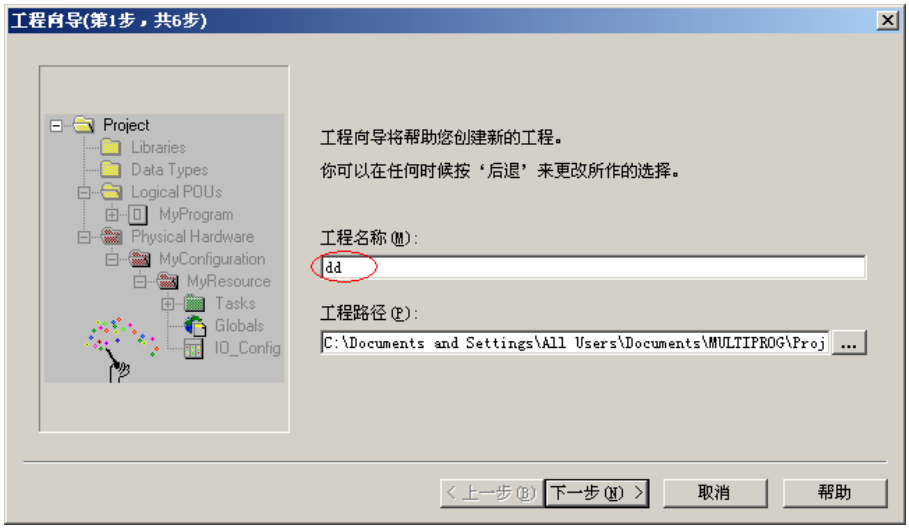
第 2 步 创建工程

选择在菜单栏“文件”中，选择“新建工程”，弹出新建工程对话框

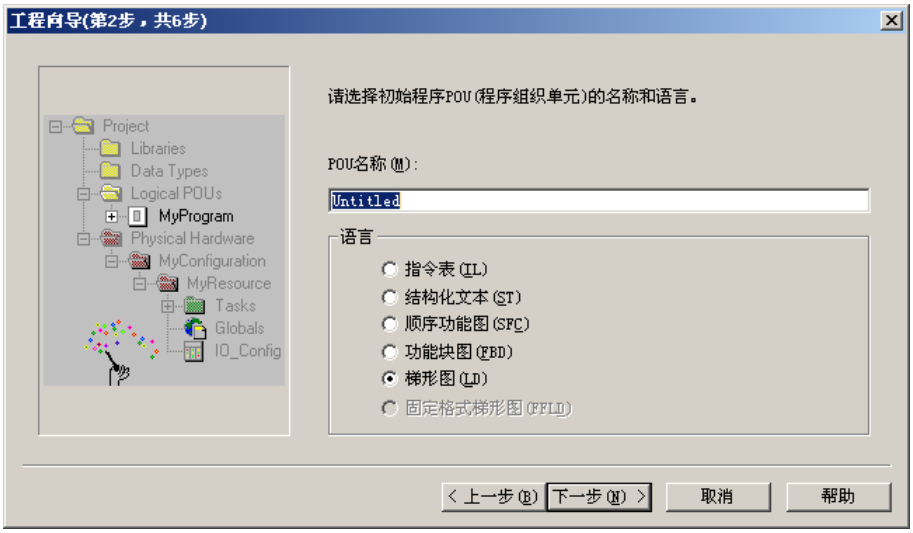


选择工程向导（用户也可选择腾控模版），点击“确定”，出现工程向导(第 1 步，共 6 步)对

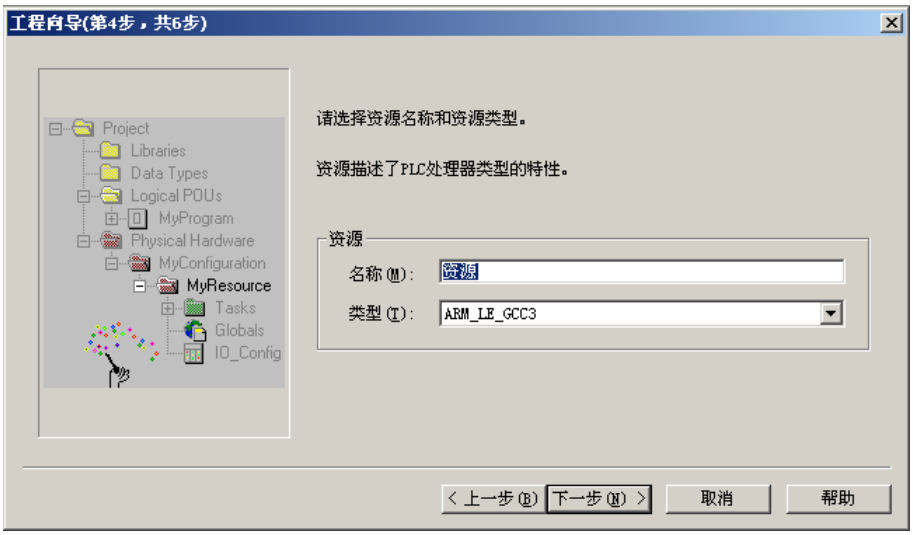
话框，如下图，在工程名称(M)栏输入工程名称，这里输入 dd，工程路径(P)栏的默认路径是 C:\Documents and Settings\All Users\Documents\MULTIPROG\Projects，用户可修改工程路径，这里不做修改。



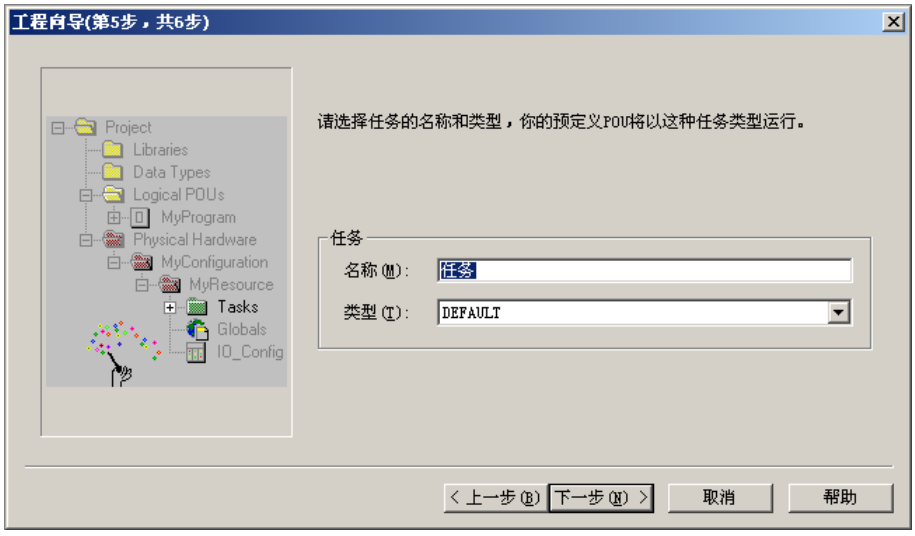
点击“下一步”，出现工程向导(第 2 步，共 6 步)对话框，如下图，POU 名称(M)栏默认的名称是 Untitled，用户可改为其他名字（不能用汉字），这里不做修改，在语言框中，可选择五种编程语言的任何一种，这里选择梯形图(LD)。



点击“下一步”，出现工程向导(第 3 步，共 6 步)对话框，如下图。

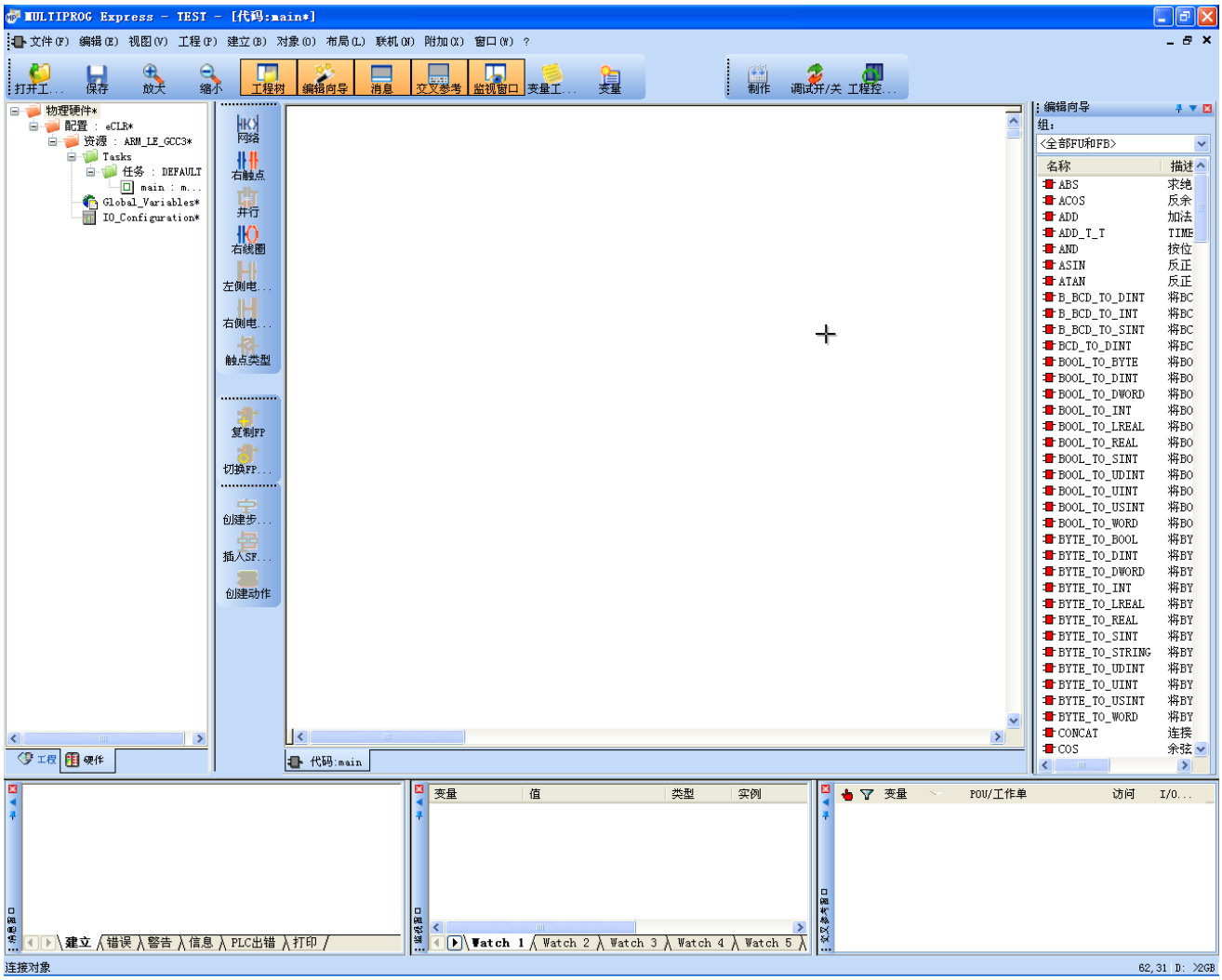


这里不做修改，点击“下一步”，出现工程向导(第 4 步，共 6 步)对话框，如下图。

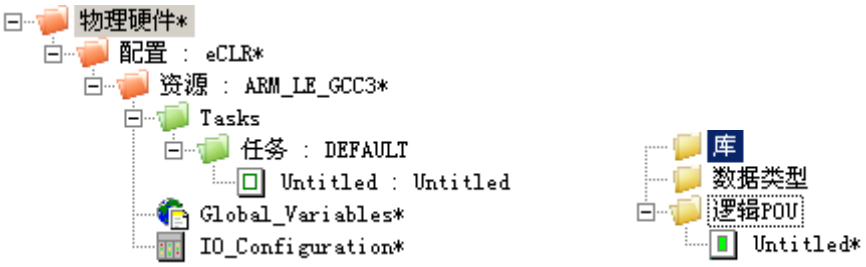


这里不做修改，点击“下一步”，出现工程向导(第 5 步，共 6 步)对话框，如下图。

在这里，描述了将要创建的工程文件，即前 5 步中所选择的内容，点击“完成”，就创建了一个初步的工程了，如下图。

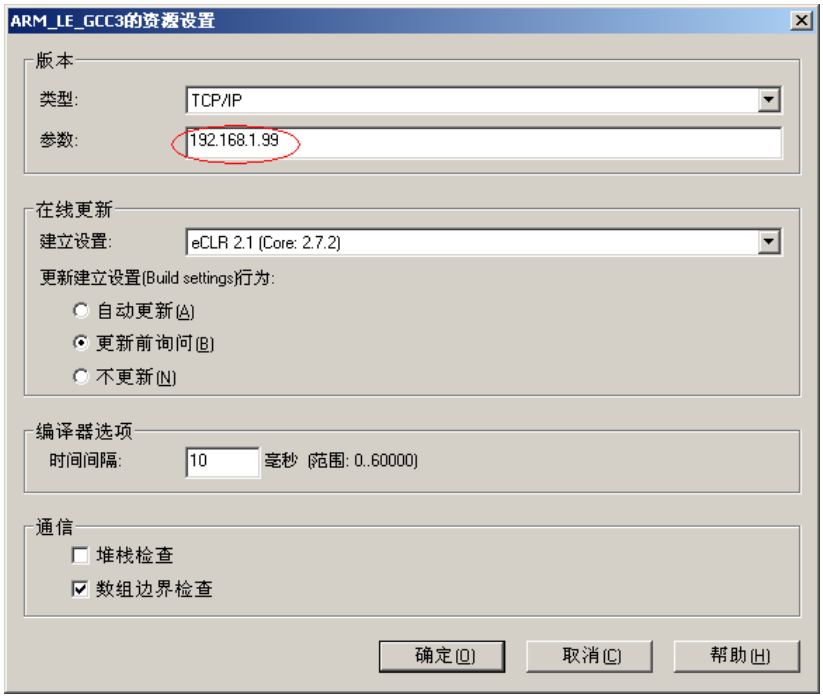


在上图中，MULTIPROG 编程界面左侧的“硬件”选项卡中出现了所创建的工程的树形目录，在“程序”选项卡中也出现了树形目录，如下两图



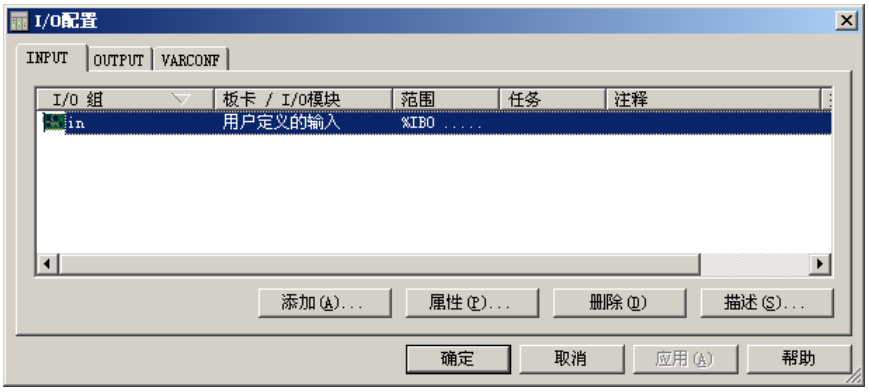
在“硬件”选项卡中，右击

选择“设置”，出现资源设置对话框，如下图，在版本→类型栏，默认 TCP/IP，不做修改，在版本→参数栏，默认 IP 地址是 127.0.0.1，这里要修改称 PLC 的实际 IP 地址 192.168.1.99，如 1.1.2 节所述。

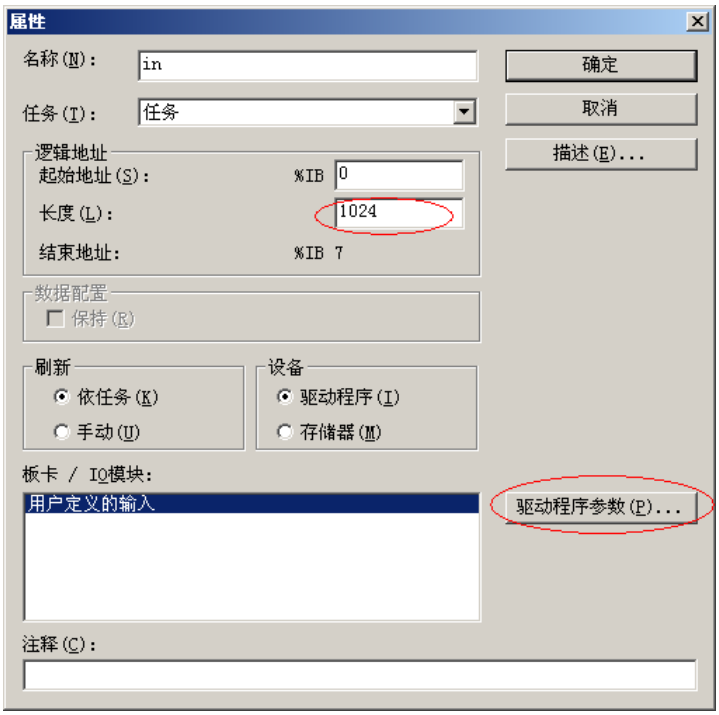


点击“确定”。在“硬件”选项卡中，双击

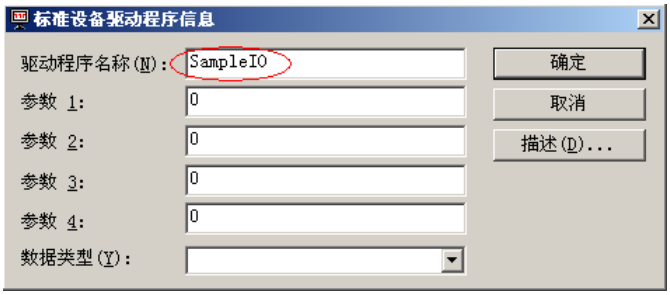
出现 I/O 配置对话框，如下图，选中 INPUT 选项卡，确保 I/O 组中的这一行为蓝色，点击“属性(P)…”。



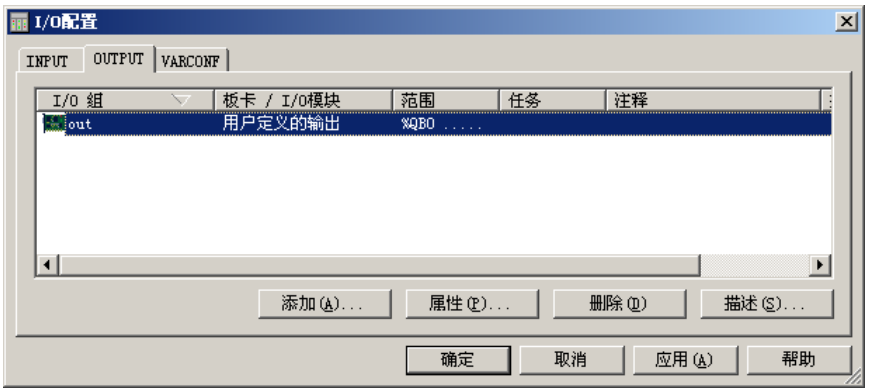
出现输入的属性设置对话框，在逻辑地址→%IB 的第二栏，默认 8，这里改为 1024。



点击“驱动程序参数(P) …”，出现标准设备驱动程序信息对话框，如下图，在驱动程序名称(N)栏，默认 DUMMYIO，这里改为 SampleIO，然后点击“确定”。



回到输入的属性设置对话框，点击“确定”，回到 I/O 配置对话框，在这里选中 OUTPUT 选项卡，确保 I/O 组中的这一行为蓝色，如下图



同 INPUT 选项卡一样，设置逻辑地址→%QB 的第二栏为 1024，驱动程序名称(N)栏为 SampleIO，设置完后，点击上图——I/O 配置对话框的“确定”，这样一个工程建立了。

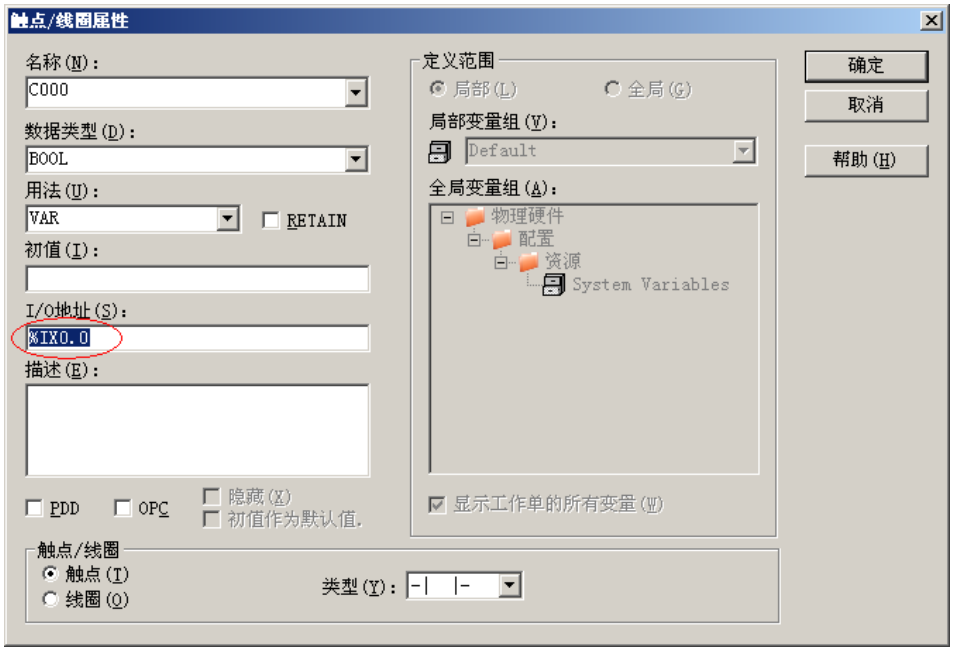
当然，如果在第 2 步开始时，选择了腾控模版，则上述第 2 步中的各种设置就由模版自动完成了，用户无需再做其他设置了。

1.3 创建一个程序

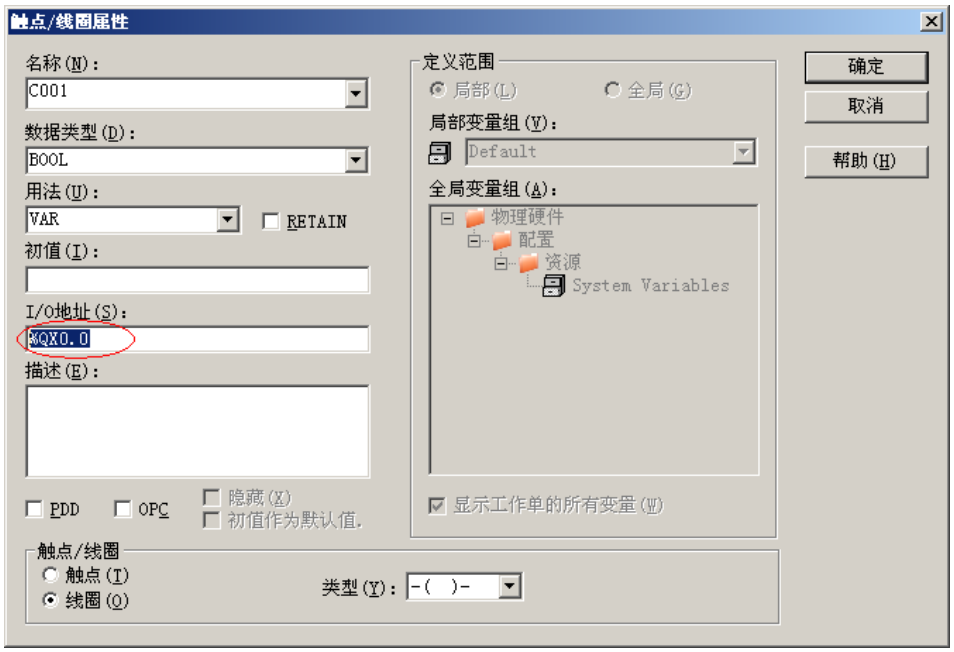
创建一个工程后，点击 MULTIPROG 编程软件的编辑区域，然后点击编辑区域左侧的

图标，在编辑区域中出现了一个梯形图的简单网络，左侧是一个常开触点，变量名是 C000，右侧是一个线圈，变量名是 C001，如下图。

双击常开触点 C000，出现触点/线圈属性对话框，在 I/O 地址(S)栏，输入%IX0.0，表示 PLC 本机的第一个数字量输入通道，点击“确定”，如下图。

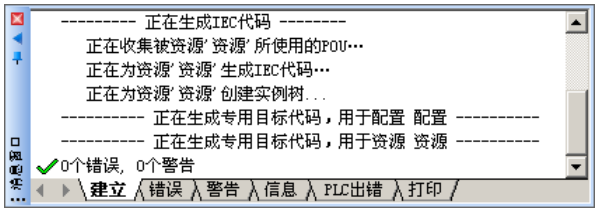


双击梯形图中线圈 C001，出现触点/线圈属性对话框，在 I/O 地址(S)栏，输入%QX0.0，表示 PLC 本机的第一个数字量输出通道，点击“确定”，如下图。



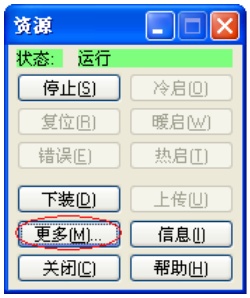
然后在 MULTIPROG 编程软件的菜单栏中点击“制作”，如下图

编译程序，编译过程中，在左下角的消息窗口显示编译过程，如下图。编译通过后，显示编译结果，如果出现“0 个错误，0 个警告”，说明编译成功，否则再根据错误提示修改程序，直到没有错误和警告发生。

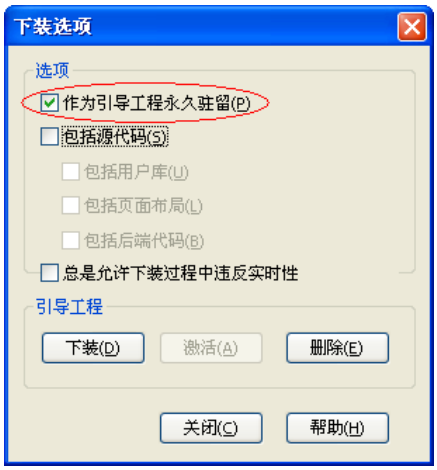


然后，在 MULTIPROG 编程软件的菜单栏中点击“工程控制对话框”，如下图

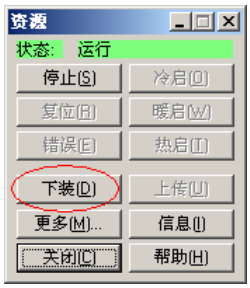
出现资源对话框，如下图。



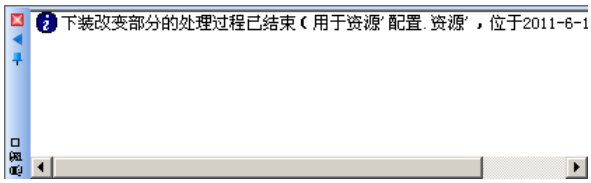
在上图中，点击“更多(M)”，弹出如下窗口。



在上图中，勾选上“作为引导工程永久驻留(P)”，然后点击“关闭”，回到上一层窗口，如下。



在上图中，点击“下装(D)”，程序下载到 PLC 中，如果下载成功，在左下角的消息窗口显示下载状态，如下图，此时，程序在 PLC 中已经开始执行了。如果在第一个数字量输入通道引入一个信号（24VDC 开关量信号），则第一个数字量输出通道就导通了。



如果下载失败，在消息窗口中会提示相关失败信息，下载失败时，程序无法执行。为了监视程序的执行状态，可在 MULTIPROG 编程软件的菜单栏中点击“调试开/关”，如下图

如果梯 PLC 外接设备的触点闭合，则外接设备的线圈导通，他们变为红色，如下图。

如果外接设备的触点断开，则线圈断开，梯形图中为蓝色，如下图。

如果 PLC 没有外接设备，则双击梯形图中的触点或线圈，在弹出窗口中点击“强制”或“覆盖”也可产生动作。


调试: 资源
强制/覆盖
C000
值
TRUE
FALSE
强制(O)
复位强制(E)
覆盖(W)
复位强制列表(L)
断点
设置(S)
复位(R)
全部复位(A)
值的显示
标准(D)
十进制(M)
十六进制(X)
二进制(Y)
实数值
宽度(I):
精度(P):
3
7
关闭(C)
信息(I)
帮助(H)

2. PLC 工作原理

T9 系列 PLC 根据来自被控设备的信号，经用户编写的程序对其处理、运算，把结果发送到现场设备，实现自动控制的目的，本章介绍程序如何执行和存储器的分配。

本章分成以下四节：

2.1 T9 系列 PLC 如何执行用户编写的程序

2.2 T9 系列 PLC 数据存取

2.3 T9 系列 PLC 掉电保持数据

2.1 T9 系列 PLC 如何执行用户编写的程序

用户把程序下载到 PLC，然后运行 PLC，就能循环执行用户的程序了。CPU 停止时，程序不执行。

PLC 每执行一次用户的程序称为一个扫描周期，在一个扫描周期中，将执行以下工作：

A. 读输入：将数字量、模拟量输入信号读取到输入映射区中。
B. 执行程序：执行用户的程序指令并将中间、最终数据存储在各种存储区中。
C. 处理通讯请求：执行通讯任务。
D. CPU 自诊断：检查固件、程序存储器工作状态。
E. 写输出：在输出映射区中存储的数据被复制到物理输出点。

2.2 T9 系列 PLC 数据存取

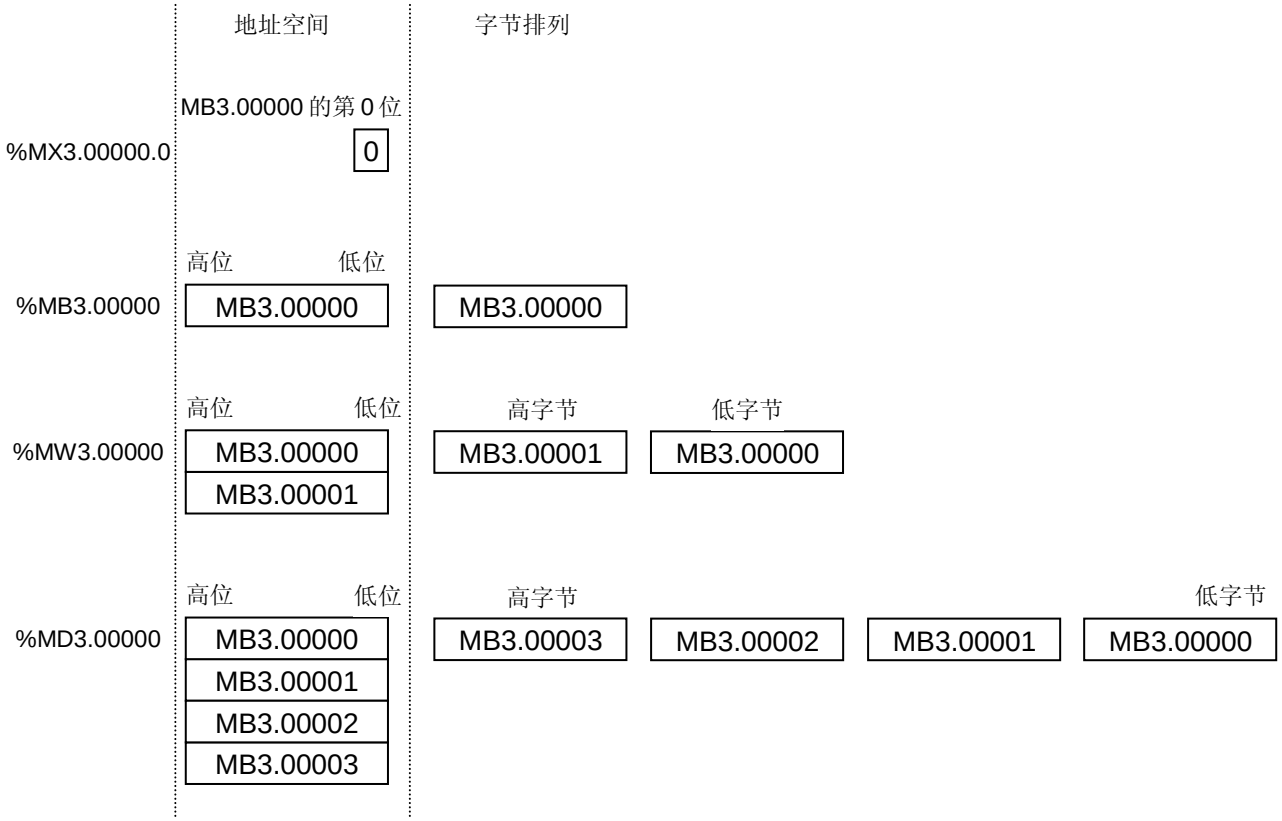
用户的数据可存放在 PLC 的不同存储器单元，每个单元都有唯一的地址。下表列出了不同长度的数据所能表示的数值范围

数值	布尔（X）	字节（B）	字（W）	双字（DW）
无符号整数	0 到 1	0 到 255	0 到 65535	0 到 4,294,967,295
有符号整数	-	-128 到+127	-32768 到 +32767	-2,147,483,648到+2,147,483,647
实数 IEEE 32 位浮点数	-	-	-	+1.175495E--38 到+3.402823E+38（正数） --1.175495E--38 到-3.402823E+38（负数）

对数据的存取必须指定地址，地址以%开头，后续是位置前缀，大小前缀，用整数表示字节地址，用小数点“.”加整数表示位。如%IX0.0 表示输入映射区第 0 个字节的第 0 个位，下表是数据的地址特性

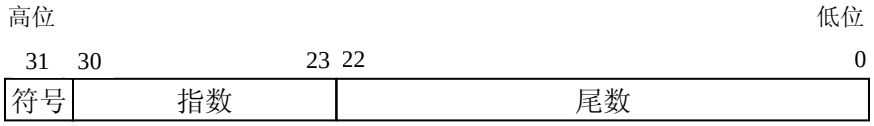
序号	前缀		定义	约定数据类型
1	位置前缀	I	输入映射区	
2		Q	输出映射区	
3		M	中间变量映射区	
4	大小前缀	X	位	BOOL
5		无	位	BOOL
6		B	字节（8 位）	BYTE
7		W	字（16 位）	WORD
8		D	双字（32 位）	DWORD
9		L	长型（64 位）	LREAL

字节、字和双字之间的地址关系是一个双字包含两个字，或者说包含四个字节，下面以 MX3.00000.0、MB3.00000、MW3.00000 和 MD3.00000 为例说明字节、字和双字之间的地址关系和数据排列



如一个十六进制数 16#1234 存放在%MW3.00000 中，则 16#34 存放在%MB3.00000 中，16#12 存放在%MB3.00001 中。程序中若对位操作，则会影响到位所在字节、字和双字，反过来也一样。

实数用 32 位浮点数表示，所以实数按照双字长度来存取，如下图。



变量地址的示例

%IX1.3 或%I1.3	表示数字量输入映射区中第 1 个字节中的第 3 位
%IW64	表示第一个模拟量输入
%QX0.0 或%Q0.0	表示数字量输出映射区中第 0 个字节中的第 0 位
%MD3.4	表示中间变量区中第 4 个字节开始的 1 个双字
%MX3.0.0	表示中间变量区中第 0 个字节的第 0 位

注：(5.3 版本的 1 个字节只能定义 1 个位，如%MX3.0.0，5.35 版本的一个字节可以定义 8 个位变量如，%MX3.0.0~%MX3.0.0)

PLC 支持常量值的输入，常量可以是二进制数、十进制数、十六进制数、字符串、ASCII 码或者实数，输入格式见下表。

数制	格式	举例
二进制	2#____	2#1110
十进制	____	52

十六进制	16#____	16#A8
字符串	‘____’	‘MYPLC’
ASCII 码	ASCII 码值	16#30
实数	REAL#____	REAL#3.14

2.3 T9 系列 PLC 保存数据

T9 系列 PLC 具有 2M 用户程序存储区、2M 用户数据数据存储区。

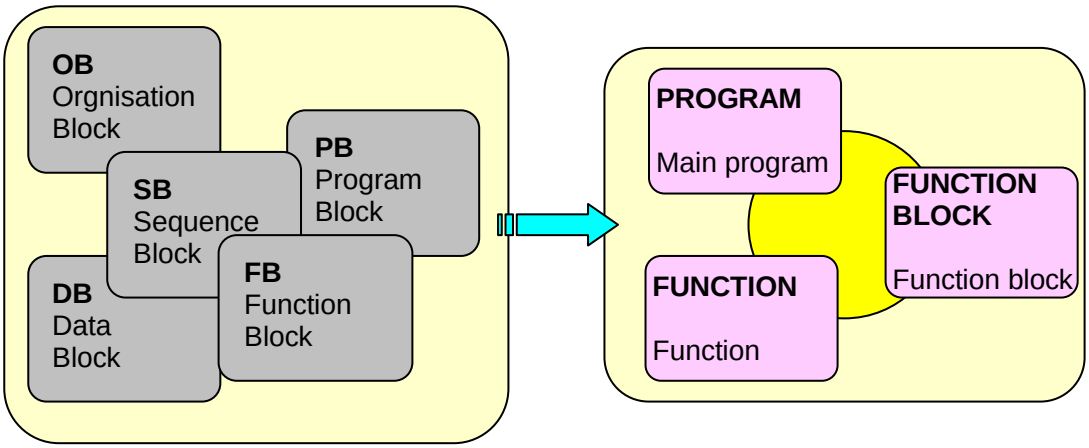
保持寄存器，在保持寄存器中，一部分是能够掉电保持的，PLC 断电后，依靠电池能保存这里的数据。

永久存储器，用来储存 PLC 的操作系统、用户程序、保持数据、系统块等。

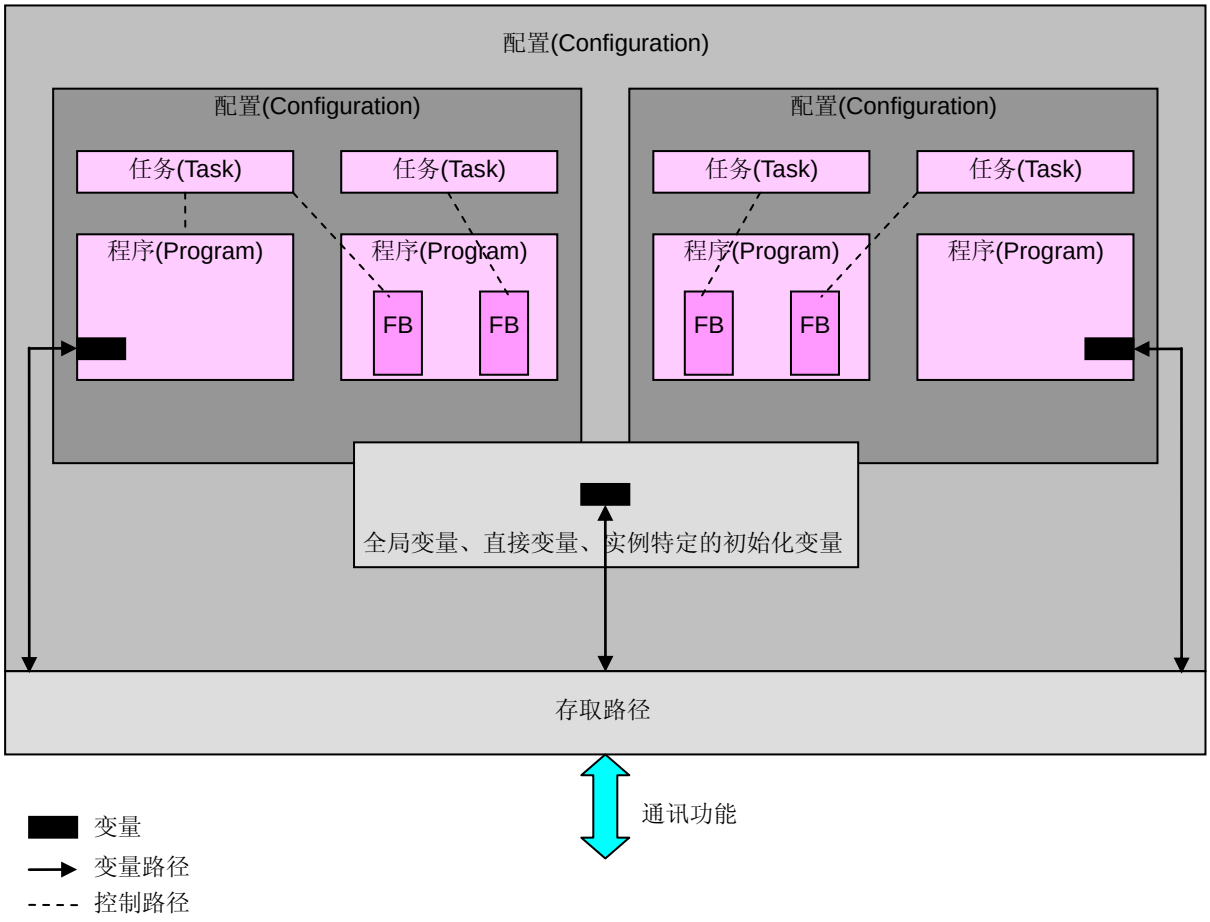
可通过用户程序向 PLC 中写入.txt、.xls 等文档文件，储存在用户数据存储区，也可以通过 ftp 工具将写入的文件复制、剪切和删除。

3. 编程模型

T9 系列 PLC 使用 MULTIPROG 编程软件，其编程语言和程序结构符合 IEC 61131-3 编程系统，在 IEC 61131-3 中，程序和工程的建立是在程序组织单元（POU）中完成的，程序组织单元 POU 包括程序（PROGRAM）、功能块（FUNCTION BLOCK）和功能（FUNCTION）共三部分，代替了传统 PLC 编程语言的 OB、PB、DB、SB 和 FB 五种功能块，如下图，使得编程变得更加高效，更加简洁。



IEC 61131-3 编程的软件模型用分层结构表示，如下图，软件模型描述了各部分之间的相互关系，包括配置、资源、任务、程序组织单元、全局变量、I/O 配置等。编程过程中可将一个复杂的程序分解为若干个小程序或模块，也可将多个独立的程序同时下载、运行，或将程序分成多个任务去执行，提高了程序的模块化和运行效率。



工程树包括“硬件”和“工程”两部分，分别对应所建立工程的硬件和软件部分。

3.1 硬件

打开已经建立的工程，在左侧的“工程树”窗口，点击“硬件”选项卡，可以看到“物理硬件”，“物理硬件”能够显示软件模型的结构，用户可对各层进行查看、设置。

3.1.1 物理硬件

“物理硬件”树能够反映符合 IEC 61131-3 的程序结构，是整个工程的全部配置文件，负责管理它的下一层——“配置”，目前，MULTIPROG Express 版本仅支持一个“配置”，不能插入多个配置，但能删除“配置”，也可从另一个工程中拷贝一个“配置”过来。

3.1.2 配置

“配置”是软件模型中的第一层，在“物理硬件”的下一层，相当于可编程控制器系统，负责管理它的下一层——“资源”，目前，MULTIPROG Express 版本仅支持一个“资源”，不能插入多个资源，但可以删除“资源”，也可从另一个工程中拷贝一个“资源”过来。

可编程控制器的类型，可通过右击“配置”，选择属性查看，在 PLC/处理选项卡中的 PLC 类型下拉列表中为“eCLR”，不可更改。

3.1.3 资源

“资源”是软件模型中的第二层，在“配置”的下一层，相当于可编程控制器的处理器，负责管理它的下一层——“Tasks”、“Global_Variables”和“IO_Configuration”，这三个均不可删除。

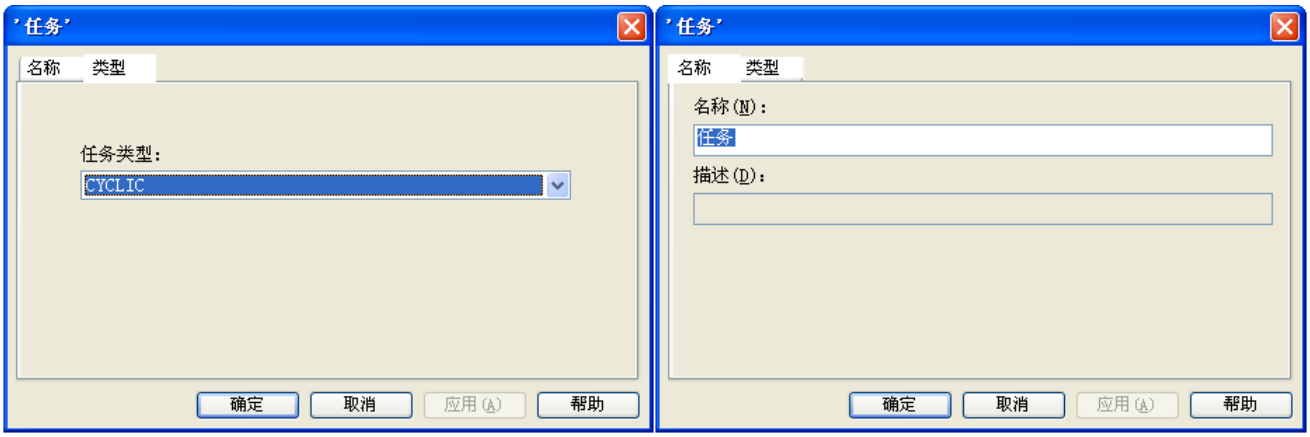
处理器的类型，可通过右击“资源”，选择属性查看，在 PLC/处理选项卡中的处理器类型下拉列表中选择“ARM_LE_GCC3”，当没有连接 PLC 只是仿真时，可选择“eCLR_Simulation”。

处理器的设置，可通过右击“资源”，选择设置查看，可以看到可编程控制器的通讯协议、IP 地址、处理器版本等。

3.1.4 Tasks

“Tasks”是软件模型中的第三层，在“资源”的下一层，“Tasks”下可插入多个任务，可以是循环扫描的，也可是周期扫描的，如果插入的任务是周期扫描的，可设置不同的扫描周期和优先级，这是 MULTIPROG 的多任务特性。

在创建一个工程时，MULTIPROG 自动声明了一个“Tasks”和这个目录下的第一个任务，用户可以右击“Tasks”插入一个新的任务，或从另一个工程中拷贝一个任务过来，插入的任务类型默认为 DEFAULT，即循环扫描，也可以选择 CYCLIC，即周期扫描，选择 CYCLIC 后，需要设置时间间隔、优先权和监视定时时间。右击“任务”，在弹出的对话框中分别选择“属性”和“设置”，可查看修改任务的类型和扫描周期，如下图。



插入新的“任务”后，右击新插入的任务名，选择“程序实例”，指定程序实例名和实例类型，实例类型是在 POU 中插入的程序（PROGRAM），有关实例类型将在 3.2 工程中介绍。

一个工程中，只允许有一个 DEFAULT 类型的任务，其它都默认为 CYCLIC 类型任务；在一个任务下，可插入多个程序实例，多个程序实例的执行顺序按照它们出现在任务下的顺序来执行。

3.1.5 Global_Variables——全局变量

“Global_Variables”是软件模型中的第三层，在“资源”的下一层，与“Tasks”并列，Global_Variables 不可复制、粘贴。全局变量是一个变量表，包括 MULTIPROG 提供的系统变量和用户建立的变量，用户建立的变量只有指定为 VAR_GLOBAL 时，才会出现在这个表中。

下表是腾控 PLC 内置的全局变量（VAR_GLOBAL），直接使用变量名即可：

变量名	类型	描述	地址
PLC_SYS_TICK_CNT	DINT	PLC 运行后节拍数	%MD1.0
PLC_TASK_DEFINED	INT	用户创建的工程文件中的任务数量	%MW1.4
PLCMODE_ON	BOOL	不可用	%MX1.2016.0
PLCMODE_LOADING	BOOL	不可用	%MX1.2017.0
PLCMODE_STOP	BOOL	PLC 停止时为 1，运行后为 0	%MX1.6.0
PLCMODE_RUN	BOOL	PLC 上电并运行后，一直为 1，停止时为 0	%MX1.7.0
PLCMODE_HALT	BOOL	不可用	%MX1.8.0
PLC_TICKS_PER_SEC	DINT	每秒钟的节拍数	%MD1.2000
PLC_MAX_ERRORS	DINT	PLC 的最大错误数量	%MD1.2004
PLC_ERRORS	DINT	PLC 错误	%MD1.2008
PLC_TASK_AVAILABLE	INT	PLC 的有效任务	%MW1.2012
PLC_SYSTASK_AVAILABLE	INT	PLC 的有效系统任务	%MW1.2016
PLCDEBUG_FORCE	BOOL	不可用	%MX1.2018.0
PLCDEBUG_BPSET	BOOL	不可用	%MX1.2019.0
PLCDEBUG_POWERFLOW	BOOL	不可用	%MX1.2020.0

3.1.6 IO_Configuration——IO 配置

“IO_Configuration”是软件模型中的第三层，在“资源”的下一层，与“Tasks”、“Global_Variables”并列，IO_Configuration 不可复制、粘贴。

双击“IO_Configuration”，打开 I/O 配置对话框，它用于编辑 I/O 组态工作单，包括了 INPUT（输入）、OUTPUT（输出）、VARCONF 的属性设置，用户只需设置 INPUT、OUTPUT 即可，在 INPUT 中，有 INPUT 的名称，默认“in”，所属的任务，逻辑起始地址，板卡/IO 模块的驱动程序参数，在驱动程序参数中，用户需要指定驱动程序名称，默认为 DUMMYIO，需要改为 SampleIO。

程序执行时，PLC 通过 I/O 来接收来自现场设备的信号、发送控制命令到现场设备，所以用户必须指定逻辑起始地址，驱动程序名称是指定 I/O 的驱动，必须修改为腾控的驱动程序名——SampleIO。

3.2 工程

工程包含库、数据类型、逻辑 POU 三个部分，组成一个完整而又强大的程序。

3.2.1 库

库提供了功能块、功能、程序和数据类型。在插入一个库之后，用户可以像使用 IEC 功能块一样使用库中所带的功能、功能块。右击“库”，可以插入“用户库”和“固件库”。这两种库不是

必须的，用户要根据自己程序的需要选择插入。

用户库是用户创建的其他工程，用户库的文件扩展名为*.mwt。

固件库是特殊功能的功能、功能块，需要用户单独插入到工程中，固件库的文件扩展名为*.fwl，固件库的默认存放目录是C:\Documents and Settings\All Users\Application Data\KW-Software\MULTIPROG Express\5_35_519\plc\FW_LIB，用户需要将固件库解压后将整个文件夹粘贴在这个目录下。

用户在工程树目录中右击“库”，选择“插入”->“固件库”，在弹出对话框中选择需要插入的固件库即可。

插入固件库后，在右侧的编辑向导中的下拉列表中会出现固件库的名称，选择这个名称会列出所插入的固件库中带有的功能、功能块。

3.2.2 数据类型

如果用户要定义自己的数据类型(如：数组、结构，等等)，则这些数据类型必须要在“数据类型”中声明。右击“数据类型”，选择插入“数据类型(T)”，指定数据类型工作单的名称，双击生成的数据类型工作单，进入编辑区域，键入以下字符

```
TYPE
    DATA1: ARRAY [1..100] OF INT;
END_TYPE
```

以上代码定义了一个包含 100 个 INT 型变量的数组，数组名 DATA1。

3.2.3 逻辑 POU

程序组织单元 POU 是 PLC 程序的语言元素。它们是包含了程序代码的小的、独立的软件单位。POU 的名称在工程内必须是唯一的，右击“逻辑 POU”可插入以下三种程序组织单元：

- 程序（PROGRAM）
- 功能块（FUNCTION BLOCK）
- 功能（FUNCTION）

每个 POU 都由两个不同部分组成：变量工作单和代码本体，在变量工作单中是这个 POU 中出现的所有变量。一个 POU 的代码工作单是用户用 IL、ST、FBD、LD、SFC 五种编程语言中的一种编写的，其中 IL 是指令表编程语言（Instruction List），ST 是结构化文本编程语言（Structured Text），FBD 是功能块图编程语言（Function Block Diagram），LD 是梯形图编程语言（Ladder Diagram），SFC 是顺序功能表编程语言（Sequential Function Chart）。

3.2.3.1 功能

“功能”，缩写为 FU，是带有多个输入和一个输出的程序组织单元 POU，类似于高级编程语言中的函数，“功能”的返回值可以是 BOOL、INT 等简单数据类型，一个“功能”内部可以调用另外的“功能”，但不能调用“功能块”或“程序”，不允许递归调用。

声明一个“功能”时，必须在这个“功能”的变量工作单中声明输入输出变量、中间变量、外部变量。

MULTIPROG 支持的功能列表：

- 类型转换功能，如 INT_TO_REAL

- 数值功能，如：ABS 和 LOG
- 标准算术运算功能，如：ADD 和 MUL
- 位串功能，如：AND 和 SHL
- 选择和比较功能，如：SEL 和 GE
- 字符串功能，如：RIGHT 和 INSERT
- 时间数据类型功能，如带有 TIME 数据类型的 SUB

3.2.3.2 功能块

“功能块”，缩写为 FB，是带有多个输入和多个输出的程序组织单元 POU，“功能块”内可以调用另外的“功能块”或“功能”，但不能调用“程序”，不允许递归调用。

所有的“功能块”(IEC 定义的，库件库 FB 和用户定义的 FB)可以很容易地被插入到用户的“功能块”或“程序”中。声明一个“功能块”时，必须在这个“功能块”的变量工作单中声明输入输出变量、中间变量、外部变量。

MULTIPROG 支持的功能块列表：

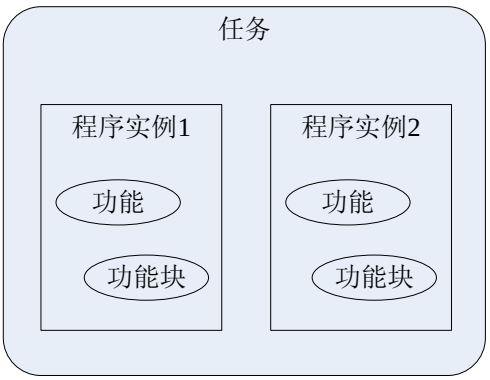
- 双稳态元素，如 SR 和 RS
- 边沿检测功能块，如：R_TRIG 和 F_TRIG
- 计数器，如：CTU 和 CTD
- 定时器功能块，如：TON 和 TOF

3.2.3.3 程序

“程序”是包含了功能、功能块的一个程序代码组合，“程序”的行为和用途类似于功能块，它可以具有输入和输出参数，可以具有内部存储区，但不允许递归调用。

在创建一个工程时，MULTIPROG 自动声明了一个“程序”，声明一个新的“程序”时，MULTIPROG 同时生成这个“程序”的变量工作单（双击这个新声明的程序，然后点击变量工作单即可进入这个程序的变量工作单），并把它加载到 Tasks 目录下的第一个任务中，可通过剪切粘贴把这个程序放到其他任务中。如前所述，在 Tasks->任务下插入一个程序实例时，要输入程序名和程序实例类型，这个程序实例类型是在程序->逻辑 POU 下插入的“程序”，这样，可在多个任务中插入多个程序实例，这些程序实例的名字可以不相同，但可以是同一个程序实例类型，既是说，一个程序实例类型可以在多个任务中被执行。

“程序”一定要被关联到任务上，下图显示了一个带有两个程序的默认任务：



4. 数据类型

程序包括代码和数据两部分，代码可以是 IL、ST、FBD、LD、SFC 五种编程语言中的任何一种，或几种语言的组合，数据分为三种类型：基本数据类型、派生数据类型和用户自定义数据类型，数据必须通过变量的方式存在，数据类型决定了变量的格式、位数、可能值的范围和初始值。

变量要在变量工作单中声明。

4.1 基本数据类型

数据类型	描述	位长度	范围	默认初始值
BOOL	布尔	1	0 或 1，即真或假	0
SINT	短整型	8	-128…127	0
INT	整型	16	-32,768…32,767	0
DINT	双整型	32	-2,147,483,648…2,147,483,647	0
USINT	无符号短整型	8	0…255	0
UINT	无符号整型	16	0…65,535	0
UDINT	无符号双整型	32	0…4,294,967,295	0
REAL	实数	32	-3.402823466 E+38 … -1.175494351 E-38 以及 +1.175494351 E-38 … +3.402823466 E+38 注：科学计数法，7 位小数	0.0
LREAL	长实数	64	-1.798 E+308 …-2.225 E-308 以及 +2.225 E-308 …+1.798 E+308 注：科学计数法，7 位小数	0.0
TIME	时间	32	0... 4,294,967,295 毫秒	T#0S
BYTE	字节	8	0...255(16#00...16#FF)	0
WORD	字	16	0...65,535(16#00...16#FFFF)	0
DWORD	双字	32	0...4,294,967,295 (16#00....16#FFFFFFFF)	0

4.2 类属数据类型

类属数据类型是把基本数据类型分级分组，以 ANY 作为数据类型的前缀，如，ANY_INT 表示包括 SINT、INT、DINT、USINT、UINT 和 UDINT 等全部整型数据。如果一个功能块的输入或输出与 ANY_INT 相连，则意味着这个功能块可以处理 SINT、INT、DINT、USINT、UINT 和 UDINT 等整型数据的变量。

类属数据类型以如下形式组织：

<table>
<tr><td colspan="5">ANY</td></tr>
<tr><td colspan="2">ANY_NUM</td><td>ANY_BIT</td><td>STRING</td><td>TIME</td></tr>
<tr><td>ANY_REAL</td><td>ANY_INT</td><td rowspan="2">BOOL
BYTE
WORD
DWORD</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>REAL
LREAL</td><td>SINT USINT
INT UINT
DINT UDINT</td></tr>
</table>

4.3 用户自定义数据类型

用户自定义的数据类型必须在工程->数据类型中插入，见软件模型一章。

用户自定义数据类型必须以 TYPE ...END_TYPE 声明块来完成，声明块中间部分是定义的衍生数据，衍生数据类型可以是结构，或者是数组。

数组

数组是一个单一数据类型对象的集合，同基本数据一样它具有唯一的名字，其中单个的对象并没有被命名，但用户可以通过它在数组中的位置对它进行访问，声明一个数组的例子如下：

```
TYPE
    graph : ARRAY [0..23] OF INT;
END_TYPE
```

注：ARRAY 型数组 graph 的最低字节是 graph[0]

结构

结构是多个不同数据类型对象的集合，同基本数据一样它有唯一的名字，结构的成员是基本数据类型或数组类型，也可以是另一个结构，或者嵌套。声明一个结构的例子如下：

```
TYPE
    machine :
    STRUCT
        x_pos : INT;
        y_pos : INT;
        depth : INT;
        rpm   : INT;
    END_STRUCT;
END_TYPE
```

字符串

字符串是由多个字符组成的有限序列，每个字符占用一个字节，字符串的数据类型是 STRING，声明一个字符串时，其长度在数据类型后面的括号内设定，声明一个字符串的例子：

```
TYPE
```

STRING10 : STRING(10)
END_TYPE
在这个例子中，字符串的长度是 10,即 STRING10 是一个包含 10 个字符的字符串。
最短的字符串长度为 1，最长的字符串长度为 32766。

4.4 常量数据的表示

数据类型	描述	位长度	表示方法举例
BOOL	布尔	1	BOOL#0
SINT	短整型	8	SINT#-128
INT	整型	16	INT#-32,768
DINT	双整型	32	DINT#-2,147,483,648
USINT	无符号短整型	8	USINT#255
UINT	无符号整型	16	UINT#65,535
UDINT	无符号双整型	32	UDINT#4,294,967,295
REAL	实数	32	REAL#3.1415629
LREAL	长实数	64	LREAL#3.1415629
TIME	时间	32	T#10MS 、 T#10S 、 T#10M 、 T#10H、T#10D、T#1D_10H
DATE	日期		D#2011-07-24
TIME OF DATE	时刻		TOD#15:23:45,55
TIME AND DATE	日期和时刻		DT#2011-07-24-15:23:45,55
BYTE	字节	8	BYTE#16#FF
WORD	字	16	WORD#16#FFFF)
DWORD	双字	32	DWORD#16#FFFFFFFF)
STRING	字符串		‘A’

5. 指令集

PLC 指令是封装好了程序块，每个指令能够完成一定的逻辑、运算操作，指令集是 PLC 指令的集合。

在 MULTIPROG 编程中，为编程方便，将这些指令被分配到几个不同的功能区（或库）中，在 MULTIPROG 中的编辑向导中可分别罗列出这些功能块，本章将按照这几个不同功能区（或库）的划分，介绍上述指令

5.1 功能
5.2 功能块
5.3 类型转换 FU
5.4 字符串 FU
5.5 BIT_UTIL
5.6 PROCONOS
此外，腾控针对自己的 PLC 特点提供了特殊功能的指令
5.7 TCNET 通讯协议
5.8 高速计数
5.9 文件读写

注：

1. IL 编程语言的指令用法说明中，经常用到 LD 和 ST 操作符，它们的用法如下
   LD IN (* LD 表示将变量 IN 装入累加器 *)
   ABS (* ABS 表示将累加器的值求绝对值，结果再送累加器 *)
   ST OUT (* ST 表示将累加器的值赋给变量 OUT *)
2. ST 编程语言的指令用法说明中，“:=”为赋值操作符。

5.1 功能

功能是带有多个输入参数和一个输出参数的程序组织单元 POU，它们没有任何内部存储器，调用带有相同值的功能总是返回相同的结果。返回值是单变量，或多元素变量，如数组或结构。功能的缩写为 FU。

MULTIPROG 编程过程中，可以使用以下功能

- 类型转换功能
- 数值功能
- 算数运算功能
- 按位布尔运算功能
- 位串功能
- 选择运算功能
- 比较运算功能
- 字符串功能

功能中包含的指令（在编辑向导中，从下拉列表选择“功能”）

ABS	DIV_T_R	MAX	ROL
ACOS	EQ	MIN	ROR
ADD	EXP	MOD	SEL
ADD_T_T	EXPT	MOVE	SHL
AND	GE	MUL	SHR
ASIN	GT	MUL_T_AI	SIN
ATAN	LE	MUL_T_AN	SQRT
COS	LIMIT	MUL_T_R	SUB
DIV	LN	NE	SUB_T_T
DIV_T_AI	LOG	NOT	TAN
DIV_T_AN	LT	OR	XOR

在一下的 LD 和 FBD 的指令说明中，仅当输入引脚 EN 为 1 时，该指令激活；当该指令被顺利执行后，输出引脚 ENO 置 1，否则引脚 ENO 置 0。

5.1.1 绝对值——ABS 指令

功能

ABS 指令用于求负数的绝对值。

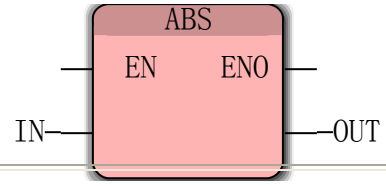
用法

IL 编程语言

```
LD IN
ABS
ST OUT
```

ST 编程语言

LD、 FBD 编程语言



OUT := ABS(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

ABS 指令处理的数据类型

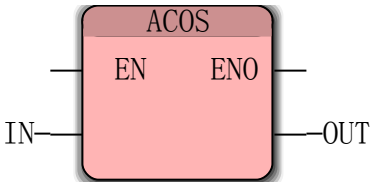
输入和输出	数据类型	描述
IN	ANY_NUM	输入
OUT	ANY_NUM	输出

5.1.2 反余弦——ACOS 指令

功能

ACOS 指令用于求输入值的反余弦。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN ACOS ST OUT	
ST 编程语言	
OUT := ACOS(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

ACOS 指令处理的数据类型

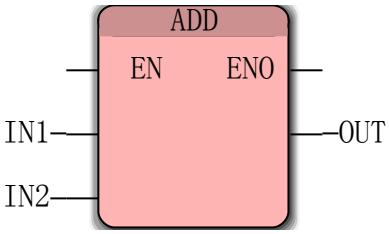
输入和输出	数据类型	描述
IN	REAL	输入
OUT	REAL	输出，以弧度表示角度

5.1.3 加法——ADD 指令

功能

ADD 指令用于求两个数据的和。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 ADD IN2 ST OUT	
ST 编程语言	
OUT := IN1 + IN2 ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量

ADD 指令处理的数据类型

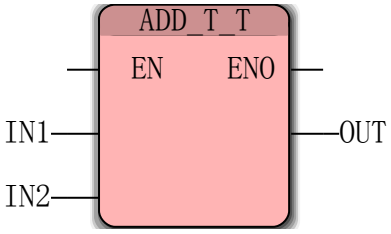
输入和输出	数据类型	描述
IN1	ANY_NUM	被加数
IN2	ANY_NUM	加数
OUT	ANY_NUM	和，OUT = IN1 + IN2

5.1.4 时间加法——ADD_T_T 指令

功能

ADD_T_T 指令用于求两个时间数据的和。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 ADD_T_T IN2 ST OUT	
ST 编程语言	
OUT := ADD_T_T(IN1 , IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

ADD_T_T 指令处理的数据类型

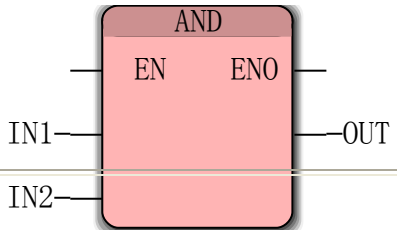
输入和输出	数据类型	描述
IN1	TIME	加数
IN2	TIME	被加数
OUT	TIME	和，OUT = IN1 + IN2

5.1.5 逻辑与——AND 指令

功能

AND 指令用于两个数据的逻辑与运算。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 AND IN2 ST OUT	
ST 编程语言	

OUT := IN1 & IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

AND 指令处理的数据类型

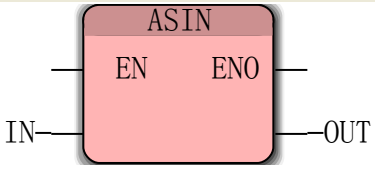
输入和输出	数据类型	描述
IN1	ANY_BIT	数据 1
IN2	ANY_BIT	数据 2
OUT	ANY_BIT	结果 IN1=0，IN2=0，OUT=0； IN1=0，IN2=1，OUT=0； IN1=1，IN2=0，OUT=0； IN1=1，IN2=1，OUT=1；

5.1.6 反正弦——ASIN 指令

功能

ASIN 指令用于求反正弦。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN ASIN ST OUT	
ST 编程语言	
OUT := ASIN(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

ASIN 指令处理的数据类型

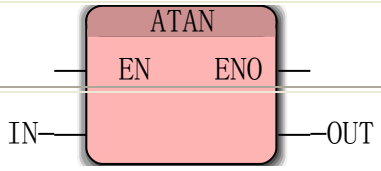
输入和输出	数据类型	描述
IN	REAL	输入
OUT	REAL	输出，以弧度表示角度

5.1.7 反正切——ATAN 指令

功能

ATAN 指令用于求反正切。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN ATAN	

ST OUT	
ST 编程语言	
OUT := ATAN(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

ATAN 指令处理的数据类型

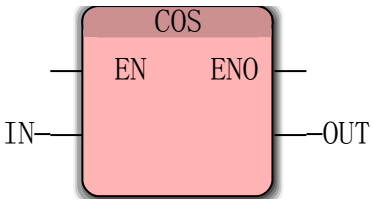
输入和输出	数据类型	描述
IN	REAL	输入
OUT	REAL	输出，以弧度表示角度

5.1.8 余弦——COS 指令

功能

COS 指令用于求输入值的余弦。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN COS ST OUT	
ST 编程语言	
OUT := COS(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

COS 指令处理的数据类型

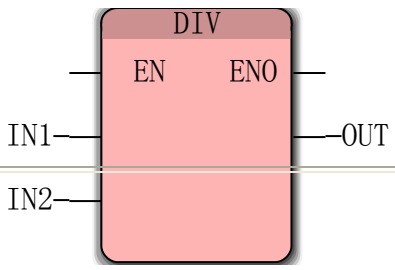
输入和输出	数据类型	描述
IN	REAL	输入，以弧度表示角度
OUT	REAL	输出

5.1.9 除法——DIV 指令

功能

DIV 指令用于除法运算。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 DIV IN2 ST OUT	
ST 编程语言	

OUT := IN1 / IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

DIV 指令处理的数据类型

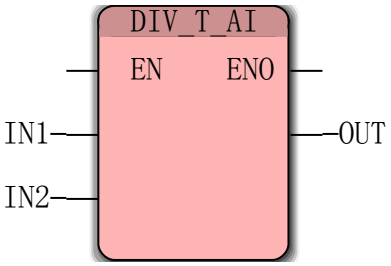
输入和输出	数据类型	描述
IN1	ANY_NUM	被除数
IN2	ANY_NUM	除数
OUT	ANY_NUM	商

5.1.10 除法(时间除以整数)——DIV_T_AI 指令

功能

DIV_T_AI 指令用于时间除以整数运算。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 DIV_T_AI IN2 ST OUT	
ST 编程语言	
OUT := DIV_T_AI(IN1, IN2);	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

DIV_T_AI 指令处理的数据类型

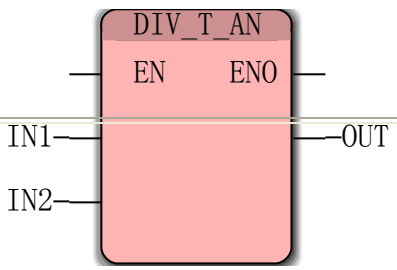
输入和输出	数据类型	描述
IN1	TIME	被除数
IN2	ANY_INT	除数
OUT	TIME	商

5.1.11 除法(时间除以整数、实数)——DIV_T_AN 指令

功能

DIV_T_AN 指令用于时间除以整数或实数运算。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 DIV_T_AN IN2 ST OUT	

ST 编程语言	
OUT := DIV_T_AN(IN1, IN2);	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

DIV_T_AN 指令处理的数据类型

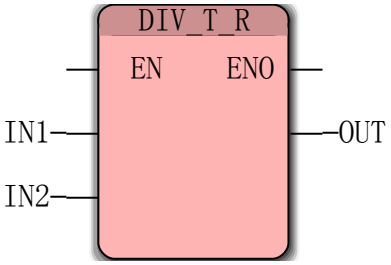
输入和输出	数据类型	描述
IN1	TIME	被除数
IN2	ANY_NUM	除数
OUT	TIME	商

5.1.12 除法(时间除以实数)——DIV_T_R 指令

功能

DIV_T_R 指令用于时间除以实数运算。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 DIV_T_R IN2 ST OUT	
ST 编程语言	
OUT := DIV_T_R(IN1, IN2);	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

DIV_T_R 指令处理的数据类型

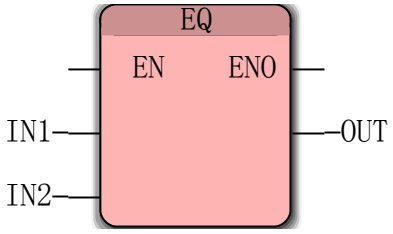
输入和输出	数据类型	描述
IN1	TIME	被除数
IN2	REAL	除数
OUT	TIME	商

5.1.13 等于——EQ 指令

功能

EQ 指令用于判断两个数是否相等。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 EQ IN2 ST OUT	
ST 编程语言	
OUT := IN1 = IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

EQ 指令处理的数据类型

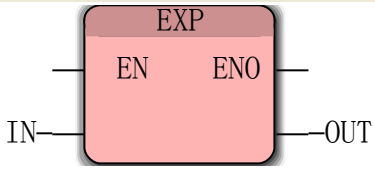
输入和输出	数据类型	描述
IN1	ELEMENTARY	数据 1
IN2	ELEMENTARY	数据 2
OUT	BOOL	输出 两数相等，为 TRUE； 两数不相等，为 FALSE

5.1.14 自然数 e 的指数函数——EXP 指令

功能

EXP 指令用于计算自然常数 e 的 x 次方，在这里 x 是输入，其中 e≈2.718281828。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN EXP ST OUT	
ST 编程语言	
OUT := EXP(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

EXP 指令处理的数据类型

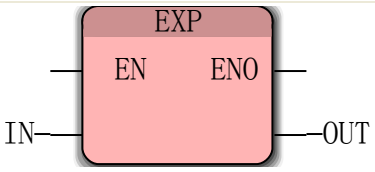
输入和输出	数据类型	描述
IN	REAL	指数
OUT	REAL	结果，e 的 IN 次方

5.1.15 幂(x 的 y 次方)——EXPT 指令

功能

EXPT 指令用于计算 x 的 y 次方，在这里 x 是第一个输入，y 是第二个输入。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 EXP IN2 ST OUT	
ST 编程语言	
OUT := EXPT(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

EXPT 指令处理的数据类型

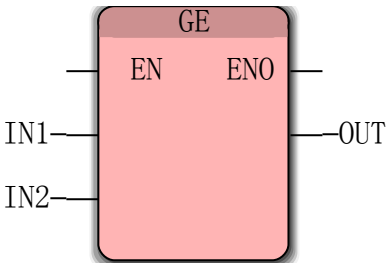
输入和输出	数据类型	描述
IN1	ANY_REAL	基数
IN2	ANY_NUM	指数
OUT	REAL	结果，IN1 的 IN2 次方

5.1.16 大于等于——GE 指令

功能

GE 指令用于比较两个数值的大小，当第一个输入大于或等于第二个时，输出为 1，其他为 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 GE IN2 ST OUT	
ST 编程语言	
OUT := IN1 >= IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

GE 指令处理的数据类型

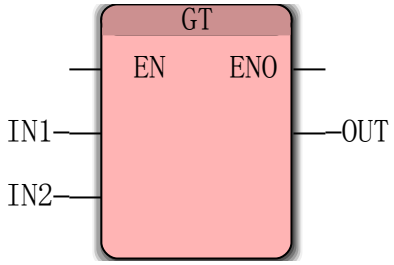
输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果，IN1 >= IN2 时，OUT 为 1

5.1.17 大于——GT 指令

功能

GT 指令用于比较两个数值的大小，当第一个输入大于第二个时，输出为 1，其他为 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 GT IN2 ST OUT	
ST 编程语言	
OUT := IN1 > IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

GT 指令处理的数据类型

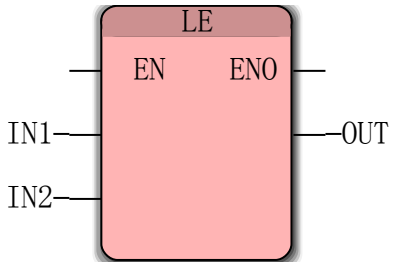
输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果，IN1 > IN2 时，OUT 为 1

5.1.18 小于等于——LE 指令

功能

LE 指令用于比较两个数值的大小，当第一个输入小于或等于第二个时，输出为 1，其他为 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 LE IN2 ST OUT	
ST 编程语言	
OUT := IN1 <= IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

LE 指令处理的数据类型

输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入

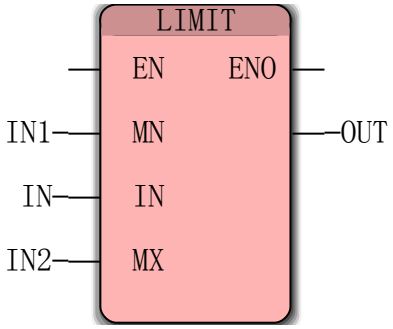
OUT	BOOL	结果 当 IN1 <= IN2 时，OUT 为 1; 当 IN1 > IN2 时，OUT 为 0;

5.1.19 极限选择——LIMIT 指令

功能

LIMIT 指令用于把输入值限定到由最大值和最小值所确定的区间。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 LIMIT IN, IN2 ST OUT	
ST 编程语言	
OUT := LIMIT (IN1, IN, IN2) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、IN1、IN2 和 OUT 或使用常量

LIMIT 指令处理的数据类型

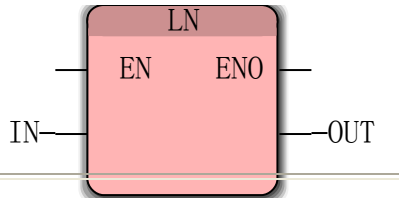
输入和输出	数据类型	描述
IN1	ANY_INT	最小值
IN	ANY_INT	输入值
IN2	ANY_INT	最大值
OUT	ANY	输出值 当 IN1 <= IN <= IN2 时，OUT = IN; 当 IN < IN1 时，OUT = IN1; 当 IN > IN2 时，OUT = IN2;

5.1.20 自然对数——LN 指令

功能

LN 指令用于计算输入的自然对数。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN LN ST OUT	
ST 编程语言	

OUT := LN(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

LN 指令处理的数据类型

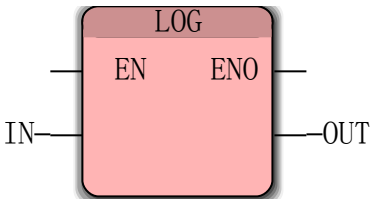
输入和输出	数据类型	描述
IN	REAL	输入值
OUT	REAL	结果，OUT = $LOG_{10}IN$

5.1.21 对数——LOG 指令

功能

LOG 指令用于计算输入的以 10 为底的对数。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN LOG ST OUT	
ST 编程语言	
OUT := LOG(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

LOG 指令处理的数据类型

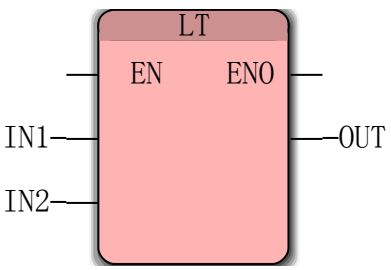
输入和输出	数据类型	描述
IN	REAL	输入值
OUT	REAL	结果，OUT = $LOG_{10}IN$ = lg(IN)

5.1.22 小于——LT 指令

功能

LT 指令用于比较两个数值的大小，当第一个输入小于第二个时，输出为 1，其他为 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 LT IN2 ST OUT	
ST 编程语言	
OUT := IN1 < IN2 ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量

LT 指令处理的数据类型

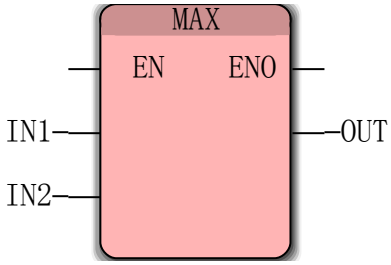
输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果 当 IN1 < IN2 时，OUT 为 1； 当 IN1 >= IN2 时，OUT 为 0；

5.1.23 最大值——MAX 指令

功能

MAX 指令用于确定两个数值的最大值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MAX IN2 ST OUT	
ST 编程语言	
OUT := MAX(IN1, IN2) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量

MAX 指令处理的数据类型

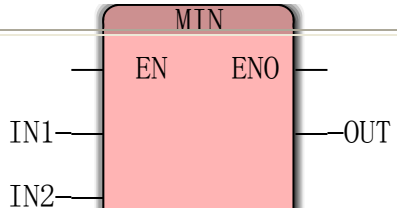
输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY	结果 当 IN1 <= IN2 时，OUT 为 IN2； 当 IN1 >= IN2 时，OUT 为 IN1；

5.1.24 最小值——MIN 指令

功能

MIN 指令用于确定两个数值的最小值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1	

MIN IN2 ST OUT	
ST 编程语言	
OUT := MIN(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

MIN 指令处理的数据类型

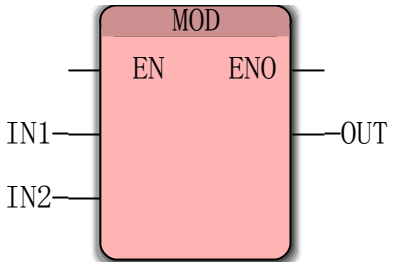
输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY	结果 当 IN1 <= IN2 时，OUT 为 IN1； 当 IN1 >= IN2 时，OUT 为 IN2；

5.1.25 取摸——MOD 指令

功能

MOD 指令用于确定两个数值相除后的余数。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MOD IN2 ST OUT	
ST 编程语言	
OUT := MOD(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

MOD 指令处理的数据类型

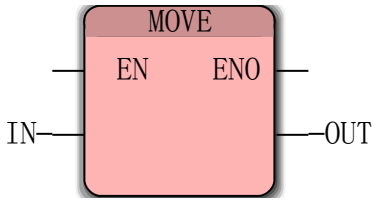
输入和输出	数据类型	描述
IN1	ANY_INT	被除数
IN2	ANY_INT	除数
OUT	ANY_INT	结果，IN1 除以 IN2 后的余数

5.1.26 赋值——MOVE 指令

功能

MOVE 指令用于将输入值赋给输出值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN MOVE ST OUT ST 编程语言 OUT := MOVE(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

MOVE 指令处理的数据类型

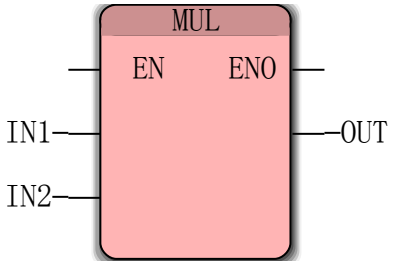
输入和输出	数据类型	描述
IN	ANY_NUM	输入值
OUT	ANY_NUM	输出值，OUT = IN

5.1.27 乘法——MUL 指令

功能

MUL 指令用于求两个数据的积。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MUL IN2 ST OUT ST 编程语言 OUT := IN1 * IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

MUL 指令处理的数据类型

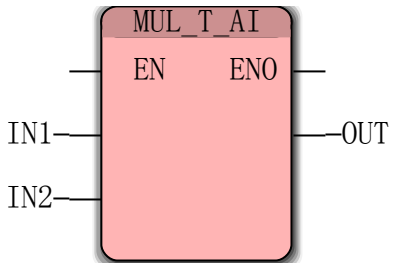
输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY_NUM	积，OUT = IN1 * IN2

5.1.28 乘法(时间乘以整数)——MUL_T_AI 指令

功能

MUL_T_AI 指令用于计算一个时间型数据与一个整型数据的积。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MUL_T_AI IN2 ST OUT ST 编程语言 OUT := MUL_T_AI(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

MUL_T_AI 指令处理的数据类型

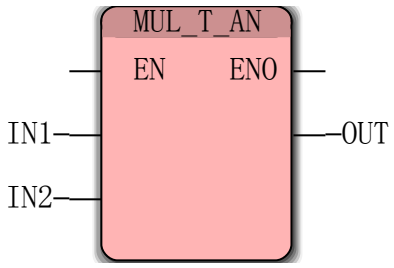
输入和输出	数据类型	描述
IN1	TIME	第一个输入，时间
IN2	ANY_INT	第二个输入，整数
OUT	TIME	积，OUT = IN1 * IN2

5.1.29 乘法(时间乘以整数、实数)——MUL_T_AN 指令

功能

MUL_T_AN 指令用于计算一个时间型数据与一个整数或浮点数的积。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MUL_T_AN IN2 ST OUT ST 编程语言 OUT := MUL_T_AN(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

MUL_T_AN 指令处理的数据类型

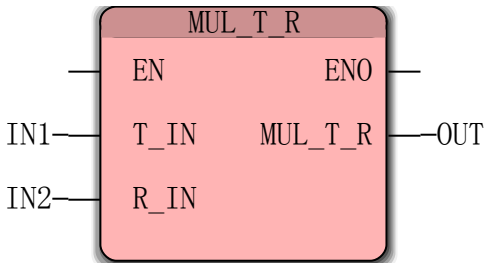
输入和输出	数据类型	描述
IN1	TIME	第一个输入，时间
IN2	ANY_NUM	第二个输入，整数或浮点数
OUT	TIME	积，OUT = IN1 * IN2

5.1.30 乘法(时间乘以实数)——MUL_T_R 指令

功能

MUL_T_R 指令用于计算一个时间型数据与一个浮点型数据的积。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MUL_T_R IN2 ST OUT **ST 编程语言** OUT := MUL_T_R(IN1, IN2) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量

MUL_T_R 指令处理的数据类型

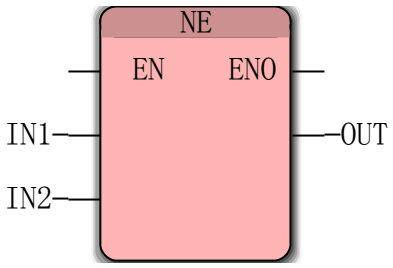
输入和输出	数据类型	描述
IN1（T_IN）	TIME	第一个输入，时间
IN2（R_IN）	ANY_NUM	第二个输入，浮点数
OUT（MUL_T_R）	TIME	积，OUT = IN1 * IN2

5.1.31 不等于——NE 指令

功能

NE 指令用于判断两个数值的大小关系，当第一个输入不等于第二个时，输出为 1，其他为 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 NE IN2 ST OUT **ST 编程语言** OUT := IN1 <> IN2 ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量

NE 指令处理的数据类型

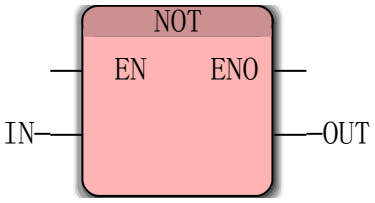
输入和输出	数据类型	描述
IN1	ANY	第一个输入
IN2	ANY	第二个输入
OUT	BOOL	结果 当 IN1 <> IN2 时，OUT 为 1； 当 IN1 = IN2 时，OUT 为 0；

5.1.32 逻辑非——NOT 指令

功能

NOT 指令用于将输入值按位取反，如 BYTE#2#11001100，经 NOT 运算得 BYTE#2#00110011。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN NOT ST OUT	
ST 编程语言	
OUT := NOT(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

NOT 指令处理的数据类型

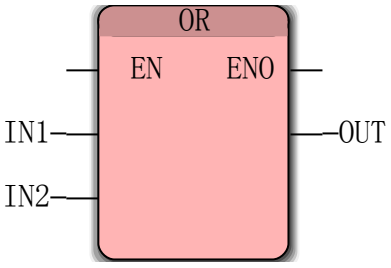
输入和输出	数据类型	描述
IN	ANY_BIT	输入
OUT	ANY_BIT	结果

5.1.33 逻辑或——OR 指令

功能

OR 指令用于将输入值按位做逻辑或运算。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 OR IN2 ST OUT	
ST 编程语言	
OUT := (IN1) OR (IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

OR 指令处理的数据类型

输入和输出	数据类型	描述
IN1	ANY_BIT	第一个输入
IN2	ANY_BIT	第二个输入
OUT	ANY_BIT	结果，逻辑或运算 IN1=0，IN2=0，OUT 为 0； IN1=0，IN2=1，OUT 为 1；

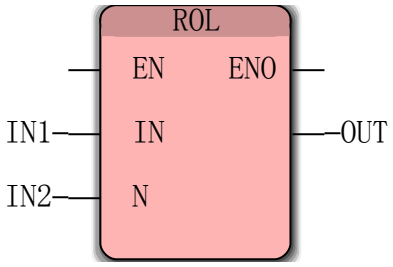
		IN1=1，IN2=0，OUT 为 1； IN1=1，IN2=1，OUT 为 1；

5.1.34 循环左移——ROL 指令

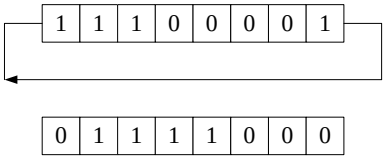
功能

ROL 指令用于将输入值按位循环左移。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 ROL IN2 ST OUT	
ST 编程语言	
OUT := ROL(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

ROL 指令处理的数据类型

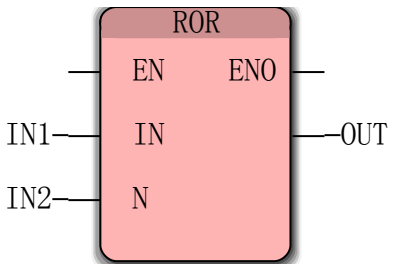
输入和输出	数据类型	描述
IN1	ANY_BIT	输入
IN2	ANY_INT	循环左移的位数
OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，左移一次。左移两位，如下 

5.1.35 循环右移——ROR 指令

功能

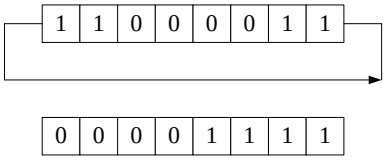
ROR 指令用于将输入值按位循环右移。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 ROR IN2 ST OUT	
ST 编程语言	
OUT := ROR(IN1, IN2) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量

ROR 指令处理的数据类型

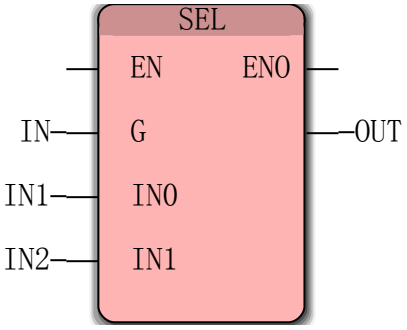
输入和输出	数据类型	描述
IN1（IN）	ANY_BIT	输入
IN2（N）	ANY_INT	循环右移的位数
OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，右移一次。右移两位，如下 

5.1.36 选择——SEL 指令

功能

SEL 指令用于根据一个布尔量的两个状态选择不同的输入值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN SEL IN1, IN2 ST OUT	
ST 编程语言	
OUT := SEL(IN, IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、IN1、IN2 和 OUT 或使用常量	

SEL 指令处理的数据类型

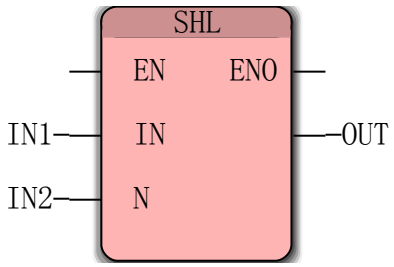
输入和输出	数据类型	描述
IN（G）	BOOL	选择输入端
IN1（IN0）	ANY	第一个输入
IN2（IN1）	ANY	第二个输入
OUT	ANY	结果 如果 IN=0，OUT = IN1； 如果 IN=1，OUT = IN2。

5.1.37 左移——SHL 指令

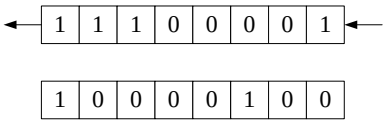
功能

SHL 指令用于将输入值按位左移，数据左端移出，右端补 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 SHL IN2 ST OUT	
ST 编程语言	
OUT := SHL(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

SHL 指令处理的数据类型

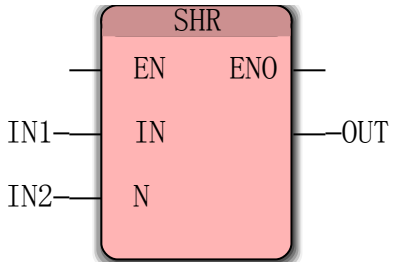
输入和输出	数据类型	描述
IN1（IN）	ANY_BIT	输入
IN2（N）	ANY_INT	左移的位数
OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，左移一次。左移两位，如下 

5.1.38 右移——SHR 指令

功能

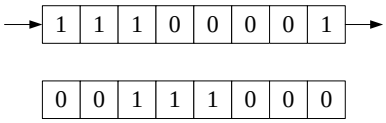
SHR 指令用于将输入值按位右移，数据右端移出，左端补 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 SHR IN2 ST OUT	
ST 编程语言	
OUT := SHR(IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

SHR 指令处理的数据类型

输入和输出	数据类型	描述
IN1（IN）	ANY_BIT	输入
IN2（N）	ANY_INT	右移的位数
OUT	ANY_BIT	结果，当引脚 IN 或 N 的值有变化时，右移一次。右移两位，如下

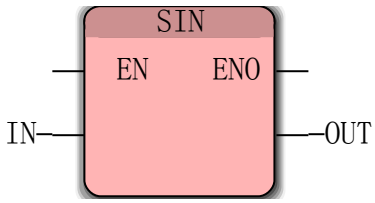
		

5.1.39 正弦——SIN 指令

功能

SIN 指令用于求输入值的正弦。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN SIN ST OUT	
ST 编程语言	
OUT := SIN(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

SIN 指令处理的数据类型

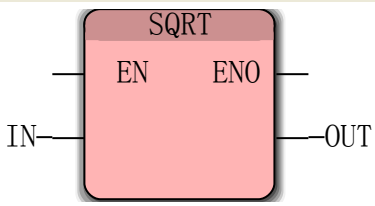
输入和输出	数据类型	描述
IN	REAL	输入，以弧度表示角度
OUT	REAL	输出

5.1.40 平方根——SQRT 指令

功能

SQRT 指令用于求输入值的平方根。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN SQRT ST OUT	
ST 编程语言	
OUT := SQRT(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

SQRT 指令处理的数据类型

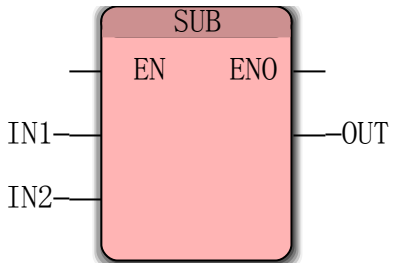
输入和输出	数据类型	描述
IN	REAL	输入
OUT	REAL	输出

5.1.41 减法——SUB 指令

功能

SUB 指令用于求两个输入值的差。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 SUB IN2 ST OUT	
ST 编程语言	
OUT := IN1 – IN2 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

SUB 指令处理的数据类型

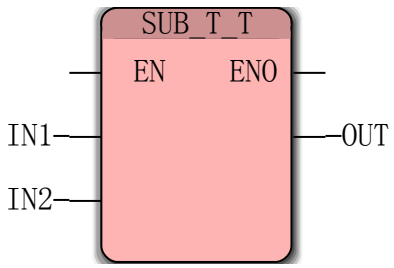
输入和输出	数据类型	描述
IN1	ANY_NUM	第一个输入
IN2	ANY_NUM	第二个输入
OUT	ANY_NUM	输出，OUT = IN1 – IN2

5.1.42 时间减法——SUB_T_T 指令

功能

SUB_T_T 指令用于求两个时间输入值的差。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 SUB IN2 ST OUT	
ST 编程语言	
OUT := SUB_T_T (IN1, IN2);	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

SUB_T_T 指令处理的数据类型

输入和输出	数据类型	描述
IN1	TIME	第一个输入
IN2	TIME	第二个输入

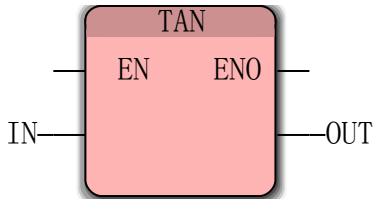
OUT	TIME	输出，OUT = IN1 – IN2

5.1.43 正切——TAN 指令

功能

TAN 指令用于求输入值的正切。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN TAN ST OUT	
ST 编程语言	
OUT := TAN(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

TAN 指令处理的数据类型

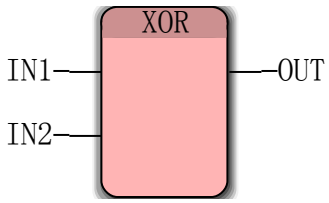
输入和输出	数据类型	描述
IN	REAL	输入，以弧度表示角度
OUT	REAL	输出

5.1.44 逻辑异或——XOR 指令

功能

XOR 指令用于将输入值按位做逻辑异或运算。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 XOR IN2 ST OUT	
ST 编程语言	
OUT := (IN1) XOR (IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

XOR 指令处理的数据类型

输入和输出	数据类型	描述
IN1	ANY_BIT	第一个输入
IN2	ANY_BIT	第二个输入
OUT	ANY_BIT	结果，逻辑异或运算 IN1=0，IN2=0，OUT 为 0；

		IN1=0，IN2=1，OUT 为 1； IN1=1，IN2=0，OUT 为 1； IN1=1，IN2=1，OUT 为 0；

5.2 功能块

功能块是带有多个输入和输出参数的程序组织单元 POU，它们有内部存储器，功能块的返回值取决于其内部存储单元的值。功能块的缩写为 FB。

与前述的功能不同，功能块必须被实例化，实例名称可以是默认的名称，也可更改默认名，实例名称在 POU 内必须是独一无二的。在 FBD 和 LD 编程中，这个实例名称出现在功能块上部。

MULTIPROG 编程过程中，可以使用以下功能块

- 双稳态功能块
- 脉冲边沿检测功能块
- 计数器功能块
- 定时器功能块

功能块中包含的指令（在编辑向导中，从下拉列表选择“功能块”）

CTD	CTU	CTUD	F_TRIG	R_TRIG	RS	SR	TOF	TON	TP

5.2.1 递减计数器——CTD 指令

功能

CTD 指令用于对输入端做递减计数。

LOAD 端为 FALSE 时，如果 CD 输入端出现一个上升沿，则 CV 端减 1。如果计数值 CV 达到了记数器的下限值 0，则在 Q 输出端发出 TRUE 信号，CTD 功能块停止记数。

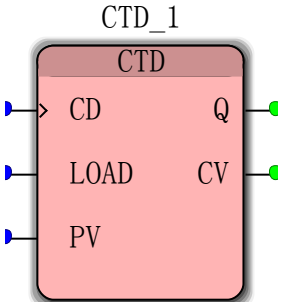
LOAD 端为 TRUE 时，计数器停止计数，并将 PV 输入端的数值赋给 CV 端。

用法

IL 编程语言

```
LD var1
ST CTD_1.CD
LD var2
ST CTD_1.LOAD
LD var3
ST CTD_1.PV
CAL CTD_1
LD CTD_1.Q
ST var4
LD CTD_1.CV
ST var5
```

LD、 FBD 编程语言



ST 编程语言

```
CTD_1(CD := var1, LOAD := var2, PV := var3) ;
var4 := CTD_1.Q ;
var5 := CTD_1.CV ;
```

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var5 或使用常量

CTD 指令处理的数据类型

输入	数据类型	描述
CD	BOOL	如果 CD 出现一个上升沿，CV 减 1
LOAD	BOOL	LOAD 为 FALSE 时，启动计数，为 TRUE 时，将 PV 赋给 CV，初始化计数器。
PV	INT	递减计数起始值
输出	数据类型	描述
Q	BOOL	当 CV=0 时，Q=1
CV	INT	计数值

5.2.2 递增计数器——CTU 指令

功能

CTU 指令用于对输入端做递增计数。

RESET 端为 FALSE 时，如果 CU 输入端出现一个上升沿，则 CV 端加 1。如果计数值 CV 达到了记数器的上限值 PV，则在 Q 输出端发出 TRUE 信号，CTU 功能块停止记数。

RESET 端为 TRUE 时，计数器停止计数，CV 端清零。

用法

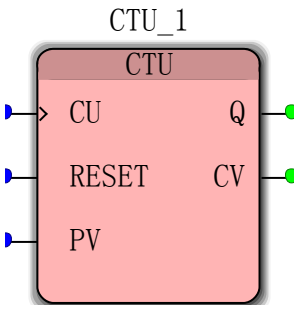
IL 编程语言

```
LD var1
ST CTU_1.CU
LD var2
ST CTU_1.RESET
LD var3
ST CTU_1.PV
CAL CTU_1
LD CTU_1.Q
ST var4
LD CTU_1.CV
ST var5
```

ST 编程语言

```
CTU_1(CD := var1, LOAD := var2, PV := var3) ;
var4 := CTU_1.Q ;
var5 := CTU_1.CV ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var5 或使用常量

CTU 指令处理的数据类型

输入	数据类型	描述
CU	BOOL	如果 CU 出现一个上升沿，CV 加 1
RESET	BOOL	RESET 为 FALSE 时，启动计数；为 TRUE 时，将 CV 清零，初始化计数器，Q 端复位。

PV	INT	递增计数上限值
输出	数据类型	描述
Q	BOOL	当 CV=PV 时，Q=1
CV	INT	计数值

5.2.3 增减双向计数器——CTUD 指令

功能

CTUD 指令用于对输入端做递增或者递减计数。

RESET 端和 LOAD 端同为 FALSE 时，允许计数：

如果 CU 输入端出现一个上升沿，则 CV 端加 1；

如果 CD 输入端出现一个上升沿，则 CV 端减 1；

当 CV=PV 时，则 QU=1，CTUD 功能块停止递增计数；

如果 CV=0，则 QD=1，CTUD 功能块停止递减记数。

RESET 端为 TRUE 时，计数器停止递增、递减计数，CV 端清零；

LOAD 端为 TRUE 时，计数器停止递增、递减计数，PV 值赋给 CV 端。

用法

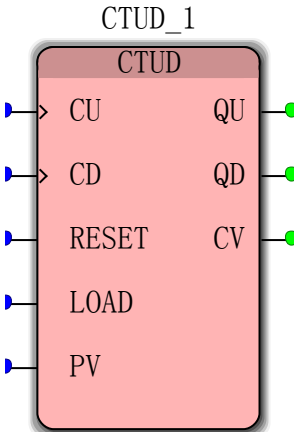
IL 编程语言

```
LD var1
ST CTUD_1.CU
LD var2
ST CTUD_1.CD
LD var3
ST CTUD_1.RESET
LD var4
ST CTUD_1.LOAD
LD var5
ST CTUD_1.PV
CAL CTUD_1
LD CTUD_1.QU
ST var6
LD CTUD_1.QD
ST var7
LD CTUD_1.CV
ST var8
```

ST 编程语言

```
CTUD_1(CU := var1, CD := var2, RESET ：= var3， LOAD := var4, PV := var5) ;
var6 := CTUD_1.QU ;
var7 := CTUD_1.QD ;
var8 := CTUD_1.CV ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var8 或使用常量

CTU 指令处理的数据类型

输入	数据类型	描述
CU	BOOL	如果 CU 出现一个上升沿，CV 加 1

CD	BOOL	如果 CD 出现一个上升沿，CV 减 1
RESET	BOOL	RESET 为 FALSE 时，启动计数，为 TRUE 时，将 CV 清零，初始化计数器。
LOAD	BOOL	LOAD 为 FALSE 时，启动计数，为 TRUE 时，将 PV 赋给 CV，初始化计数器。
PV	INT	计数上限值或起始值
输出	数据类型	描述
QU	BOOL	当 CV=PV 时，QU=1
QD	BOOL	当 CV=0 时，QD=1
CV	INT	计数值

5.2.4 下降沿检测——F_TRIG 指令

功能

F_TRIG 指令用于检测输入端的下降沿。

如果在输入端 CLK 检测到一个下降沿，则输出端 Q 由 FALSE 变为 TRUE，直到下一次扫描到这条指令，Q 输出端都将一直保持为 TRUE。

用法

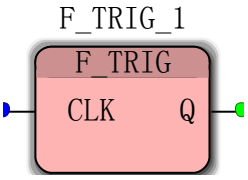
IL 编程语言

```
LD var1
ST F_TRIG_1.CLK
CAL F_TRIG_1
LD F_TRIG_1.Q
ST var2
```

ST 编程语言

```
F_TRIG_1(CLK := var1) ;
var2 := F_TRIG_1.Q ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var2 或使用常量

F_TRIG 指令处理的数据类型

参数	数据类型	描述
CLK	BOOL	下降沿有效
Q	BOOL	当 CLK 出现下降沿时，Q=由 0 变为 1，直到下一次扫描到这条指令

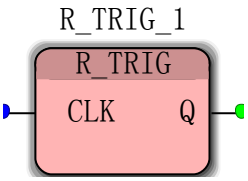
5.2.5 上升沿检测——R_TRIG 指令

功能

R_TRIG 指令用于检测输入端的上升沿。

如果在输入端 CLK 检测到一个上升沿，则输出端 Q 由 FALSE 变为 TRUE，直到下一次扫描到这条指令，Q 输出端都将一直保持为 TRUE。

用法

IL 编程语言	LD、 FBD 编程语言
LD var1 ST R_TRIG_1.CLK CAL R_TRIG_1 LD R_TRIG_1.Q ST var2	
ST 编程语言	
R_TRIG_1(CLK := var1) ; var2 := R_TRIG_1.Q ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var2 或使用常量	

CTU 指令处理的数据类型

参数	数据类型	描述
CLK	BOOL	上升沿有效
Q	BOOL	当 CLK 出现上升沿时，Q=由 0 变为 1，直到下一次扫描到这条指令

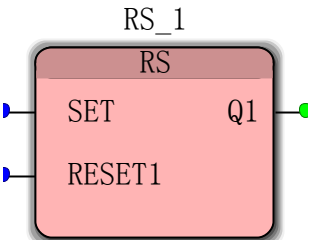
5.2.6 RS 触发器——RS 指令

功能

RS 指令用于实现 RS 触发器的功能。

如果 SET 端为 TRUE 且 RESET 端为 FALSE，则输出端 Q1 被置位，即使 SET 变为 FALSE，Q1 仍然保持置位状态。如果 RESET1=TRUE，则不论 SET 端为 TRUE 还是 FALSE，Q1 端均被复位，即使 RESET1 变为 FALSE，Q1 仍然保持复位状态。

用法

IL 编程语言	LD、 FBD 编程语言
LD var1 ST RS_1.SET LD var2 ST RS_1.RESET1 CAL RS_1 LD RS_1.Q1 ST var3	
ST 编程语言	
RS_1(SET := var1, RESET := var2) ; var3 := RS_1.Q1 ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var3 或使用常量	

RS 指令处理的数据类型

参数	数据类型	描述

SET	BOOL	置位
RESET1	BOOL	复位
Q1	BOOL	结果 SET=0，RESET1=0，Q1 保持上次状态； SET=1，RESET1=0，Q1=1； SET=0，RESET1=1，Q1=0； SET=1，RESET1=1，Q1=0；

5.2.7 SR 触发器——SR 指令

功能

RS 指令用于实现 RS 触发器的功能。

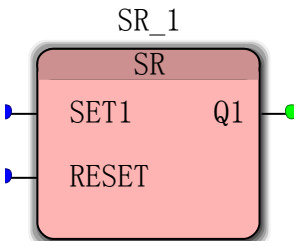
如果 SET1 端为 TRUE，则不论 RESET 端为 TRUE 还是 FALSE，输出端 Q1 被置位，即使 SET1 变为 FALSE，Q1 仍然保持置位状态。如果 RESET 端为 TRUE 且 SET1 端为 FALSE，则 Q1 端均被复位，即使 RESET 变为 FALSE，Q1 仍然保持复位状态。

用法

IL 编程语言

```
LD var1
ST SR_1.SET1
LD var2
ST SR_1.RESET
CAL SR_1
LD SR_1.Q1
ST var3
```

LD、 FBD 编程语言



ST 编程语言

```
SR_1(SET1 := var1, RESET := var2) ;
var3 := SR_1.Q1 ;
```

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var3 或使用常量

RS 指令处理的数据类型

参数	数据类型	描述
SET1	BOOL	置位
RESET	BOOL	复位
Q1	BOOL	结果 SET1=0，RESET=0，Q1 保持上次状态； SET1=1，RESET=0，Q1=1； SET1=0，RESET=1，Q1=0； SET1=1，RESET=1，Q1=1；

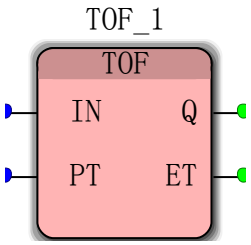
5.2.8 延迟断开定时器——TOF 指令

功能

TOF 指令用于实现延迟断开功能。

如果输入端 IN 为 TRUE，则输出端 Q 立即变为 TRUE；如果输入端 IN 由 TRUE 变为 FALSE，输出端 Q 将延迟一定时间后再由 TRUE 变为 FALSE，这个延迟时间为 PT 的值，ET 端记录了从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间。

用法

IL 编程语言	LD、 FBD 编程语言
LD var1 ST TOF_1.IN LD var2 ST TOF_1.PT CAL TOF_1 LD TOF_1.Q ST var3 LD TOF_1.ET ST var4	
ST 编程语言	
TOF_1(IN := var1, PT := var2) ; var3 := TOF_1.Q ; var4 := TOF_1.ET ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var4 或使用常量	

TOF 指令处理的数据类型

参数	数据类型	描述
IN	BOOL	使能输入端
PT	TIME	要 Q 端延迟断开的时间
Q	BOOL	结果 IN=1，Q=1；IN=0，延迟 PT 时间后，Q 由 1 变为 0
ET	TIME	从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间

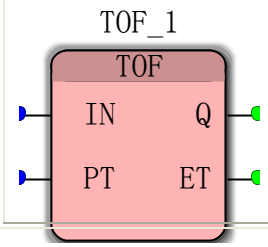
5.2.9 延迟接通定时器——TON 指令

功能

TON 指令用于实现延迟接通功能。

如果输入端 IN 为 FALSE，则输出端 Q 立即变为 FALSE；如果输入端 IN 由 FALSE 变为 TRUE，输出端 Q 将延迟一定时间后再由 FALSE 变为 TRUE，这个延迟时间为 PT 的值，ET 端记录了从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间。

用法

IL 编程语言	LD、 FBD 编程语言
LD var1 ST TON_1.IN LD var2 ST TON_1.PT CAL TON_1 LD TON_1.Q ST var3	

```
LD TON_1.ET
ST var4
```

ST 编程语言

```
TON_1(IN := var1, PT := var2) ;
var3 := TON_1.Q ;
var4 := TON_1.ET;
```

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var4 或使用常量

TON 指令处理的数据类型

参数	数据类型	描述
IN	BOOL	使能输入端
PT	TIME	要 Q 端延迟接通的时间，如 T#5S
Q	BOOL	结果 如果 IN=0，则 Q=0；如果 IN=1，延迟 PT 时间后，Q 由 0 变为 1
ET	TIME	从 IN 变为 TRUE 的时刻起到 Q 端变为 TRUE 之间的计时时间

注：如果要引用定时器（如用户插入了一个 TON_1 功能块）的输出 Q 端，则可以插入一个触电，名称为 TON_1。Q 即可。

5.2.10 脉冲——TP 指令

功能

TP 指令用于实现一定宽度脉冲的功能。

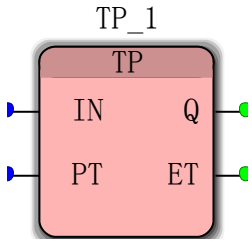
如果输入端 IN 由 FALSE 变为 TRUE，则输出端 Q 产生一个时间间隔为 PT 的脉冲。如果输入端 IN 在 PT 时间内又变为 FALSE，则输出端 Q 照样产生一个 PT 宽度的脉冲。ET 端记录了从 IN 变为 FALSE 的时刻起到 Q 端变为 TRUE 之间的计时时间。

用法

IL 编程语言

```
LD var1
ST TP_1.IN
LD var2
ST TP_1.PT
CAL TP_1
LD TP_1.Q
ST var3
LD TP_1.ET
ST var4
```

LD、 FBD 编程语言



ST 编程语言

```
TP_1(IN := var1, PT := var2) ;
var3 := TP_1.Q ;
var4 := TP_1.ET ;
```

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 var1~var4 或使用常量

TON 指令处理的数据类型

参数	数据类型	描述
IN	BOOL	IN 的上升沿有效
PT	TIME	脉冲时间间隔
Q	BOOL	结果，IN 上升沿时，Q 产生一个 PT 宽度的脉冲
ET	TIME	从 IN 变为 TRUE 的时刻起到 Q 端变为 TRUE 之间的计时时间，期间 IN 的状态变化对 Q 不起作用

5.3 类型转换 FU

类型转换功能，简称类型转换 FU，它将一个类型的数据转换为另一个类型的数据，所以它带有一个输入参数和一个输出参数。

MULTIPROG 编程过程中，可以使用以下的类型转换 FU

- BYTE 型 BCD 数据的转换
- WORD 型 BCD 数据的转换
- DWORD 型 BCD 数据的转换
- BOOL 型数据的转换
- BYTE 型数据的转换
- WORD 型数据的转换
- DWORD 型数据的转换
- SINT 型数据的转换
- INT 型数据的转换
- DINT 型数据的转换
- USINT 型数据的转换
- UINT 型数据的转换
- UDINT 型数据的转换
- LREAL 型数据的转换
- REAL 型数据的转换
- TRUNC 小数取整

功能中包含的指令（在编辑向导中，从下拉列表选择“类型转换 FU”）

分类	功能		
BYTE 型 BCD 数据的转换	B_BCD_TO_SINT	B_BCD_TO_INT	B_BCD_TO_DINT
WORD 型 BCD 数据的转换	W_BCD_TO_SINT	W_BCD_TO_INT	W_BCD_TO_DINT
DWORD 型 BCD 数据的转换	D_BCD_TO_SINT	D_BCD_TO_INT	D_BCD_TO_DINT
BCD 型数据的转换	BCD_TO_DINT		
TIME 型数据的转换	TIME_TO_DINT		
BOOL 型数据的转换	BOOL_TO_BYTE	BOOL_TO_WORD	BOOL_TO_DWORD
	BOOL_TO_SINT	BOOL_TO_INT	BOOL_TO_DINT
	BOOL_TO_USINT	BOOL_TO_UINT	BOOL_TO_UDINT
	BOOL_TO_REAL	BOOL_TO_LREAL	
BYTE 型数据的转换	BYTE_TO_BOOL	BYTE_TO_WORD	BYTE_TO_DWORD

	BYTE_TO_SINT	BYTE_TO_INT	BYTE_TO_DINT
	BYTE_TO_USINT	BYTE_TO_UINT	BYTE_TO_UDINT
	BYTE_TO_REAL	BYTE_TO_LREAL	
WORD 型数据的转换	WORD_TO_BOOL	WORD_TO_BYTE	WORD_TO_DWORD
	WORD_TO_SINT	WORD_TO_INT	WORD_TO_DINT
	WORD_TO_USINT	WORD_TO_UINT	WORD_TO_UDINT
	WORD_TO_REAL	WORD_TO_LREAL	
DWORD 型数据的转换	DWORD_TO_BOOL	DWORD_TO_BYTE	DWORD_TO_WORD
	DWORD_TO_SINT	DWORD_TO_INT	DWORD_TO_DINT
	DWORD_TO_USINT	DWORD_TO_UINT	DWORD_TO_UDINT
	DWORD_TO_REAL	DWORD_TO_LREAL	
SINT 型数据的转换	SINT_TO_B_BCD	SINT_TO_W_BCD	SINT_TO_D_BCD
	SINT_TO_BOOL	SINT_TO_BYTE	SINT_TO_WORD
	SINT_TO_DWORD	SINT_TO_INT	SINT_TO_DINT
	SINT_TO_USINT	SINT_TO_UINT	SINT_TO_UDINT
	SINT_TO_REAL	SINT_TO_LREAL	
INT 型数据的转换	INT_TO_B_BCD	INT_TO_W_BCD	INT_TO_D_BCD
	INT_TO_BOOL	INT_TO_BYTE	INT_TO_WORD
	INT_TO_DWORD	INT_TO_SINT	INT_TO_DINT
	INT_TO_USINT	INT_TO_UINT	INT_TO_UDINT
	INT_TO_REAL	INT_TO_LREAL	
DINT 型数据的转换	DINT_TO_B_BCD	DINT_TO_W_BCD	DINT_TO_D_BCD
	DINT_TO_BOOL	DINT_TO_BYTE	DINT_TO_WORD
	DINT_TO_DWORD	DINT_TO_SINT	DINT_TO_INT
	DINT_TO_USINT	DINT_TO_UINT	DINT_TO_UDINT
	DINT_TO_REAL	DINT_TO_LREAL	DINT_TO_BCD
	DINT_TO_TIME		
USINT 型数据的转换	USINT_TO_BOOL	USINT_TO_BYTE	USINT_TO_WORD
	USINT_TO_DWORD	USINT_TO_SINT	USINT_TO_INT
	USINT_TO_DINT	USINT_TO_UINT	USINT_TO_UDINT
	USINT_TO_REAL	USINT_TO_LREAL	
UINT 型数据的转换	UINT_TO_BOOL	UINT_TO_BYTE	UINT_TO_WORD
	UINT_TO_DWORD	UINT_TO_SINT	UINT_TO_INT
	UINT_TO_DINT	UINT_TO_USINT	UINT_TO_UDINT
	UINT_TO_REAL	UINT_TO_LREAL	
UDINT 型数据的转换	UDINT_TO_BOOL	UDINT_TO_BYTE	UDINT_TO_WORD
	UDINT_TO_DWORD	UDINT_TO_SINT	UDINT_TO_INT
	UDINT_TO_DINT	UDINT_TO_USINT	UDINT_TO_UINT
	UDINT_TO_REAL	UDINT_TO_LREAL	
LREAL 型数据的转换	LREAL_TO_BOOL	LREAL_TO_BYTE	LREAL_TO_WORD
	LREAL_TO_DWORD	LREAL_TO_SINT	LREAL_TO_INT
	LREAL_TO_DINT	LREAL_TO_USINT	LREAL_TO_UINT
	LREAL_TO_UDINT	LREAL_TO_REAL	
REAL 型数据的转换	REAL_TO_BOOL	REAL_TO_BYTE	REAL_TO_WORD

	REAL_TO_DWORD	REAL_TO_SINT	REAL_TO_INT
	REAL_TO_DINT	REAL_TO_USINT	REAL_TO_UINT
	REAL_TO_UDINT	REAL_TO_LREAL	
TRUNC 型数据的转换	TRUNC	TRUNC_SINT	TRUNC_INT
	TRUNC_DINT		

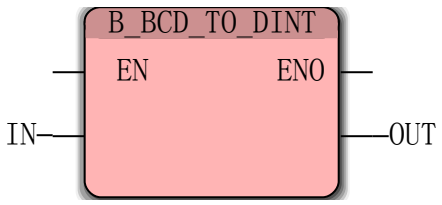
5.3.1 BYTE 型 BCD 数据的转换

BYTE 型 BCD 数据的转换包括以下三个指令：
B_BCD_TO_SINT、B_BCD_TO_INT 和 B_BCD_TO_DINT。
这三个指令能将一个 BYTE 数据类型的 BCD（二进制编码的十进制数）输入值分别转换为一个 SINT、INT 和 DINT 数据类型的输出值。

BYTE 型 BCD 数据的转换指令

指令	输入值	输出值	描述
B_BCD_TO_SINT	BYTE 型 BCD 码	SINT	输入值取值范围 BCD 码 16#00~99； 对应输出值 SINT、INT 和 DINT 均为 0~99。
B_BCD_TO_INT	BYTE 型 BCD 码	INT	
B_BCD_TO_DINT	BYTE 型 BCD 码	DINT	

用法（以 **B_BCD_TO_SINT** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN B_BCD_TO_SINT ST OUT	
ST 编程语言	
OUT := B_BCD_TO_SINT (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

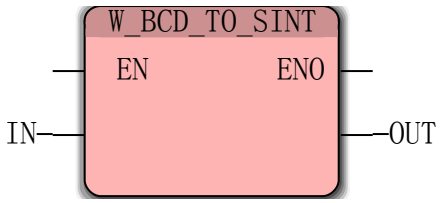
5.3.2 WORD 型 BCD 数据的转换

WORD 型 BCD 数据的转换包括以下三个指令：
W_BCD_TO_SINT、W_BCD_TO_INT 和 W_BCD_TO_DINT。
这三个指令能将一个 WORD 数据类型的 BCD（二进制编码的十进制数）输入值分别转换为一个 SINT、INT 和 DINT 数据类型的输出值。

WORD 型 BCD 数据的转换指令

指令	输入值	输出值	描述
W_BCD_TO_SINT	WORD 型 BCD 码	SINT	输入值 BCD 码 16#0000~16#00127，对应输出值 SINT 为 0~127，输入再增加则输出为-1；输入值 BCD 码 16#0000~16#9999，对应输出值 INT、DINT 均为 0~9999。
W_BCD_TO_INT	WORD 型 BCD 码	INT	
W_BCD_TO_DINT	WORD 型 BCD 码	DINT	

用法（以 **W_BCD_TO_SINT** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN W_BCD_TO_SINT ST OUT	
ST 编程语言	
OUT := W_BCD_TO_SINT (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.3.3 DWORD 型 BCD 数据的转换

DWORD 型 BCD 数据的转换包括以下三个指令：

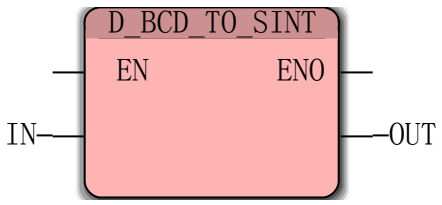
D_BCD_TO_SINT、D_BCD_TO_INT 和 D_BCD_TO_DINT。

这三个指令能将一个 DWORD 数据类型的 BCD（二进制编码的十进制数）输入值分别转换为一个 SINT、INT 和 DINT 数据类型的输出值。

DWORD 型 BCD 数据的转换指令

指令	输入值	输出值	描述
D_BCD_TO_SINT	DWORD 型 BCD 码	SINT	输入值 BCD 码 16#00000000~16#00000127，对应输出值 SINT 为 0~127，输入再增加则输出为 -1；输入值 BCD 码 16#00000000~16#00032767，输出值 INT 为 0~32,767，输入再增加则输出为-1；输入值 BCD 码 16#00000000~16#99999999，输出值 DINT 0~99999999。
D_BCD_TO_INT	DWORD 型 BCD 码	INT	
D_BCD_TO_DINT	DWORD 型 BCD 码	DINT	

用法（以 **D_BCD_TO_SINT** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN D_BCD_TO_SINT ST OUT	
ST 编程语言	
OUT := D_BCD_TO_SINT (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.3.4 BCD 型数据的转换

BCD 型数据的转换指令 BCD_TO_DINT，用于将一个 BCD（二进制编码的十进制数）输入值分别转换为一个 DINT 数据类型的输出值。

这条指令同 D_BCD_TO_DINT，详见 D_BCD_TO_DINT 的用法。

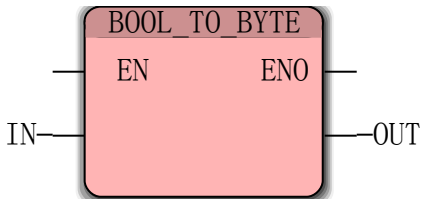
5.3.5 BOOL 型数据的转换

BOOL 型数据的转换有 11 个指令，可以将 BOOL 型数据分别转换为 BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL 和 LREAL 等类型。

BOOL 型数据的转换指令

指令	输入值	输出值	描述
BOOL_TO_BYTE	BOOL	BYTE	输入值取值范围 FALSE 或 TRUE 输入为 FALSE 时，输出为 0； 输入为 TRUE 时，输出为 1
BOOL_TO_WORD	BOOL	WORD	
BOOL_TO_DWORD	BOOL	DWORD	
BOOL_TO_SINT	BOOL	SINT	
BOOL_TO_INT	BOOL	INT	
BOOL_TO_DINT	BOOL	DINT	
BOOL_TO_USINT	BOOL	USINT	
BOOL_TO_UINT	BOOL	UINT	
BOOL_TO_UDINT	BOOL	UDINT	
BOOL_TO_REAL	BOOL	REAL	
BOOL_TO_LREAL	BOOL	LREAL	

用法（以 **BOOL_TO_BYTE** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN BOOL_TO_BYTE ST OUT	
ST 编程语言	
OUT := BOOL_TO_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.3.6 BYTE 型数据的转换

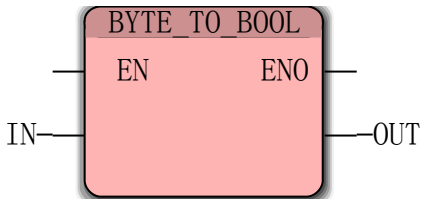
BYTE 型数据的转换有 11 个指令，可以将 BYTE 型数据分别转换为 BOOL、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL 和 LREAL 等类型。

BYTE 型数据的转换指令

指令	输入值	输出值	描述
BYTE_TO_BOOL	BYTE	BOOL	输入值取值范围 0~255； ● 输出为 BOOL 型：仅当输入为 0 时，输出为 FALSE，其它情况下输出均为 TRUE；
BYTE_TO_WORD	BYTE	WORD	
BYTE_TO_DWORD	BYTE	DWORD	

BYTE_TO_SINT	BYTE	SINT	● 输出为 SINT 型：输入 0~127 对应输出 0~127，输入 128~255 对应输出-128~-1； ● 输出为 WORD、DWORD 等其它类型时，输出等于输入。
BYTE_TO_INT	BYTE	INT	
BYTE_TO_DINT	BYTE	DINT	
BYTE_TO_USINT	BYTE	USINT	
BYTE_TO_UINT	BYTE	UINT	
BYTE_TO_UDINT	BYTE	UDINT	
BYTE_TO_REAL	BYTE	REAL	
BYTE_TO_LREAL	BYTE	LREAL	

用法（以 BYTE_TO_BOOL 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN BYTE_TO_ BOOL ST OUT	
ST 编程语言	
OUT := BYTE_TO_BOOL (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

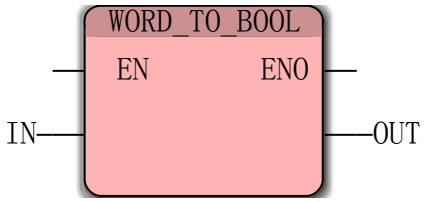
5.3.7 WORD 型数据的转换

WORD 型数据的转换有 11 个指令，可以将 WORD 型数据分别转换为 BOOL、BYTE、DWORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL 和 LREAL 等类型。

WORD 型数据的转换指令

指令	输入值	输出值	描述
WORD_TO_BOOL	WORD	BOOL	输入值取值范围 0~65,535； ● 输出为 BOOL 型：仅当输入为 0 时，输出为 FALSE，其它情况下输出均为 TRUE； ● 输出为 SINT 型：输入 0~127 对应输出 0~127，输入 128~255 对应输出-128~-1，输入再增加则输出将重复 0~127, -128~-1； ● 输出为 USINT、BYTE 型：输入 0~255 对应输出 0~255，输入再增加则输出将重复 0~255； ● 输出为 INT 型：输入 0~32767 对应输出 0~32767，输入 32768~65535 对应输出-32768~-1； ● 输出为 WORD、DWORD 等其它类型时，输出等于输入。
WORD_TO_BYTE	WORD	BYTE	
WORD_TO_DWORD	WORD	DWORD	
WORD_TO_SINT	WORD	SINT	
WORD_TO_INT	WORD	INT	
WORD_TO_DINT	WORD	DINT	
WORD _TO_USINT	WORD	USINT	
WORD _TO_UINT	WORD	UINT	
WORD _TO_UDINT	WORD	UDINT	
WORD _TO_REAL	WORD	REAL	
WORD_TO_LREAL	WORD	LREAL	

用法（以 WORD_TO_BOOL 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN WORD_TO_BOOL ST OUT	
ST 编程语言	
OUT := WORD_TO_BOOL (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

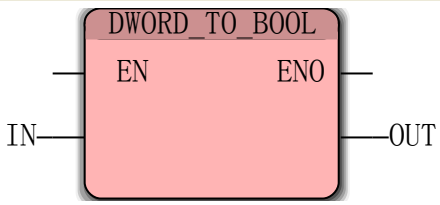
5.3.8 DWORD 型数据的转换

DWORD 型数据的转换有 11 个指令，可以将 DWORD 型数据分别转换为 BOOL、BYTE、WORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL 和 LREAL 等类型。

DWORD 型数据的转换指令

指令	输入值	输出值	描述
DWORD_TO_BOOL	DWORD	BOOL	输入值取值范围 0~4,294,967,295； ● 输出为 BOOL 型：仅当输入为 0 时，输出为 FALSE，其它情况下输出均为 TRUE； ● 输出为 SINT 型：输入 0~127 对应输出 0~127，输入 128~255 对应输出-128~-1，输入再增加则输出将重复 0~127, -128~-1； ● 输出为 USINT、BYTE 型：输入 0~255 对应输出 0~255，输入再增加则输出将重复 0~255； ● 输出为 INT 型：输入 0~32767 对应输出 0~32767，输入 32768~65535 对应输出-32768~-1，输入再增加则输出将重复 0~32767, -32768~-1；； ● 输出为 UINT、WORD 型：输入 0~65535 对应输出 0~65535，输入再增加则输出将重复 0~65535； ● 输出为 BYTE、WORD 等类型时，输出分别等于输入的低 8 位、低 16 位数据。
DWORD_TO_BYTE	DWORD	BYTE	
DWORD_TO_WORD	DWORD	WORD	
DWORD_TO_SINT	DWORD	SINT	
DWORD_TO_INT	DWORD	INT	
DWORD_TO_DINT	DWORD	DINT	
DWORD _TO_USINT	DWORD	USINT	
DWORD _TO_UINT	DWORD	UINT	
DWORD _TO_UDINT	DWORD	UDINT	
DWORD _TO_REAL	DWORD	REAL	
DWORD_TO_LREAL	DWORD	LREAL	

用法（以 **DWORD_TO_BOOL** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN DWORD _TO_ BOOL ST OUT	
ST 编程语言	
OUT := DWORD _TO_ BOOL (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

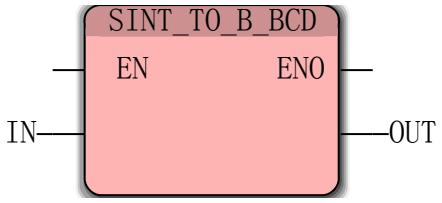
5.3.9 SINT 型数据的转换

SINT 型数据的转换有 14 个指令，可以将 SINT 型数据分别转换为 B_BCD、W_BCD、D_BCD、BOOL、BYTE、WORD、DWORD、INT、DINT、USINT、UINT、UDINT、REAL 和 LREAL 等类型。

SINT 型数据的转换指令

指令	输入值	输出值	描述 输入值取值范围-128~127；
SINT_TO_B_BCD	SINT	BYTE	输入 0~99 时，输出 16#0~99；输入其它值时，输出 16#FF
SINT_TO_W_BCD	SINT	WORD	输入 0~127 时，输出 16#0~127；输入其它值时，输出 16#FFFF
SINT_TO_D_BCD	SINT	DWORD	输入 0~127 时，输出 16#0~127，输入其它值时，输出 16#FFFFFFFF
SINT _TO_BOOL	SINT	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
SINT _TO_BYTE	SINT	BYTE	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_WORD	SINT	WORD	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_DWORD	SINT	DWORD	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_INT	SINT	INT	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_DINT	SINT	DINT	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_USINT	SINT	USINT	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_UINT	SINT	UINT	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_UDINT	SINT	UDINT	输入 0~127 时，输出 0~127；输入-128~-1，输出 128~255
SINT _TO_REAL	SINT	REAL	输入 0~127 时，输出 0~127；输入-128~-1，输出-128~-1
SINT _TO_LREAL	SINT	LREAL	输入 0~127 时，输出 0~127；输入-128~-1，输出-128~-1

用法（以 SINT_TO_B_BCD 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN SINT_TO_B_BCD ST OUT	
ST 编程语言	
OUT := SINT_TO_B_BCD (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.3.10 INT 型数据的转换

INT 型数据的转换有 14 个指令，可以将 INT 型数据分别转换为 B_BCD、W_BCD、D_BCD、BOOL、BYTE、WORD、DWORD、SINT、DINT、USINT、UINT、UDINT、REAL 和 LREAL

等类型。

INT 型数据的转换指令

指令	输入值	输出值	描述 输入值取值范围-32768~32767；
INT_TO_B_BCD	INT	BYTE	输入 0~99 时，输出 16#0~99；输入其它值时，输出 16#FF
INT_TO_W_BCD	INT	WORD	输入 0~9999 时，输出 16#0~9999；输入其它值时，输出 16#FFFF
INT_TO_D_BCD	INT	DWORD	输入 0~32767 时，输出 16#0~32767，输入其它值时，输出 16#FFFFFFFF
INT _TO_BOOL	INT	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
INT_TO_BYTE	INT	BYTE	输入 0~255 时，输出 0~255；输入再增加则输出将重复 0~255；输入-1~-255，输出 255~0；输入再减小则输出重复 255~0
INT_TO_WORD	INT	WORD	输入 0~32767 时，输出 0~32767；输入-32768~-1，输出 32768~65535
INT_TO_DWORD	INT	DWORD	输入 0~32767 时，输出 0~32767；输入-32768~-1，输出 32768~65535
INT_TO_SINT	INT	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出 -128~-1；输入再增加则输出重复 0~127, -128~-1；输入 -1~-128，输出-1~-128；输入-129~-256，输出 127~0；输入再减小则输出重复-1~-128, 127~0
INT_TO_DINT	INT	DINT	输入-32768~32767 时，输出-32768~32767
INT_TO_USINT	INT	USINT	输入 0~255 时，输出 0~255；输入再增加则输出重复 0~255
INT_TO_UINT	INT	UINT	输入 0~32767 时，输出 0~32767；输入-32768~-1，输出 32768~65535
INT_TO_UDINT	INT	UDINT	输入 0~32767 时，输出 0~32767；输入-32768~-1，输出 32768~65535
INT_TO_REAL	INT	REAL	输入 0~32767 时，输出 0~32767；输入-32768~-1，输出-32768~-1
INT_TO_LREAL	INT	LREAL	输入 0~32767 时，输出 0~32767；输入-32768~-1，输出-32768~-1

用法（以 INT_TO_B_BCD 为例）

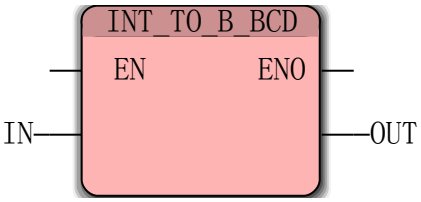
IL 编程语言

```
LD IN
INT_TO_ B_BCD
ST OUT
```

ST 编程语言

```
OUT := INT_TO_B_BCD (IN) ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量

5.3.11 DINT 型数据的转换

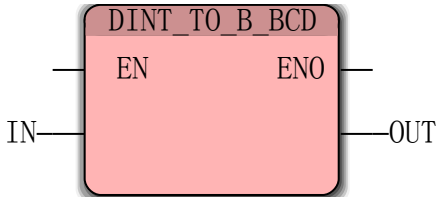
DINT 型数据的转换有 16 个指令，可以将 DINT 型数据分别转换为 B_BCD、W_BCD、D_BCD、BOOL、BYTE、WORD、DWORD、SINT、INT、USINT、UINT、UDINT、REAL、LREAL、BCD 和 TIME 等类型。

DINT 型数据的转换指令

指令	输入值	输出值	描述 输入值取值范围-2,147,483,648…2,147,483,647
DINT_TO_B_BCD	DINT	BYTE	输入 0~99 时，输出 16#0~99；输入其它值时，输出 16#FF
DINT_TO_W_BCD	DINT	WORD	输入 0~9999 时，输出 16#0~9999；输入其它值时，输出 16#FFFF
DINT_TO_D_BCD	DINT	DWORD	输入 0~99999999 时，输出 16#0~99999999，输入其它值时，输出 16#FFFFFFFF
DINT _TO_BOOL	DINT	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
DINT _TO_BYTE	DINT	BYTE	输入 0~255 时，输出 0~255；输入再增加则输出将重复 0~255；输入-1~-255，输出 255~0；输入再减小则输出重复 255~0
DINT _TO_WORD	DINT	WORD	输入 0~65535 时，输出 0~65535；输入再增加则输出重复 0~65535；输入-1~-65536，输出 65535~0；输入再减小则输出重复 65535~0
DINT _TO_DWORD	DINT	DWORD	输入 0~2,147,483,647 时，输出 0~2,147,483,647；输入-2,147,483,648~-1，输出 2,147,483,648~ 4,294,967,295
DINT _TO_SINT	DINT	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出 -128~-1；输入再增加则输出重复 0~127, -128~-1；输入 -1~-128，输出-1~-128；输入-129~-256，输出 127~0；输入再减小则输出重复-1~-128, 127~0
DINT _TO_INT	DINT	INT	输入-32768~32767 时，输出-32768~32767；输入大于 32767 时，输出重复-32768~32767；输入小于-32768 是，输出重复 32767~-32768
DINT _TO_USINT	DINT	USINT	输入 0~255 时，输出 0~255；输入再增加则输出重复 0~255
DINT _TO_UINT	DINT	UINT	输入 0~65535 时，输出 0~65535；输入再增加则输出重复 0~65535；输入-65536~-1，输出 0~65535；输入再减小则输出重复 65535~0
DINT _TO_UDINT	DINT	UDINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647；输入-2,147,483,648~-1，输出 2,147,483,648~ 4,294,967,295

DINT _TO_REAL	DINT	REAL	输入 0~2,147,483,647 时，输出 0~2,147,483,647；输入-2,147,483,648~-1，输出-2,147,483,648~-1，精度会降低
DINT _TO_LREAL	DINT	LREAL	输入 0~2,147,483,647 时，输出 0~2,147,483,647；输入-2,147,483,648~-1，输出-2,147,483,648~-1，精度会降低
DINT _TO_BCD	DINT	BCD	输入值 DINT 0~99999999，输出值 BCD 码 16#00000000~16#99999999。
DINT _TO_TIME	DINT	TIME	输出值单位为秒；输入值 0~2,147,483,647，输出值 0~2147483.647 秒；输入值-2147483648~-1 秒，输出值 2147483.648~4294967.295 秒

用法（以 DINT_TO_B_BCD 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN DINT _TO_ B_BCD ST OUT	
ST 编程语言	
OUT := DINT_TO_B_BCD (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

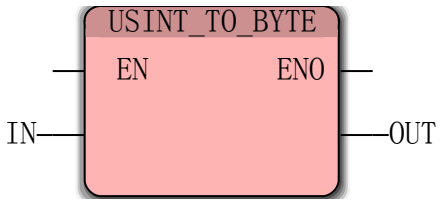
5.3.12 USINT 型数据的转换

USINT 型数据的转换有 11 个指令，可以将 USINT 型数据分别转换为 BOOL、BYTE、WORD、DWORD、SINT、INT、DINT、UINT、UDINT、REAL 和 LREAL 等类型。

USINT 型数据的转换指令

指令	输入值	输出值	描述 输入值取值范围 0~255；
USINT _TO_BOOL	USINT	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
USINT _TO_BYTE	USINT	BYTE	输入 0~255 时，输出 0~255
USINT _TO_WORD	USINT	WORD	输入 0~255 时，输出 0~255
USINT _TO_DWORD	USINT	DWORD	输入 0~255 时，输出 0~255
USINT _TO_SINT	USINT	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出-128~-1
USINT _TO_INT	USINT	INT	输入 0~255 时，输出 0~255
USINT _TO_DINT	USINT	DINT	输入 0~255 时，输出 0~255
USINT _TO_UINT	USINT	UINT	输入 0~255 时，输出 0~255
USINT _TO_UDINT	USINT	UDINT	输入 0~255 时，输出 0~255
USINT _TO_REAL	USINT	REAL	输入 0~255 时，输出 0~255
USINT _TO_LREAL	USINT	LREAL	输入 0~255 时，输出 0~255

用法（以 **USINT_TO_BYTE** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN USINT _TO_BYTE ST OUT	
ST 编程语言	
OUT := USINT_TO_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

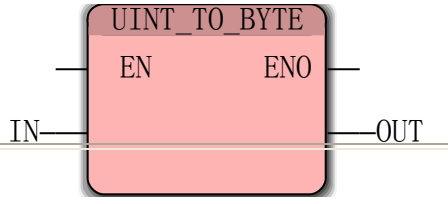
5.3.13 UINT 型数据的转换

UINT 型数据的转换有 11 个指令，可以将 UINT 型数据分别转换为 BOOL、BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UDINT、REAL 和 LREAL 等类型。

UINT 型数据的转换指令

指令	输入值	输出值	描述 输入值取值范围 0~65535
UINT _TO_BOOL	UINT	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
UINT _TO_BYTE	UINT	BYTE	输入 0~255 时，输出 0~255；输入再增加则输出将重复 0~255
UINT _TO_WORD	UINT	WORD	输入 0~65535 时，输出 0~65535
UINT _TO_DWORD	UINT	DWORD	输入 0~65535 时，输出 0~65535
UINT _TO_SINT	UINT	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出 -128~-1；输入再增加则输出重复 0~127, -128~-1
UINT _TO_INT	UINT	INT	输入 0~32767 时，输出 0~32767；输入 32768~65535，输出-32768~-1
UINT _TO_DINT	UINT	DINT	输入 0~65535 时，输出 0~65535
UINT _TO_USINT	UINT	USINT	输入 0~255 时，输出 0~255；输入再增加则输出重复 0~255
UINT _TO_UDINT	UINT	UDINT	输入 0~65535 时，输出 0~65535
UINT _TO_REAL	UINT	REAL	输入 0~65535 时，输出 0~65535
UINT _TO_LREAL	UINT	LREAL	输入 0~65535 时，输出 0~65535

用法（以 **UINT_TO_BYTE** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN UINT _TO_ BYTE ST OUT	
ST 编程语言	

OUT := UINT_TO_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

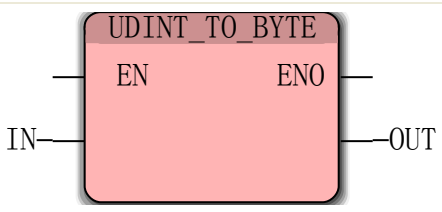
5.3.14 UDINT 型数据的转换

UDINT 型数据的转换有 11 个指令，可以将 UDINT 型数据分别转换为 BOOL、BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、REAL 和 LREAL 等类型。

UDINT 型数据的转换指令

指令	输入值	输出值	描述 输入值取值范围 0~4,294,967,295
UDINT _TO_BOOL	UDINT	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
UDINT _TO_BYTE	UDINT	BYTE	输入 0~255 时，输出 0~255；输入再增加则输出将重复 0~255
UDINT _TO_WORD	UDINT	WORD	输入 0~65535 时，输出 0~65535，输入再增加则输出将重复 0~65535
UDINT _TO_DWORD	UDINT	DWORD	输入 0~4,294,967,295 时，输出 0~4,294,967,295
UDINT _TO_SINT	UDINT	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出 -128~-1；输入再增加则输出重复 0~127, -128~-1
UDINT _TO_INT	UDINT	INT	输入 0~32767 时，输出 0~32767；输入 32768~65535，输出 -32768~-1，输入再增加则输出重复 0~32767, -32768~-1
UDINT _TO_DINT	UDINT	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647；输入 2,147,483,648~4,294,967,295，输出 -2,147,483,648~-1
UDINT _TO_USINT	UDINT	USINT	输入 0~255 时，输出 0~255；输入再增加则输出重复 0~255
UDINT _TO_UINT	UDINT	UDINT	输入 0~65535 时，输出 0~65535；输入再增加则输出重复 0~65535
UDINT _TO_REAL	UDINT	REAL	输入 0~4,294,967,295 时，输出 0~4,294,967,295，精度会降低
UDINT _TO_LREAL	UDINT	LREAL	输入 0~4,294,967,295 时，输出 0~4,294,967,295，精度会降低

用法（以 UDINT_TO_BYTE 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN UDINT _TO_ BYTE ST OUT	
ST 编程语言	
OUT := UDINT_TO_BYTE (IN) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量

5.3.15 REAL 型数据的转换

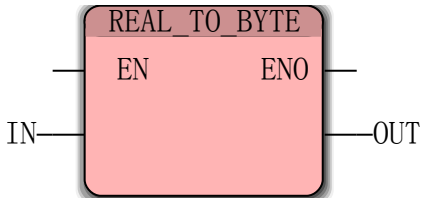
REAL 型数据的转换有 11 个指令，可以将 REAL 型数据分别转换为 BOOL、BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT 和 LREAL 等类型。

REAL 型数据的转换指令

指令	输入值	输出值	描述 REAL 型的输入值其小数部分在转换时被舍去
REAL_TO_BOOL	REAL	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
REAL_TO_BYTE	REAL	BYTE	输入 0~255 时，输出 0~255；输入再增加则输出将重复 0~255；输入-1~-255，输出 255~0；输入再减小则输出重复 255~0
REAL _TO_WORD	REAL	WORD	输入 0~65535 时，输出 0~65535，输入再增加则输出重复 0~65535；输入-1~-65536，输出 65535~0，输入再减小则输出重复 65535~0
REAL _TO_DWORD	REAL	DWORD	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出为 2,147,483,647；输入-1~-2,147,483,648，输出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648；精度会降低
IREAL _TO_SINT	REAL	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出-128~-1；输入再增加则输出重复 0~127, -128~-1；输入-1~-128，输出-1~-128；输入-129~-256，输出 127~0；输入再减小则输出重复-1~-128, 127~0
REAL _TO_INT	REAL	DINT	输入 0~32767 时，输出 0~32767；输入 32768~65535，输出-32768~-1；输入再增加则输出重复 0~32767, -32768~-1；输入-1~-32768，输出-1~-32768；输入-32769~-65536，输出 32767~0；输入再减小则输出重复-1~-32768, 32767~0
REAL _TO_DINT	REAL	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出仍为 2,147,483,647；输入-1~-2,147,483,648，输出-1~-2,147,483,648，输入再减小输出仍为-2,147,483,648
REAL _TO_USINT	REAL	USINT	输入 0~255 时，输出 0~255，输入再增加则输出重复 0~255；输入-1~-256，输出 255~0，输出再减小则输出重复 255~0
REAL _TO_UINT	REAL	UINT	输入 0~65535 时，输出 0~65535，输入再增加则输出重复 0~65535；输入-1~-65536，输出 65535~0，输入再减小则输出重复 65535~0
REAL _TO_UDINT	REAL	UDINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输

			入再增加输出为 2,147,483,647；输入-1~-2,147,483,648，输出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648；精度会降低
REAL _TO_LREAL	REAL	LREAL	输入等于输出

用法（以 **REAL_TO_BYTE** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN REAL_TO_ BYTE ST OUT	
ST 编程语言	
OUT := REAL_TO_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.3.16 LREAL 型数据的转换

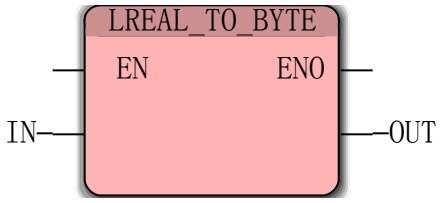
LREAL 型数据的转换有 11 个指令，可以将 LREAL 型数据分别转换为 BOOL、BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT 和 REAL 等类型。

LREAL 型数据的转换指令

指令	输入值	输出值	描述 REAL 型的输入值其小数部分在转换时被舍去
LREAL_TO_BOOL	LREAL	BOOL	输入 0 时，输出 FALSE；输入其它值时，输出 TRUE
LREAL_TO_BYTE	LREAL	BYTE	输入 0~255 时，输出 0~255；输入再增加则输出将重复 0~255；输入-1~-255，输出 255~0；输入再减小则输出重复 255~0
LREAL _TO_WORD	LREAL	WORD	输入 0~65535 时，输出 0~65535，输入再增加则输出重复 0~65535；输入-1~-65536，输出 65535~0，输入再减小则输出重复 65535~0
LREAL _TO_DWORD	LREAL	DWORD	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出为 2,147,483,647；输入-1~-2,147,483,648，输出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648；精度会降低
LIREAL _TO_SINT	LREAL	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出-128~-1；输入再增加则输出重复 0~127, -128~-1；输入-1~-128，输出-1~-128；输入-129~-256，输出 127~0；输入再减小则输出重复-1~-128, 127~0
LREAL _TO_INT	LREAL	DINT	输入 0~32767 时，输出 0~32767；输入 32768~65535，输出-32768~-1；输入再增加则输出重复 0~32767,

			-32768~-1；输入-1~-32768，输出-1~-32768；输入-32769~-65536，输出 32767~0；输入再减小则输出重复-1~-32768, 32767~0
LREAL _TO_DINT	LREAL	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出仍为 2,147,483,647；输入-1~-2,147,483,648，输出-1~-2,147,483,648，输入再减小输出仍为-2,147,483,648
LREAL _TO_USINT	LREAL	USINT	输入 0~255 时，输出 0~255，输入再增加则输出重复 0~255；输入-1~-256，输出 255~0，输出再减小则输出重复 255~0
LREAL _TO_UINT	LREAL	UINT	输入 0~65535 时，输出 0~65535，输入再增加则输出重复 0~65535；输入-1~-65536，输出 65535~0，输入再减小则输出重复 65535~0
LREAL _TO_UDINT	LREAL	UDINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出为 2,147,483,647；输入-1~-2,147,483,648，输出 4,294,967,295~2,147,483,648，输入再减小输出仍为 2,147,483,648；精度会降低
LREAL _TO_LREAL	LREAL	REAL	输入等于输出，精度会降低

用法（以 **REAL_TO_BYTE** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN LREAL_TO_ BYTE ST OUT	
ST 编程语言	
OUT := LREAL_TO_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.3.17 TRUNC 小数取整

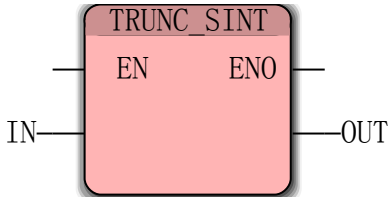
TRUNC 小数取整有 4 个指令，可以将浮点型数据分别转换为 SINT、INT、DINT 等类型。

TRUNC 小数取整指令

指令	输入值	输出值	描述
TRUNC	REAL	无符号整型	截去位于输入值的小数部分，得到一个整型值
TRUNC_SINT	REAL	SINT	输入 0~127 时，输出 0~127；输入 128~255，输出-128~-1；输入再增加则输出重复 0~127, -128~-1；输入-1~-128，输出-1~-128；输入-129~-256，输出 127~0；输入再减小则输出重复-1~-128, 127~0
TRUNC _INT	REAL	INT	输入 0~32767 时，输出 0~32767；输入

			32768~65535，输出-32768~-1；输入再增加则输出重复 0~32767，-32768~-1；输入-1~-32768，输出-1~-32768；输入-32769~-65536，输出 32767~0；输入再减小则输出重复-1~-32768, 32767~0
TRUNC _DINT	REAL	DINT	输入 0~2,147,483,647 时，输出 0~2,147,483,647，输入再增加输出仍为 2,147,483,647；输入-1~-2,147,483,648，输出-1~-2,147,483,648，输入再减小输出仍为-2,147,483,648

用法（以 **TRUNC_SINT** 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN TRUNC_SINT ST OUT	
ST 编程语言	
OUT := TRUNC_SINT (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.3.18 TIME 型数据的转换

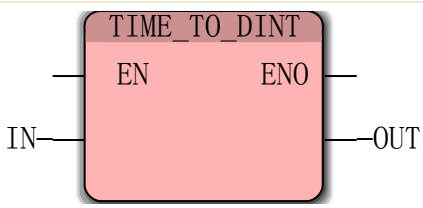
TIME_TO_DINT 类型转换功能将一个 TIME 型的输入值转换为一个 DINT 型的输出值（任何时间值都将换算为毫秒值再将毫秒值转换为 DINT）。

TIME 型数据必须是以 T#为开始的一个无符号数，大于 T#2147483647 的时间值都将为负数，因为 DINT 类型为有符号数，其最大值为 2,147,483,647，例如，输入值 T#4294967295 毫秒将导致输出值为-1。

TIME_TO_DINT 转换指令处理的数据类型

指令	输入值	输出值	描述
TIME_TO_DINT	TIME	DINT	输入值取值范围 T#0~2147483647MS（等于 T#0~2147483.647S），转换为 DINT 输出值为 0~2147483647；输入值取值范围 T#2147483648~4294967295，转换为 DINT 输出值为-2147483648~-1。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN TIME_TO_DINT ST OUT	
ST 编程语言	
OUT := TIME_TO_DINT (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

5.4 字符串 FU

字符串功能，简称字符串 FU，是对于字符串的比较、转换、查找、连接等操作，它带有多个输入参数和一个输出参数。

MULTIPROG 编程过程中，可以在编辑向导中，从下拉列表选择“字符串 FU”，使用以下指令

分类	功能		
合并插入删除替换长度限制	CONCAT	INSERT	DELETE
	REPLACE	LEN	LIMIT_STRING
	FIND		
大小与位置	MAX_STRING	MIN_STRING	LEFT
	MID	RIGHT	SEL_STRING
比较	GT_STRING	GE_STRING	EQ_STRING
	NE_STRING	LE_STRING	LT_STRING
字符串转换为其它	STRING_TO_BYTE	STRING_TO_WORD	STRING_TO_DWORD
	STRING_TO_SINT	STRING_TO_INT	STRING_TO_DINT
	STRING_TO_USINT	STRING_TO_UINT	STRING_TO_UDINT
	STRING_TO_LREAL	STRING_TO_REAL	STRING_TO_TIME
其它转换为字符串	BYTE_TO_STRING	WORD_TO_STRING	DWORD_TO_STRING
	SINT_TO_STRING	INT_TO_STRING	DINT_TO_STRING
	USINT_TO_STRING	UINT_TO_STRING	UDINT_TO_STRING
	REAL_TO_STRING	LREAL_TO_STRING	TIME_TO_STRING

5.4.1 合并字符串——CONCAT

功能

CONCAT 合并字符串指令用于合并两个字符串。

用法

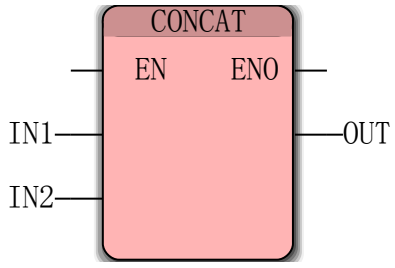
IL 编程语言

```
LD IN
CONCAT
ST OUT
```

ST 编程语言

```
OUT := CONCAT( IN1, IN2 ) ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1，IN2 和 OUT 或使用常量

CONCAT 指令处理的数据类型

输入	数据类型	描述
IN1	STRING	第一个输入

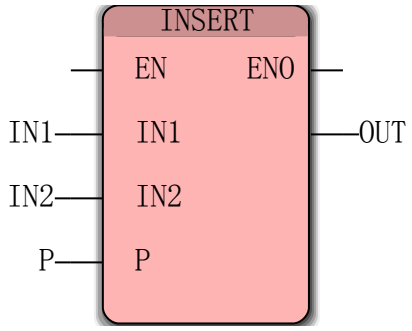
IN2	STRING	第二个输入
输出	数据类型	描述
OUT	STRING	输出，OUT=IN1+IN2； 第二个输入被添加到第一个输入的后面，输出不允许与输入同名

5.4.2 插入字符串——INSERT

功能

INSERT 插入字符串指令用于将一个字符串或字符插入到另一个字符串中。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 INSERT IN2, P ST OUT	
ST 编程语言	
OUT := INSERT(IN1, IN2, P) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1，IN2，P 和 OUT 或使用常量	

INSERT 指令处理的数据类型

输入和输出	数据类型	描述
IN1	STRING	第一个输入字符串，使用变量时，变量必须设非空的初值，也可使用字符串常量
IN2	STRING	第二个输入字符串——将被插入到第一个输入中，使用变量时，变量必须设非空的初值，也可使用字符串常量
P	ANY_INT	在第一个输入中的插入点，P 必须是大于 0 的整数；字符串中的第一个字符位置为 1，后面的字符为 2，3…；
OUT	STRING	输出 第二个输入被插入到第一个输入中第 P 个字符的后面，输出不允许与输入同名

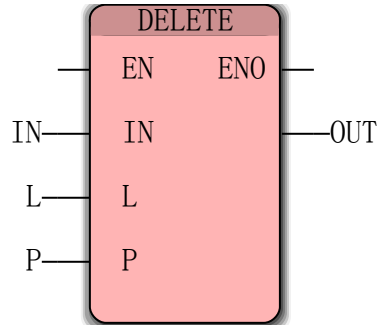
5.4.3 删除字符串——DELETE

功能

DELETE 删除字符串指令用于删除一个字符串中确定的几个相连的字符。

用法

IL 编程语言	LD、 FBD 编程语言

LD IN DELETE L, P ST OUT	
ST 编程语言	
OUT := DELETE(IN, L, P) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN，L，P 和 OUT 或使用常量	

DELETE 指令处理的数据类型

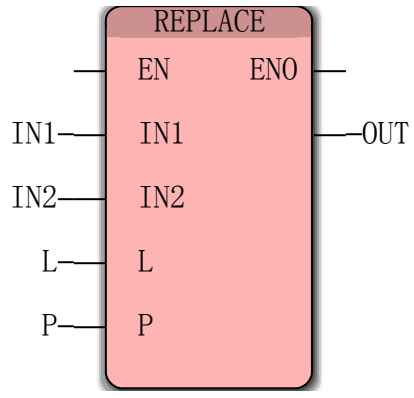
输入和输出	数据类型	描述
IN	STRING	输入字符串，也可使用常量字符串
L	ANY_INT	要删除的字符数，L 可以是 0，1，2...
P	ANY_INT	在输入字符串中要删除的第一个字符的位置，P 必须是大于 0 的整数；字符串中的第一个字符位置为 1，后面的字符为 2，3...
OUT	STRING	输出 输入字符串不受删除的影响，实际上本指令是在输入字符串中挑选几个字符串形成一个新的字符串，输出不允许与输入同名

5.4.4 替换字符串——REPLACE

功能

REPLACE 替换字符串指令可用一个字符串替换一个字符串中确定的几个相连的字符。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN REPLACE L, P ST OUT	
ST 编程语言	
OUT := REPLACE (IN, L, P) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2、L、P 和 OUT 或使用常量	

REPLACE 指令处理的数据类型

输入和输出	数据类型	描述

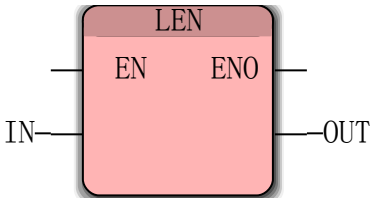
IN1	STRING	第一个输入字符串
IN2	STRING	第二个输入字符串——将要替换第一个输入字符串中的某些字符
L	ANY_INT	要替换的字符数，L 可以是 0，1，2…
P	ANY_INT	在第一个输入字符串中要替换的第一个字符的位置，P 必须是大于 0 的整数；字符串中的第一个字符位置为 1，后面的字符为 2，3…
OUT	STRING	输出 第一个输入字符串不受替换的影响，实际上本指令是在第一个输入字符串中挑选出前后两部分字符与第二个字符串形成一个新的字符串，输出不允许与输入同名

5.4.5 替换字符串——LEN

功能

LEN 替换字符串指令可用一个字符串替换一个字符串中确定的几个相连的字符。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN LEN ST OUT	
ST 编程语言	
OUT := LEN (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

LEN 指令处理的数据类型

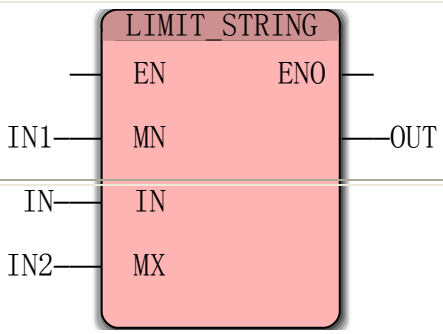
输入和输出	数据类型	描述
IN	STRING	输入字符串
OUT	INT	输出，一个字符的长度为 1

5.4.6 设置字符串极限——LIMIT_STRING

功能

LIMIT_STRING 指令用于限制输入的字符或字符串在上下限所确定的区间。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 LIMIT_STRING IN, IN2 ST OUT	
ST 编程语言	

OUT := LIMIT_STRING (IN1, IN, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1，IN，IN2 和 OUT 或使用常量	

LIMIT_STRING 指令处理的数据类型

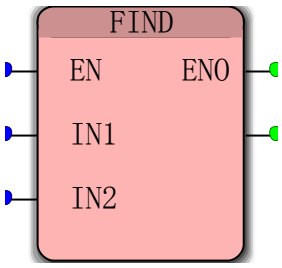
输入和输出	数据类型	描述
IN1（MN）	STRING	字符下限
IN	STRING	输入字符或字符串
IN2（MX）	STRING	字符上限
OUT	STRING	输出，输入值中高于上限和低于下限的部分将分别等于上、下限；字符的大小均按其 ASCII 码值的大小判断；当输入字符串时，仅第一个字符参与比较，即如果第一个字符在上下限之间，则字符串全部输出，否则仅输出上限或下限

5.4.7 查找字符串中的出现的一个字符——FIND

功能

FIND 指令用于确定在第一个字符串中出现的第二个字符串（以首字符为准）的位置。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 FIND IN2 ST OUT	
ST 编程语言	
OUT := FIND (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1，IN2 和 OUT 或使用常量	

FIND 指令处理的数据类型

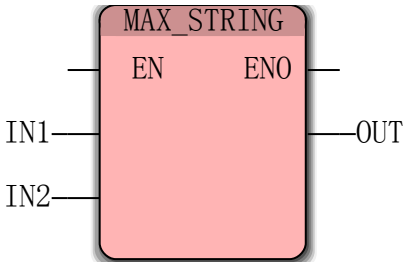
输入和输出	数据类型	描述
IN1	STRING	第一个输入字符或字符串，字符串中左起第一个字符的位置为 1
IN2	STRING	第二个输入字符或字符串，字符串中左起第一个字符的位置为 1
OUT	STRING	输出，第二个字符串中的第一个字符出现在第一个字符串中的位置，如果第一个字符串中不包含这个字符，输出为 0。

5.4.8 取较大的字符串——MAX_STRING

功能

MAX_STRING 指令用于确定两个字符中的较大的一个。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MAX_STRING IN2 ST OUT	
ST 编程语言	
OUT := MAX_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1，IN2 和 OUT 或使用常量	

MAX_STRING 指令处理的数据类型

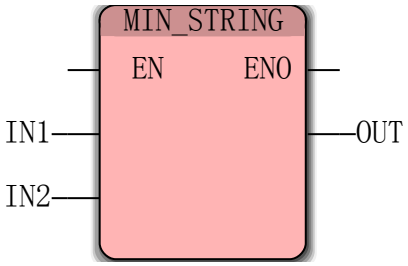
输入和输出	数据类型	描述
IN1	STRING	第一个输入字符或字符串
IN2	STRING	第二个输入字符或字符串
OUT	STRING	输出，两个字符或字符串中较大的一个； 字符的大小均按其 ASCII 码计算；当输入字符串时，仅第一个字符参与比较，即如果一个字符串的第一个字符较大，则本字符串全部输出

5.4.9 取较小的字符串——MIN_STRING

功能

MIN_STRING 指令用于确定两个字符中的较小的一个。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 MIN_STRING IN2 ST OUT	
ST 编程语言	
OUT := MIN_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1，IN2 和 OUT 或使用常量	

MIN_STRING 指令处理的数据类型

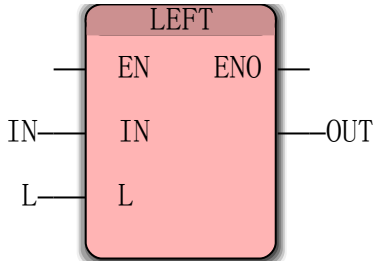
输入和输出	数据类型	描述
IN1	STRING	第一个输入字符或字符串
IN2	STRING	第二个输入字符或字符串
OUT	STRING	输出，两个字符或字符串中较小的一个； 字符的大小均按其 ASCII 码计算；当输入字符串时，仅第一个字符参与比较，即如果一个字符串的第一个字符较小，则本字符串全部输出

5.4.10 取出字符串最左边的几个字符——LEFT

功能

LEFT 指令用于取出输入字符串中最左边的几个连续的字符。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN LEFT L ST OUT	
ST 编程语言	
OUT := LEFT (IN, L) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN，L 和 OUT 或使用常量	

LEFT 指令处理的数据类型

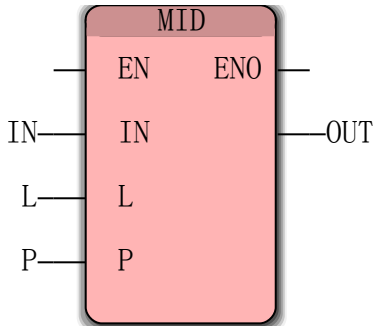
输入和输出	数据类型	描述
IN	STRING	输入字符串
L	ANY_INT	要取出的字符数
OUT	STRING	输出，取出输入字符串中从最左侧开始的 L 个连续的字符； L 必须大于 0，且小于等于输入字符串 IN 的字符个数

5.4.11 取出字符串中的几个字符——MID

功能

MID 指令用于取出输入字符串中最左边的几个连续的字符。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN MID L, P ST OUT	
ST 编程语言	
OUT := MID (IN, L, P) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN，L、P 和 OUT 或使用常量	

MID 指令处理的数据类型

输入和输出	数据类型	描述

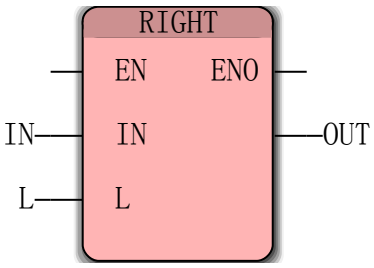
IN	STRING	输入字符串
L	ANY_INT	要取出的字符数
P	ANY_INT	要取出的字符的起始位置，即从输入字符串的第 P 个字符开始取字符
OUT	STRING	输出，取出输入字符串中从第 P 个至 P+L 个连续的字符； 注：L 必须大于 0；第 P+L 个字符必须在输入字符串中；P 必须大于 0 且小于等于输入字符串的最大字符数；IN 必须为非空字符串

5.4.12 取出字符串最右边的几个字符——RIGHT

功能

RIGHT 指令用于取出输入字符串中最右边的几个连续的字符。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN RIGHT L ST OUT	
ST 编程语言	
OUT := RIGHT (IN, L) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN，L 和 OUT 或使用常量	

RIGHT 指令处理的数据类型

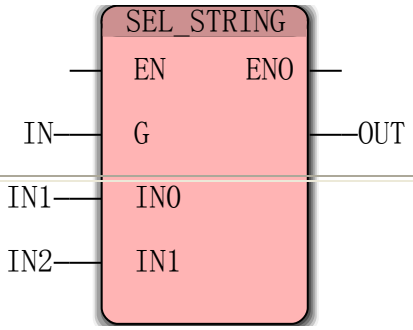
输入和输出	数据类型	描述
IN	STRING	输入字符串
L	ANY_INT	要取出的字符数
OUT	STRING	输出，取出输入字符串中从最右侧开始的 L 个连续的字符； L 必须大于 0，且小于等于输入字符串 IN 的字符个数

5.4.13 字符串的二进制选择——SEL_STRING

功能

SEL_STRING 指令用于取出输入字符串中最左边的几个连续的字符。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN SEL_STRING IN1, IN2 ST OUT	
ST 编程语言	

OUT := SEL_STRING (IN, IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN，IN1、IN2 和 OUT 或使用常量	

SEL_STRING 指令处理的数据类型

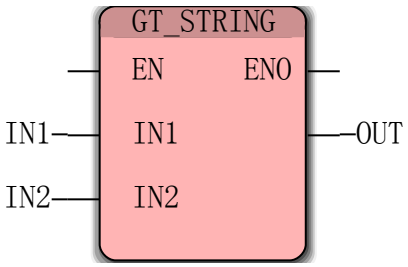
输入和输出	数据类型	描述
IN（G）	BOOL	选择输入端
IN1（IN0）	STRING	第一个输入
IN2（IN1）	STRING	第二个输入
OUT	STRING	输出 如果 IN=0，OUT=IN1；如果 IN=1，OUT=IN2

5.4.14 字符串大于——GT_STRING

功能

GT_STRING 指令用于确定第一个字符串是否大于第二个字符串。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 GT_STRING IN2 ST OUT	
ST 编程语言	
OUT := GT_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

GT_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串的第一个字符大于第二个输入字符串的第一个字符，则认为 IN1>IN2，OUT=1；否则 OUT=0；字符的大小均按其 ASCII 码计算

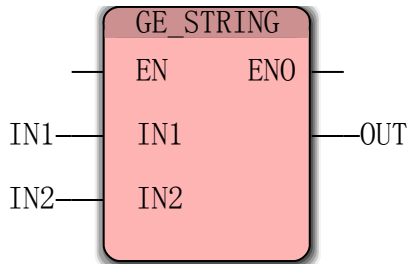
5.4.15 字符串大于等于——GE_STRING

功能

GE_STRING 指令用于确定第一个字符串是否大于等于第二个字符串。

用法

IL 编程语言	LD、 FBD 编程语言

LD IN1 GE_STRING IN2 ST OUT	
ST 编程语言	
OUT := GE_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

GE_STRING 指令处理的数据类型

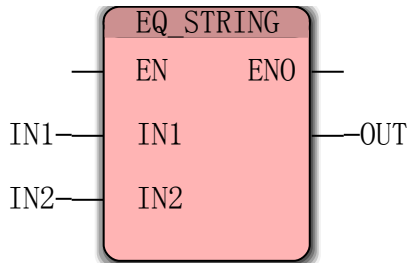
输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串的第一个字符大于或等于第二个输入字符串的第一个字符，则认为 IN1>=IN2，OUT=1；否则 OUT=0；字符的大小均按其 ASCII 码计算

5.4.16 字符串等于——EQ_STRING

功能

EQ_STRING 指令用于确定第一个字符串是否等于第二个字符串。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 EQ_STRING IN2 ST OUT	
ST 编程语言	
OUT := EQ_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

EQ_STRING 指令处理的数据类型

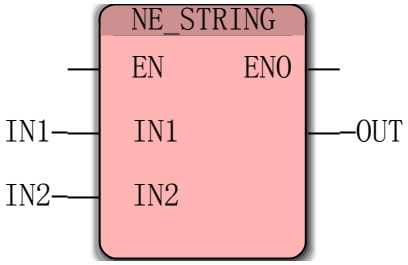
输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串与第二个输入字符串完全一样，则认为 IN1=IN2，OUT=1；否则 OUT=0

5.4.17 字符串不等于——NE_STRING

功能

NE_STRING 指令用于确定第一个字符串是否不等于第二个字符串。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 NE_STRING IN2 ST OUT	
ST 编程语言	
OUT := NE_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

NE_STRING 指令处理的数据类型

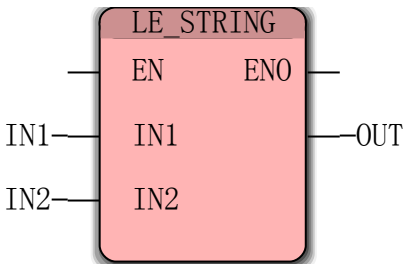
输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串与第二个输入字符串不一样，则 OUT=1；否则 OUT=0；本指令与 EQ_STRING 相反

5.4.18 字符串小于等于——LE_STRING

功能

LE_STRING 指令用于确定第一个字符串是否小于等于第二个字符串。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 LE_STRING IN2 ST OUT	
ST 编程语言	
OUT := LE_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

LE_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN1	STRING	第一个输入

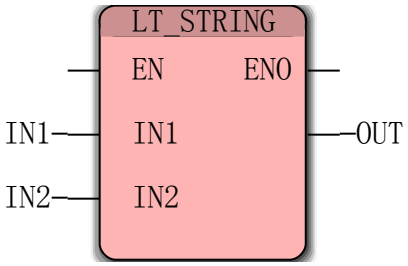
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串的第一个字符小于或等于第二个输入字符串的第一个字符，则认为 IN1<=IN2，OUT=1；否则 OUT=0；字符的大小均按其 ASCII 码计算

5.4.19 字符串小于——LT_STRING

功能

LT_STRING 指令用于确定第一个字符串是否小于第二个字符串。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 LT_STRING IN2 ST OUT	
ST 编程语言	
OUT := LT_STRING (IN1, IN2) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN2 和 OUT 或使用常量	

LT_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN1	STRING	第一个输入
IN2	STRING	第二个输入
OUT	BOOL	输出，如果第一个输入字符串的第一个字符小于第二个输入字符串的第一个字符，则认为 IN1<IN2，OUT=1；否则 OUT=0；字符的大小均按其 ASCII 码计算

5.4.20 字符串转换为其它类型——STRING_TO_*

STRING_TO_*指令用于将一个 STRING 型数据转换一个有效的 ANY 数值类型，有 12 个指令，分别是将 STRING 型数据转换为 BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL、LREAL 和 TIME 等类型。

STRING_TO_*转换指令

指令	输入值	输出值	描述
STRING_TO_BYTE	STRING	BYTE	输入格式 0~255*，输出为 BYTE 类型
STRING _TO_WORD	STRING	WORD	输入格式 0~65535*，输出为 WORD 类型
STRING _TO_DWORD	STRING	DWORD	输入格式 0~4,294,967,295*，输出为 DWORD 类型
STRING _TO_SINT	STRING	SINT	输入格式-128~127*，输出为 SINT 类型

STRING _TO_INT	STRING	INT	输入格式-32768~-32767*，输出为 INT 类型
STRING _TO_DINT	STRING	DINT	输入格式-2,147,483,648~2,147,483,647*，输出 DINT
STRING _TO_USINT	STRING	USINT	输入格式 0~255*，输出为 USINT 类型
STRING _TO_UINT	STRING	UINT	输入格式 0~65535*，输出为 UINT 类型
STRING _TO_UDINT	STRING	UDINT	输入格式 0~4,294,967,295*，输出为 UDINT 类型
STRING _TO_REAL	STRING	REAL	输入格式小数*，输出为 REAL 类型
STRING_TO_LREAL	STRING	LREAL	输入格式小数*，输出为 LREAL 类型
STRING_TO_TIME	STRING	TIME	输入格式 T#0~T#4,294,967,295*毫秒，输出秒
*：除 STRING_TO_TIME 指令外，可带对应的 BYTE#、WORD#等类型前缀和“16#”前缀，也可不带前缀；十进制时输入字符有效值为正负号和数字 0~9；数字后出现的其它字符以及再出现的数字均无效。			

用法（以 STRING_TO_BYTE 为例）

输入和输出	数据类型	描述
IN	STRING	输入字符或字符串，输入格式 0~255
OUT	BYTE	输出，把输入转换为 BYTE 类型

在数值转换过程中，可能会发生如下错误：

- 输入带有非法前缀，如 INT#123，INT#是非法的前缀；
- 输入值超出输出数据类型的范围，如 STRING_TO_BYTE 指令的输入值 1024，1024 大于 BYTE 类型的范围 0~255。

5.4.21 其它类型转换为字符串—— *_TO_STRING

*_TO_STRING 分为 12 个指令，可分别将 BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL、LREAL 和 TIME 等类型转换为 STRING 类型。

用法（以 BYTE_TO_STRING 为例）

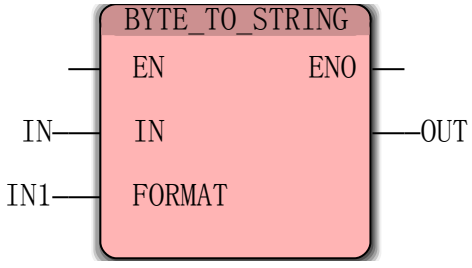
IL 编程语言

```
LD IN
BYTE_TO_STRING FORMAT
ST OUT
```

ST 编程语言

```
OUT := BYTE_TO_STRING (IN, IN1) ;
```

LD、 FBD 编程语言



BYTE_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	BYTE	输入范围 0~255
FORMAT	STRING	有效的格式：%c、%x、%u，默认%x
OUT	STRING	输出，格式为%c 时，输出为输入的 ASCII 码；格式为%x 时，输出为

		输入的十六进制数；格式为%u 时，输出为输入的无符号十进制数

WORD_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	WORD	输入范围 0~65535
IN1（FORMAT）	字符串	有效的格式：%x、%u，默认%x
OUT	STRING	输出，格式为%x 时，输出为输入的十六进制数；格式为%u 时，输出为输入的无符号十进制数

DWORD_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	DWORD	输入范围 0~4,294,967,295
IN1（FORMAT）	字符串	有效的格式：%x、%u，默认%x
OUT	STRING	输出，格式为%x 时，输出为输入的十六进制数；格式为%u 时，输出为输入的无符号十进制数

SINT_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	SINT	输入范围-128~127
IN1（FORMAT）	字符串	有效的格式：%d，默认%d
OUT	STRING	输出，输出为输入的有符号十进制数

INT_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	INT	输入范围-32768~32767
IN1（FORMAT）	字符串	有效的格式：%d，默认%d
OUT	STRING	输出，输出为输入的有符号十进制数

DINT_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	DINT	输入范围-2,147,483,648~2,147,483,647
IN1（FORMAT）	字符串	有效的格式：%d，默认%d
OUT	STRING	输出，输出为输入的有符号十进制数

USINT_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	USINT	输入范围 0~255
FORMAT	字符串	有效的格式：%u，默认%u
OUT	STRING	输出，输出为输入的符号十进制数

UINT_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	UINT	输入范围 0~65535

IN1（FORMAT）	字符串	有效的格式：%u，默认%u
OUT	STRING	输出，输出为输入的无符号十进制数

DINT_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	UDINT	输入范围 0~4,294,967,295
IN1（FORMAT）	字符串	有效的格式：%u，默认%u
OUT	STRING	输出，输出为输入的无符号十进制数

REAL_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	REAL	输入范围-3.402823466 E+38… -1.175494351 E-38 以及+1.175494351 E-38… +3.402823466 E+38
IN1（FORMAT）	字符串	有效的格式：%e、%f，默认%e
OUT	STRING	输出，格式为%e 时，输出为输入的科学计数法表示的浮点数；格式为%f 时，输出为输入的浮点数

LREAL_TO_STRING 指令处理的数据类型与 REAL_TO_STRING 相同，只是输入范围不同。

TIME_TO_STRING 指令处理的数据类型

输入和输出	数据类型	描述
IN	TIME	输入 TIME 类型，范围 T#0~T#4,294,967,295*毫秒，如 T#1MS
IN1（FORMAT）	字符串	有效的格式：%u，默认%u
OUT	STRING	输出，格式为%u 时，输出是将输入表示为不带前缀的 TIME 类型，单位毫秒，如输入 T#1S，输出 1000；如不指定格式，输出是将输入表示为带前缀的 TIME 类型，如输入 T#1S，输出为 T#1000MS

5.5 位操作函数 BIT_UTIL

位操作函数功能是带有多个输入参数和一个输出参数的程序组织单元 POU，能对输入的位串做读写操作，位操作函数的缩写为 BIT_UTIL。

位操作函数需要单独插入固件库。

位操作函数包含的指令（在编辑向导中，从下拉列表选择“BIT_UTIL”）

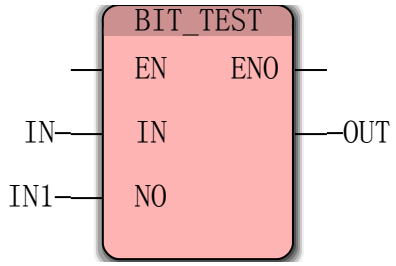
BIT_TEST	GET_CHAR	GET_LSB	GET_MSB
I_BIT_IN_BYTE	I_BIT_IN_WORD	I_BIT_IN_DWORD	PARITY_BYTE
PARITY_WORD	PARITY_DWORD	R_BIT_IN_BYTE	R_BIT_IN_WORD
R_BIT_IN_DWORD	S_BIT_IN_BYTE	S_BIT_IN_WORD	S_BIT_IN_DWORD
SET_LSB	SET_MSB	STRING_TO_BUFFER	SWAP

5.5.1 读取位串中的位值——BIT_TEST 指令

功能

BIT_TEST 指令用于读取输入位串中的单个位的值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN BIT_TEST IN1 ST OUT	
ST 编程语言	
OUT := BIT_TEST(IN, IN1) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、IN1 和 OUT 或使用常量	

BIT_TEST 指令处理的数据类型

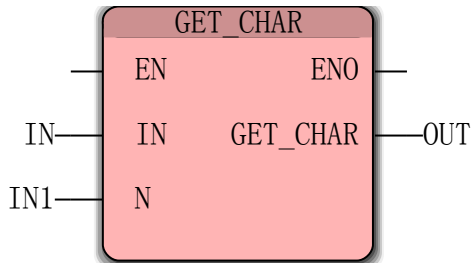
输入和输出	数据类型	描述
IN（IN）	ANY_BIT	输入位串
IN1（NO）	SINT	要读取的位串中的第 NO 个位，取值范围：BYTE0~7，WORD0~15，DWORD0~31
OUT	BOOL	输出，输入位串中第 NO 个位的值

5.5.2 取出字符串中的字符——GET_CHAR 指令

功能

GET_CHAR 指令用于从输入字符串中取得一个字符，并 ASCII 码表示该字符。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN GET_CHAR IN1 ST OUT	
ST 编程语言	
OUT := GET_CHAR(IN, IN1) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、IN1 和 OUT 或使用常量	

GET_CHAR 指令处理的数据类型

输入和输出	数据类型	描述
IN（IN）	ANY_BIT	输入字符串
IN1（N）	INT	要去出的字符串中的第 N 个字符，取值范围 0~32767
OUT（GET_CHAR）	INT	输出，输入字符串中第 N 个字符的 ASCII 码值

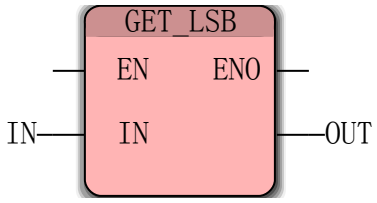
注：目前的软件版本还不支持这个指令。

5.5.3 取出位串中的低 8 位——GET_LSB 指令

功能

GET_LSB 指令用于读取输入位串中的较低字节(the Less Significant BYTE)的值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN GET_LSB ST OUT	
ST 编程语言	
OUT := GET_LSB(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、和 OUT 或使用常量	

GET_LSB 指令处理的数据类型

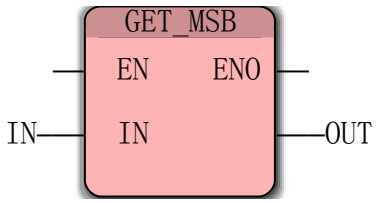
输入和输出	数据类型	描述
IN	WORD	输入位串
OUT	BYTE	输出，较低字节（低 8 位）的值

5.5.4 取出位串中的高 8 位——GET_MSB 指令

功能

GET_MSB 指令用于读取输入位串中的最高字节(the Most Significant BYTE)的值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN GET_MSB ST OUT	
ST 编程语言	
OUT := GET_MSB(IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、和 OUT 或使用常量	

GET_MSB 指令处理的数据类型

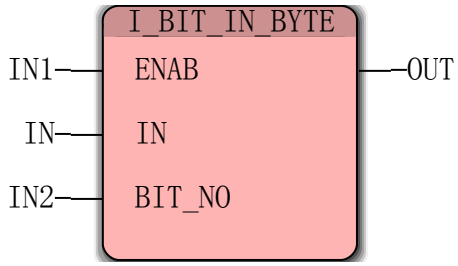
输入和输出	数据类型	描述
IN	WORD	输入位串
OUT	BYTE	输出，最高字节（高 8 位）的值

5.5.5 将位串中单个位取反——I_BIT_IN_*指令

功能

I_BIT_IN_*指令用于将输入位串中的单个位取反，包括 I_BIT_IN_BYTE、I_BIT_IN_WORD 和 I_BIT_IN_DWORD 指令，能够处理 BYTE、WORD 和 DWORD 类型的输入位串。

用法（以 I_BIT_IN_BYTE 为例）

IL 编程语言	LD、 FBD 编程语言
LD ENAB I_BIT_IN_BYTE IN, BIT_NO ST OUT **ST 编程语言** OUT := I_BIT_IN_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN、IN2 和 OUT 或使用常量	

I_BIT_IN_BYTE 指令处理的数据类型

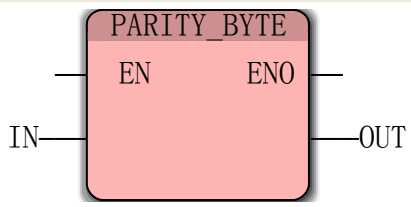
输入和输出	数 据 类 型	描述
IN1（ENAB）	BOOL	使能
IN	BYTE WORD DWORD	输入位串
IN2（BIT_NO）	SINT	要操作的位串中的第 NO 个位，取值范围：BYTE 时 0~7，WORD 时 0~15，DWORD 时 0~31（其它值无效）
OUT	BYTE	输出，当 IN1 为 FALSE 时，输出 OUT 等于输入 IN；当 IN1 为 TRUE 时，输出 OUT 是将输入 IN 的第 NO 个位取反后的值

5.5.6 位串的奇偶校验——PARITY_*指令

功能

PARITY_*指令用于检查输入位串中为 1 的位的数量是奇数还是偶数个时，如果是奇数则输出为 FALSE，如果为偶数则输出为 TRUE，PARITY_*指令包括 PARITY_BYTE、PARITY_WORD 和 PARITY_DWORD 指令，能够处理 BYTE、WORD 和 DWORD 类型的输入位串。

用法（以 PARITY_BYTE 为例）

IL 编程语言	LD、 FBD 编程语言
LD IN PARITY_BYTE ST OUT **ST 编程语言** OUT := PARITY_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

PARITY_BYTE 指令处理的数据类型

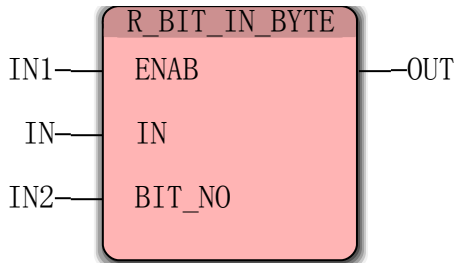
输入和输出	数据类型	描述
IN	BYTE WORD DWORD	输入位串
OUT	BOOL	输出，当输入位串中为 1 的位的数量是奇数个时，输出为 FALSE；为 1 的位的数量是偶数个时（包括 0 个为 1 的位），输出为 TRUE

5.5.7 将位串中单个位置 0——R_BIT_IN_*指令

功能

R_BIT_IN_*指令用于将输入位串中的单个位置 0，包括 R_BIT_IN_BYTE、R_BIT_IN_WORD 和 R_BIT_IN_DWORD 指令，能够处理 BYTE、WORD 和 DWORD 类型的输入位串。

用法（以 R_BIT_IN_BYTE 为例）

IL 编程语言	LD、 FBD 编程语言
LD ENAB R_BIT_IN_BYTE IN, BIT_NO ST OUT	
ST 编程语言	
OUT := R_BIT_IN_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN、IN2 和 OUT 或使用常量	

R_BIT_IN_BYTE 指令处理的数据类型

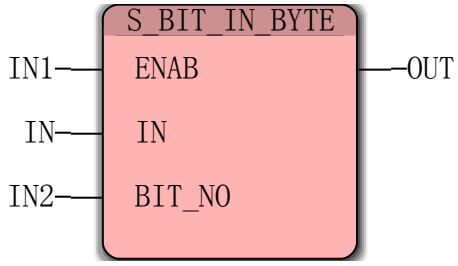
输入和输出	数据类型	描述
IN1（ENAB）	BOOL	使能
IN	BYTE WORD DWORD	输入位串
IN2（BIT_NO）	SINT	要操作的位串中的第 NO 个位，取值范围：BYTE 时 0~7，WORD 时 0~15，DWORD 时 0~31（其它值无效）
OUT	BYTE	输出，当 IN1 为 FALSE 时，输出 OUT 等于输入 IN；当 IN1 为 TRUE 时，输出 OUT 是将输入 IN 的第 NO 个位置 0 后的值

5.5.8 将位串中单个位置 1——S_BIT_IN_*指令

功能

S_BIT_IN_*指令用于将输入位串中的单个位置 1，包括 S_BIT_IN_BYTE、S_BIT_IN_WORD 和 S_BIT_IN_DWORD 指令，能够处理 BYTE、WORD 和 DWORD 类型的输入位串。

用法（以 S_BIT_IN_BYTE 为例）

IL 编程语言	LD、 FBD 编程语言
LD ENAB S_BIT_IN_BYTE IN, BIT_NO ST OUT	
ST 编程语言	
OUT := S_BIT_IN_BYTE (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN、IN2 和 OUT 或使用常量	

S_BIT_IN_BYTE 指令处理的数据类型

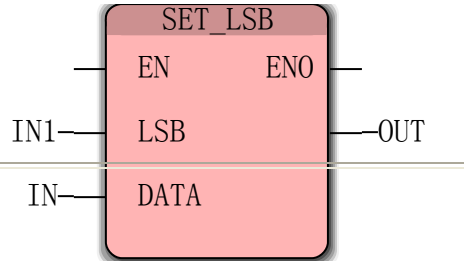
输入和输出	数据类型	描述
IN1（ENAB）	BOOL	使能
IN	BYTE WORD DWORD	输入位串
IN2（BIT_NO）	SINT	要操作的位串中的第 NO 个位，取值范围：BYTE 时 0~7，WORD 时 0~15，DWORD 时 0~31（其它值无效）
OUT	BYTE	输出，当 IN1 为 FALSE 时，输出 OUT 等于输入 IN；当 IN1 为 TRUE 时，输出 OUT 是将输入 IN 的第 NO 个位置 1 后的值

5.5.9 向位串中的低 8 位写数——SET_LSB 指令

功能

SET_LSB 指令用于向输入位串中的较低字节(the Less Significant BYTE)写数值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 SET_LSB IN ST OUT	
ST 编程语言	

OUT := SET_LSB(IN1, IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN 和 OUT 或使用常量	

SET_LSB 指令处理的数据类型

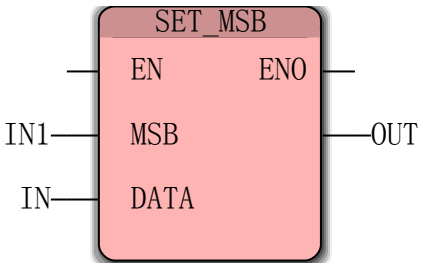
输入和输出	数据类型	描述
IN1（LSB）	BYTE	要写入的数值，范围 0~255
IN（DATA）	WORD	输入位串，0~65535
OUT	WORD	输出，将输入位串的较低字节（低 8 位）改为 IN1

5.5.10 向位串中的高 8 位写数——SET_MSB 指令

功能

SET_MSB 指令用于向输入位串中的最高字节(the Most Significant BYTE)写数值。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN1 SET_MSB IN ST OUT	
ST 编程语言	
OUT := SET_MSB(IN1, IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN1、IN 和 OUT 或使用常量	

SET_MSB 指令处理的数据类型

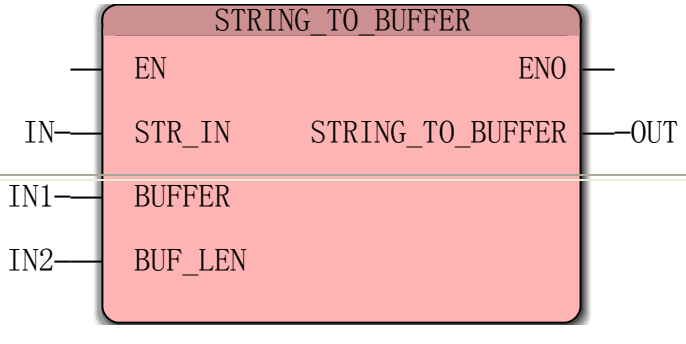
输入和输出	数据类型	描述
IN1（MSB）	BYTE	要写入的数值，范围 0~255
IN（DATA）	WORD	输入位串，0~65535
OUT	WORD	输出，将输入位串的最高字节（高 8 位）改为 IN1

5.5.11 复制字符串到缓冲区——STRING_TO_BUFFER 指令

功能

STRING_TO_BUFFER 指令用于将字符串复制到缓冲区，缓冲区为一个字节数组。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN STRING_TO_BUFFER IN1, IN2 ST OUT	
ST 编程语言	

OUT := STRING_TO_BUFFER (IN1, IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、IN1、IN2 和 OUT 或使用常量	

STRING_TO_BUFFER 指令处理的数据类型

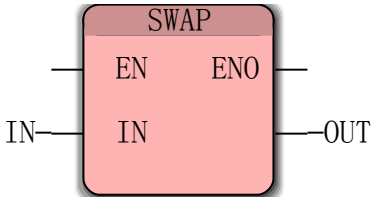
输入和输出	数据类型	描述
IN（STR_IN）	STRING	输入字符串
IN1[0]（BUFFER）	BYTE	缓冲区（字节数组）的一个元素，如 IN1[0]或 IN1[1]
IN2（BUF_LEN）	INT	拷贝到缓冲区中的字符数
OUT（STRING_TO_BUFFER）	INT	输出，暂没有定义 ，实际上字符是被拷贝到缓冲区 IN1 中了，IN1 中的一个元素（字节）存放着被拷贝字符的 ASCII 码。

5.5.12 交换高字节和低字节——SWAP 指令

功能

SWAP 指令用于交换输入位串的最高字节(MSB)和较低字节(LSB)的位置。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN SWAP ST OUT	
ST 编程语言	
OUT := SWAP (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

SWAP 指令处理的数据类型

输入和输出	数据类型	描述
IN（IN）	WORD	输入位串
OUT（STRING_TO_BUFFER）	WORD	输出位串，将输入位串的高低字节交换后输出

5.6 ProConOS 功能

ProConOS 中的 FILE_OPEN、FILE_CLOSE、FILE_SEEK、FILE_TELL、FILE_READ、FILE_WRITE、FILE_REMOVE 指令，不可用。如需文件读写，请使用腾控固件库 FileAccess，见 7.1 节。

5.6.1 BUF 型转换为其它类型

BUF 型数据可分为 12 个功能块，可将基本数据类型从字节流复制到变量、数组或者用户自定义结构的元素中，即分别转换为 BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL、STRING、TIME 等数据，它主要用于数据传送或者在不同硬件平台上执行应用中的通讯。

ProConOS 中的 BUF 型数据转换指令需要在编辑向导中，从下拉列表选择“ProConOS”。

源数据必须为字节数组（数据类型为 BYTE），也可是 ANY_BIT（BOOL 除外）或 ANY_INT 型（被转换的字节数不能超出存放目标的数据类型）。

BUF 型数据的转换指令

指令	源数据	转换后数据	描述	
BUF_TO_BYTE	BUF	BYTE	缓冲区的数据格式为 MOTOROLA 或 INTEL，根据需要，源数据也可以是 ANY_BIT（BOOL 除外）或 ANY_INT	被转换的字节数是 1
BUF_TO_WORD	BUF	WORD		被转换的字节数是 2
BUF_TO_DWORD	BUF	DWORD		被转换的字节数是 4
BUF_TO_SINT	BUF	SINT		被转换的字节数是 1
BUF_TO_INT	BUF	INT		被转换的字节数是 2
BUF_TO_DINT	BUF	DINT		被转换的字节数是 4
BUF_TO_USINT	BUF	USINT		被转换的字节数是 1
BUF_TO_UINT	BUF	UINT		被转换的字节数是 2
BUF_TO_UDINT	BUF	UDINT		被转换的字节数是 4
BUF_TO_REAL	BUF	REAL	缓冲区的数据格式为 MOTOROLA	输入为 IEEE 754 浮点数的十六进制码，被转换的字节数可以是 4
BUF_TO_STRING	BUF	STRING	缓冲区的数据格式为 MOTOROLA 或 INTEL	输入为字符的 ASCII 码，被转换的字节数可以是 1、2…等正整数
BUF_TO_TIME	BUF	TIME	缓冲区的数据格式为 INTEL	输入为以毫秒为单位的十六进制码，被转换的字节数是 4，输出单位为秒

BUF_TO_*指令处理的数据类型

引脚		数据类型	描述
REQ	输入	BOOL	上升沿有效
BUF_FORMAT	输入	BOOL	TRUE 表示缓冲区数据为 MOTOROLA 格式；FALSE 表示 INTEL 格式
BUF_OFFS	输入	DINT	缓冲区中被转换的起始字节序号，0 表示缓冲区第一个字节
BUF_CNT	输入	DINT	缓冲区中被转换的字节数
BUFFER	输入-输出	ARRAY	缓冲区，为一个字节数组

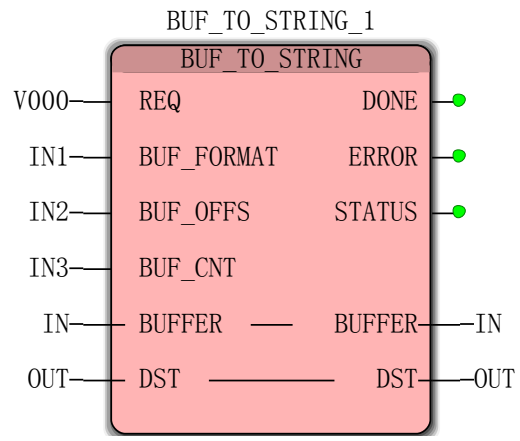
DST	输入-输出	数组或 ANY	存放区，被转换的字节内容放在这里，DST 的类型应与具体的 BUF_TO_*类型一致，如 BUF_TO_BYTE 指令的 DST 必须是 BYTE 类型
DONE	输出	BOOL	转换完成后，置 1，直到 REQ 为 0
ERROR	输出	BOOL	转换正常，为 0，否则置 1
STATUS	输出	INT	如果转换不正常，则给出错误代码，具体见下表

注：MOTOROLA 和 INTEL 微处理器的数据存放顺序不同，INTEL 格式为高低字节排列，MOTOROLA 格式为低高字节排列。

错误代码	描述
0	转换过程正常完成
1	BUFFER 和 DST 的输出-输出类型错误
2	超出缓冲区的长度，即将被复制的字节数 BUF_CNT 大于缓冲区 BUFFER 的可用字节数
3	超出存放区的长度，即将被复制的字节数 BUF_CNT 超出存放区长度
4	不支持这个数据类型
5	将要转换的字节长度与存放区的字节长度不对应，前者字节数必须能被后者的字节数整除
6	转换 INTEL / MOTOROLA 失败
7	字符串长度不适当，对于数据类型串，有必要做额外的检查
8	存放区数据类型错误
9	BUF_OFFS 数值不正确
10	BUF_CNT 数值不正确
11	缓冲区与存放区地址相同

注：缓冲区 BUFFER 的可用字节数——在缓冲区中从第 BUF_OFFS 个字节开始至最后一个字节

用法（以 BUF_TO_STRING 为例）

IL 编程语言	LD、 FBD 编程语言
见帮助文件	
ST 编程语言	
见帮助文件	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量

5.6.2 其它类型转换为 BUF 型

其它类型可转换为 BUF 型数据，可将变量、数组或者用户自定义结构的元素中的基本数据类型复制到字节流中，共有 12 个指令，分别将 BYTE、WORD、DWORD、SINT、INT、DINT、USINT、UINT、UDINT、REAL、STRING、TIME 等数据转换到 BUF 类型数据中，它主要用于数据传送或者在不同硬件平台上执行应用中的通讯。

ProConOS 中的其它类型转换为 BUF 型指令需要在编辑向导中，从下拉列表选择“ProConOS”。

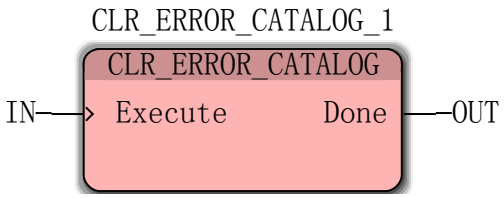
存放区必须为字节数组（数据类型为 BYTE），也可是 ANY_BIT（BOOL 除外）或 ANY_INT 型。这些指令的用法与上述“BUF 型转换为其它类型”相似，这里不再描述。

5.6.3 删除完整的错误目录 CLR_ERROR_CATALOG

功能

CLR_ERROR_CATALOG 指令用于删除完整的错误目录。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN CLR_ERROR_CATALOG ST OUT	
ST 编程语言	
OUT := CLR_ERROR_CATALOG (IN) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

CLR_ERROR_CATALOG 指令处理的数据类型

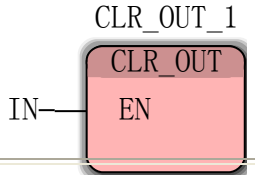
输入和输出	数据类型	描述
IN（Execute）	BOOL	上升沿有效
OUT（Done）	BOOL	0: 不能够删除错误目录。 1: 成功删除错误目录。

5.6.4 将 I/O 映像的输出设置为 0 指针 CLR_OUT

功能

CLR_OUT 指令用于将 I/O 映像区的输出设置为 0。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN ST INCLR_OUT_1.EN CAL CLR_OUT_1	
ST 编程语言	

CLR_OUT_1 (EN := (IN));	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 或使用常量	

CLR_OUT 指令处理的数据类型

输入和输出	数据类型	描述
IN（Execute）	BOOL	如果为 TRUE，则将 I/O 映像的所有输出都设置为零

注：CLR_OUT 指令暂时不能用。

5.6.5 PLC 冷启动 COLD_RESTART

功能

COLD_RESTART 指令用于冷启动 PLC。

在冷再启动过程中，初始化所有数据。如程序发生堆栈溢出、字符串错误或者被 0 除等问题时，可以调用这个指令自动重新启动程序的执行。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN COLD_RESTART ST OUT	
ST 编程语言	
COLD_RESTART_1 (IN);	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

COLD_RESTART 指令处理的数据类型

输入和输出	数据类型	描述
IN（COND）	BOOL	如果为 TRUE，则执行冷再启动
OUT	BOOL	如果 COND=TRUE，且可以冷启动，则 OUT 为 TRUE

注：COLD_RESTART 指令暂时不能用。

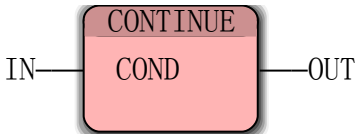
5.6.6 继续运行程序 CONTINUE

功能

CONTINUE 指令用于程序出现暂时错误时（如定时器错误），继续运行其它程序。

这个功能不应该用于被 0 除、栈溢出、总线错误和界限错误等异常事件。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN CONTINUE ST OUT	
ST 编程语言	

CONTINUE (IN);	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 和 OUT 或使用常量	

CONTINUE 指令处理的数据类型

输入和输出	数据类型	描述
IN（COND）	BOOL	如果为 TRUE，则继续程序的执行。
OUT	BOOL	如果 COND=TRUE，并且可以执行程序，则为 TRUE

注：CONTINUE 指令暂时不能用。

5.6.7 微分 DERIVAT

功能

DERIVAT 指令用于对数据做时间的微分计算，使用微分指令时，需要把 POU 所在的任务类型设置为周期扫描（CYCLIC），并根据自己的需要设置扫描周期，任务类型的设置和扫描周期见本手册编程模型->硬件->Tasks。

用法

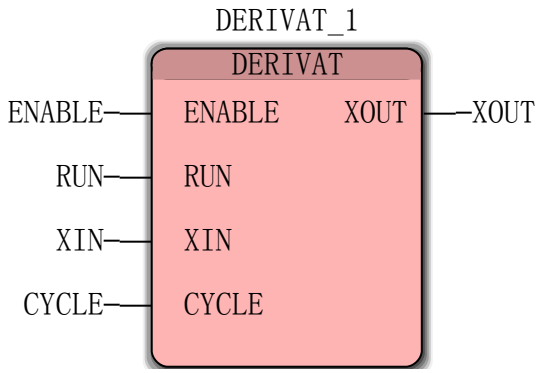
IL 编程语言

```
LD   ENABLE
ST   DERIVAT_1.ENABLE
LD   RUN
ST   DERIVAT_1.RUN
LD   XIN
ST   DERIVAT_1.XIN
LD   CYCLE
ST   DERIVAT_1.CYCLE
CALDERIVAT_1
LD   DERIVAT_1.XOUT
ST   XOUT
```

ST 编程语言

```
DERIVAT_1 ( ENABLE := IN, RUN :=
IN1, XIN := IN2, CYCLE := T );
OUT := DERIVAT_1.XOUT;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需在当前 POU 的变量工作单中插入变量 ENABLE、RUN 等或使用常量

DERIVAT 指令处理的数据类型

输入和输出	数据类型	描述
ENABLE	BOOL	TRUE 时，执行功能块
RUN	BOOL	FALSE 时，功能块暂停，并且输出被置为 0
XIN	REAL	输入值
CYCLE	TIME	时间常数，单位秒，本指令实际上是对[XIN(n)-XIN(n-1)]/CYCLE 的微分

XOUT	REAL	输出值，微分结果

5.6.8 触发事件 EVENT_TASK

功能

EVENT_TASK 指令用于触发一个事件任务，事件编号的可以定义，如果用户的一个事件任务存在该事件编号，那么它被激活，即程序被指派给该事件任务。

用法

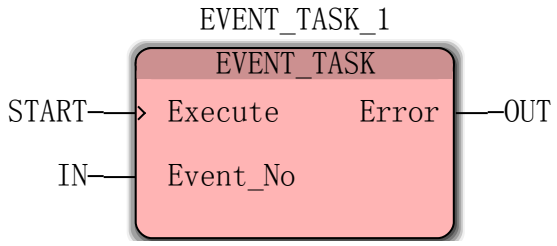
IL 编程语言

```
LD   START
ST   EVENT_TASK_1.Execute
LD   IN
ST   EVENT_TASK_1.Event_No
CALEVENT_TASK_1
LD   EVENT_TASK_1.Erroe
ST   OUT
```

ST 编程语言

```
EVENT_TASK_1 ( Execute := START,
Event_No := IN );
OUT := EVENT_TASK_1.Error;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需在当前 POU 的变量工作单中插入变量 START、IN 和 OUT 或使用常量

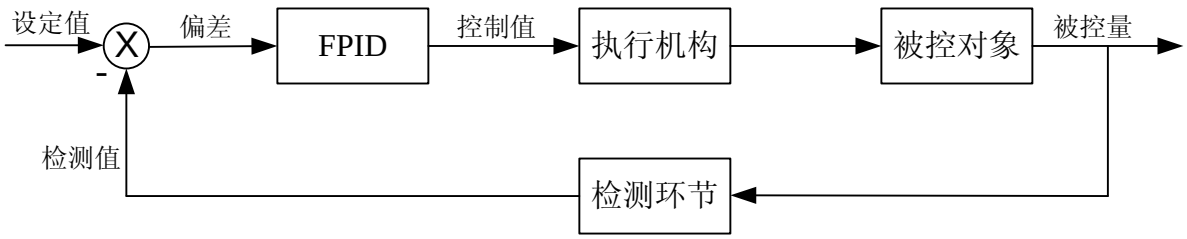
EVENT_TASK 指令处理的数据类型

输入	数据类型	描述
START（Execute）	BOOL	上升沿有效
IN（Event_No）	UINT	事件编号
OUT（Error）	BOOL	错误码: 0—— 没有错误产生； 1——事件编号不在值的有效范围内

5.6.9 FPID

功能

FPID 指令根据设定值与检测值之间的偏差自动计算控制值，使得检测值能够跟踪设定值，设定值是期望被控设备的一个行为保持的一个数值，检测值是通过仪表对被控设备的行为检测出的数值，而控制值是通过控制被控设备的这个行为或间接的行为的一个数值，这样就形成了一个闭环控制回路，如下图，FPID 是核心部分。



偏差=设定值-检测值。

FPID 指令在自动工作模式下，其输出值是经 PID 运算后的计算结果，如下

$$输出 = Kp\left(e(t)+\frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right)$$

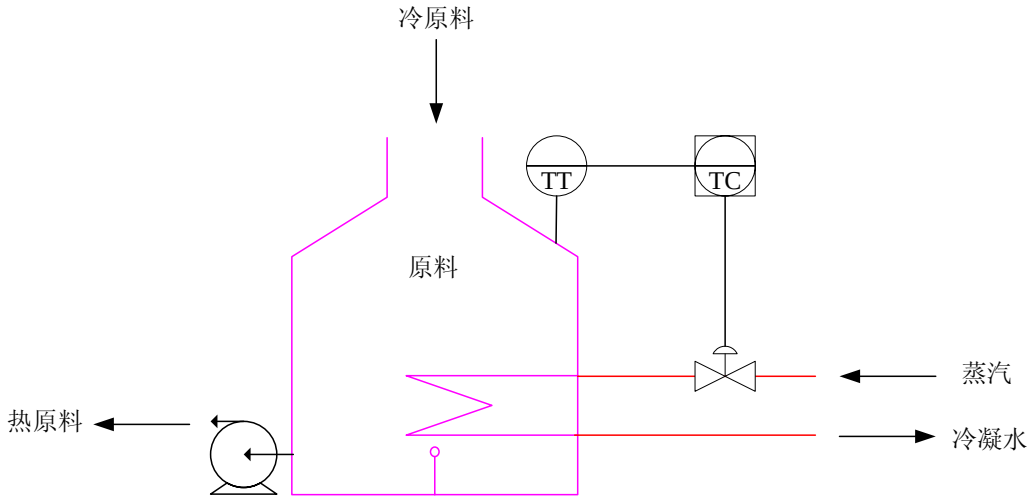
FPID 指令在手动工作模式下，其输出值等于手动输出值

FPID 指令处理的数据类型

输入和输出	类型	描述
REMOTE	BOOL	TRUE 时为远程设定，FALSE 时为本地设定
AUTO	BOOL	TRUE 时 FPID 为自动工作模式，FALSE 时为手动工作模式
DIRECTN	BOOL	TRUE 时为正作用（检测值高于设定值->控制值上升），FALSE 时为反作用（检测值高于设定值->控制值下降）
INTLCK	BOOL	TRUE 时 FPID 的输出 Yout 被强制，强制值通过 FPID 的输入 INTLCKV 设置；FALSE 时输出不强制（输出 Yout 经比例积分微分计算值或手动值）
Tscan	REAL	时间常数，单位秒，一般可设为 REAL#0.1。Tscan 值越大，控制作用越强
Yman	REAL	FPID 在手动工作模式下的输出值
SPR	REAL	远程设定值
SPL	REAL	本地设定值
X	REAL	检测值
KP	REAL	比例，用户可设一个初值，如 6.5
TI	REAL	积分，单位秒，用户可设一个初值，如 60
TD	REAL	微分，单位秒，用户可设一个初值，如 0
HIGH	REAL	输出 Yout 的上限
LOW	REAL	输出 Yout 的下限
INTLCKV	REAL	强制值，当 INTLCK 为 TRUE 时有效
输出	类型	描述
Yout	REAL	输出值

举例

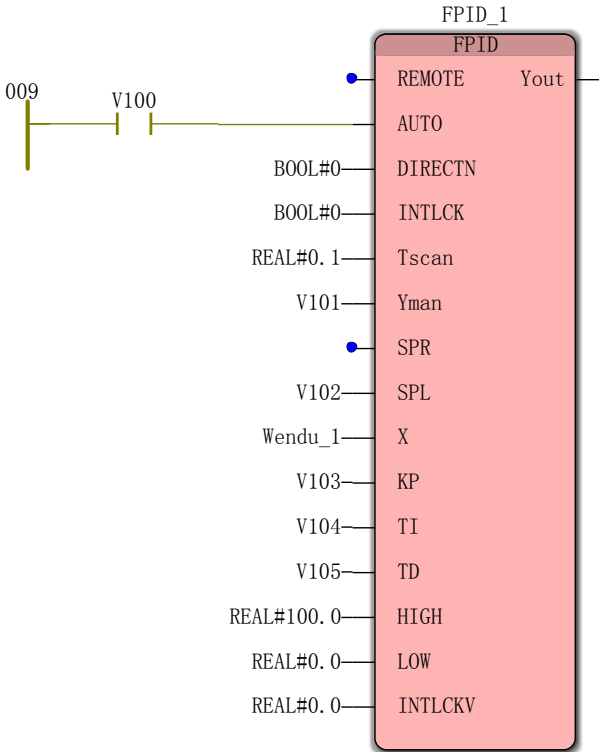
例如使用蒸汽对罐内液体原料加热，就可以使用这个 FPID 构成上述闭环控制回路，如下图



图中，原料连续进入罐内，被罐内的加热器加热并保持恒定温度，然后由泵抽出送至下道工序，高温蒸汽连续通入罐内的加热器，为了实现加热过程的自动化，在蒸汽管路上安装一台调节阀，在罐壁上安装一只温度传感器插入原料中，调节阀控制信号为 4~20mA（对应开度 0~100%），将它连接至 PLC 的模拟量输出通道 AO0（调节阀控制信号的+接 PLC 的 AO0+，-接 AO0-），温度传感器的信号为 4~20mA，将它连接至 PLC 的模拟量输入通道 AI0（温度传感器为两线制，接线见硬件手册）。到此，仪表接线完成了，接下来进行 PLC 的编程，如下

第一步：新建一个工程，并创建一个 LD 程序；

第二步：在“编辑向导”中从下拉列表选择“PROCONOS”，找到 FPID 指令，将其拖拽到编辑区，并为各引脚创建变量，如下图



上图中，引脚 REMOTE 为 0 表示本地设定，引脚 AUTO 通过变量 V100（地址%MX3.0.0）控制自动和手动，引脚 DIRECTN 为 0 表示反作用（即温度高于设定值时关阀），引脚 INTLCK 为 0 表示无强制，引脚 Tscan 为 0.1，引脚 Yman 连接变量 V101 为手动输出值（地

址%MD3.01002），引脚 SPL 连接变量 V102 为本地设定值（地址%MD3.01006），引脚 X 连接变量 Wendu_1 为检测值（地址%MD3.00058），引脚 KP 连接变量 V103 为比例（地址%MD3.01010），引脚 TI 连接变量 V104 为积分（地址%MD3.01014），引脚 TD 连接变量 V105 为微分（地址%MD3.01018），引脚 HIGH 连接常量 REAL#100.0 为输出的上限，引脚 LOW 连接常量 REAL#0.0 为输出的下限，引脚 INTLCKV 连接常量 REAL#0.0 为强制值，引脚 Yout 将连接一个输出转换模块，完成 REAL 到 WORD 的转换。

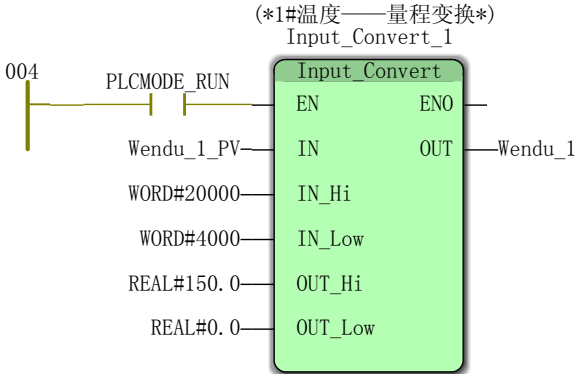
上述 FPID 引脚中有些变量设置了地址并放在中间变量区，为的是上位机软件可以访问这些变量，如果 PLC 没有上位机软件，可以不设地址。当然，地址可在中间变量区自由选择。

第三步：插入一个用 ST 语言写的功能块 Input_Converter，在编辑区写入左下图中的代码，在功能块的变量工作单中建立右下图中的变量

```
IF EN THEN
    IN_REAL := WORD_TO_REAL(IN);
    IN_Low_REAL := WORD_TO_REAL(IN_Low);
    IN_Hi_REAL := WORD_TO_REAL(IN_Hi);
    TEMP1 := (IN_Real-IN_Low_REAL)*(OUT_Hi-OUT_Low);
    TEMP2 := IN_Hi_REAL-IN_Low_REAL;
    IF IN_REAL<IN_Low_REAL*0.995 THEN
        OUT := 0.0;
    ELSIF IN_REAL>IN_Hi_REAL*1.025 THEN
        TEMP3 := (IN_Hi_REAL*1.025-IN_Low_REAL)*(OUT_Hi-OUT_Low);
        OUT := TEMP3/TEMP2+OUT_Low;
    ELSE
        OUT := TEMP1/TEMP2+OUT_Low;
    END_IF;
    ENO := TRUE;
ELSE
    ENO := FALSE;
END_IF;
```

名称	类型	用法	描述
Default			
EN	BOOL	VAR_INPUT	
IN	WORD	VAR_INPUT	
IN_REAL	REAL	VAR	
IN_Hi	WORD	VAR_INPUT	
IN_Hi_REAL	REAL	VAR	
IN_Low	WORD	VAR_INPUT	
IN_Low_REAL	REAL	VAR	
TEMP1	REAL	VAR	
TEMP2	REAL	VAR	
TEMP3	REAL	VAR	
OUT_Hi	REAL	VAR_INPUT	
OUT_Low	REAL	VAR_INPUT	
ENO	BOOL	VAR_OUTPUT	
OUT	REAL	VAR_OUTPUT	

然后将生成的功能块拖拽到主程序的编辑区



上图的功能块 Input_Converter 完成了把 WORD 转换为实数送到变量 Wendu_1，引脚 IN 连接温度传感器来的 4~20mA 信号（地址%IW64），引脚 IN_Hi 连接常量 WORD#20000 表示第一个模拟量输入通道的量程上限为 20000，引脚 IN_Low 连接常量 WORD#4000 表示量程下限为 4000，引脚 OUT_Hi 连接常量 REAL#150.0 表示模拟量转换为工程量的上限为 150℃，引脚 OUT_Low 连接常量 REAL#0.0 表示下限为 0℃，引脚 OUT 连接变量 Wendu_1 表示转换为实数（工程量）后放在 Wendu_1 中（这个变量也连接到了 FPID 的 X 引脚）。

第四步：插入一个用 ST 语言写的功能块 Output_Converter，在编辑区写入左下图中的代码，在功能块的变量工作单中建立右下图中的变量

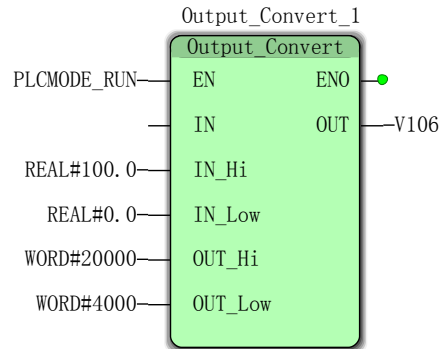
```
IF EN THEN
    OUT_Low_REAL := WORD_TO_REAL(OUT_Low);
    OUT_Hi_REAL := WORD_TO_REAL(OUT_Hi);

    TEMP1 := (IN-IN_Low)*(OUT_Hi_Real-OUT_Low_Real);
    TEMP2 := IN_Hi-IN_Low;

    IF IN<REAL#0.0 THEN
        OUT_TEMP := 0.0;
    ELSIF IN>REAL#100.0 THEN
        OUT_TEMP := 100.0;
    ELSE
        OUT_TEMP := TEMP1/TEMP2+OUT_Low_Real;
    END_IF;
    OUT := REAL_TO_WORD(OUT_TEMP);
    ENO := TRUE;
ELSE
    ENO := FALSE;
END_IF;
```

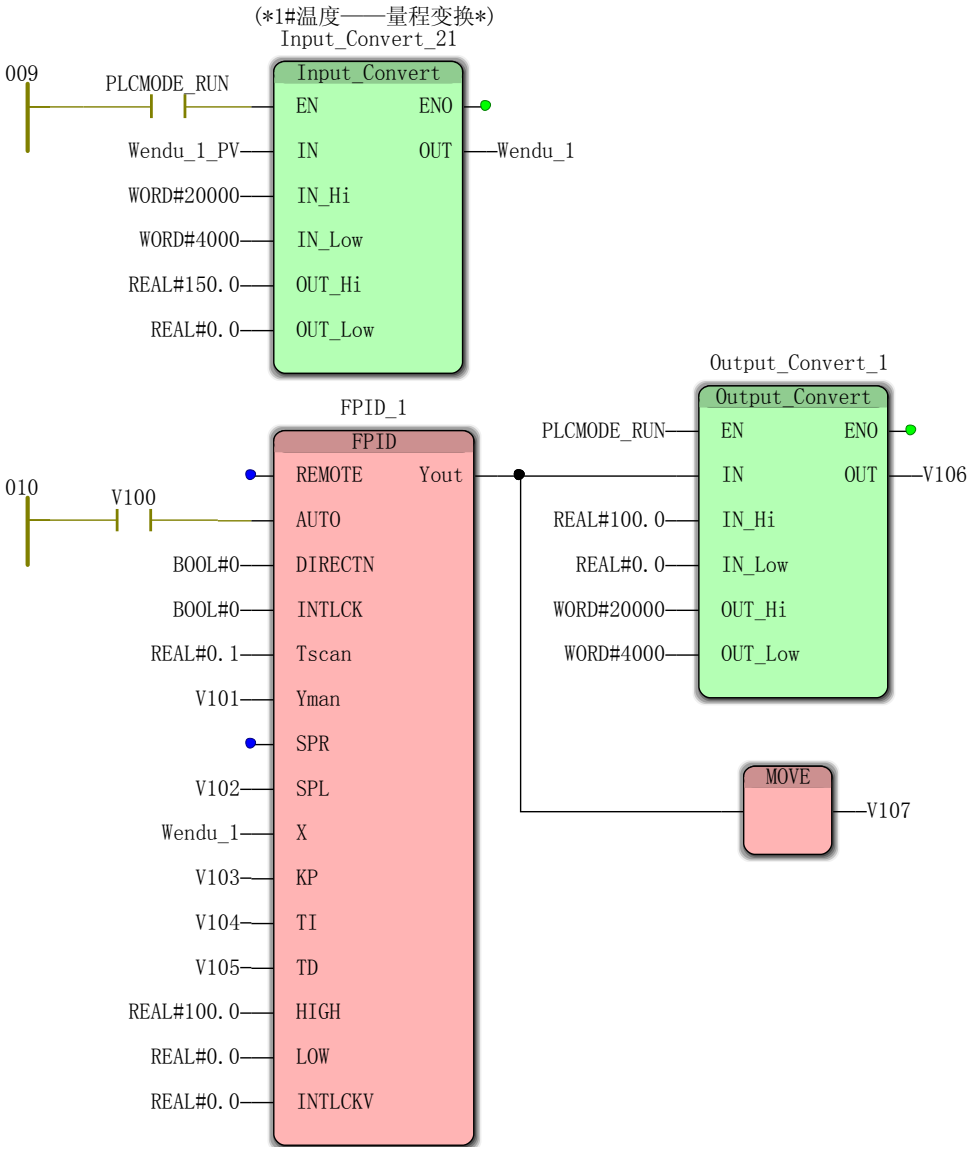
名称	类型	用法	描述
Default			
EN	BOOL	VAR_INPUT	
IN	REAL	VAR_INPUT	
IN_Hi	REAL	VAR_INPUT	
IN_Low	REAL	VAR_INPUT	
TEMP1	REAL	VAR	
TEMP2	REAL	VAR	
OUT_Hi	WORD	VAR_INPUT	
OUT_Low	WORD	VAR_INPUT	
OUT_Hi_REAL	REAL	VAR	
OUT_Low_REAL	REAL	VAR	
ENO	BOOL	VAR_OUTPUT	
OUT_TEMP	REAL	VAR	
OUT	WORD	VAR_OUTPUT	

然后将生成的功能块拖拽到主程序的编辑区



上图的功能块 Output_Converter 完成了把实数转换为 WORD 送到变量 V106，引脚 IN 连接 FPID 的输出 Yout，引脚 IN_Hi 连接常量 REAL#100.0 表示 FPID 的输出上限为 100，引脚 IN_Low 连接常量 REAL#0.0 表示 Yout 的下限为 0，引脚 OUT_Hi 连接常量 WORD#20000 表示转换为 WORD 类型数据的上限为 20000（送至模拟量输出通道），引脚 OUT_Low 连接常量 WORD#0 表示下限为 0，引脚 OUT 连接变量 V106 表示转换为 WORD 后放在 V106 中（送至第一个模拟量输出通道，地址%QW64）。

完整的 PID 程序如下



图中，变量 V107（地址%MD3.01022）为的是上位机软件可以访问 FPID 的输出值 Yout。

5.6.10 在错误目录中获得的错误的详细信息 GET_ERROR

注：GET_ERROR 指令暂时不能用。

5.6.11 在错误目录中获得的当前内容的信息 GET_ERROR_CATALOG

注：GET_ERROR_CATALOG 指令暂时不能用。

5.6.12 搜索 PDD 变量的符号名称 GET_SYM

注：GET_SYM 指令暂时不能用。

5.6.13 PLC 热启动 HOT_RESTART

注：HOT_RESTART 指令暂时不能用。

5.6.14 数据复制 IMEMCPY

功能

IMEMCPY 指令用于将数据从源数据区域复制到目标数据区域。

用法

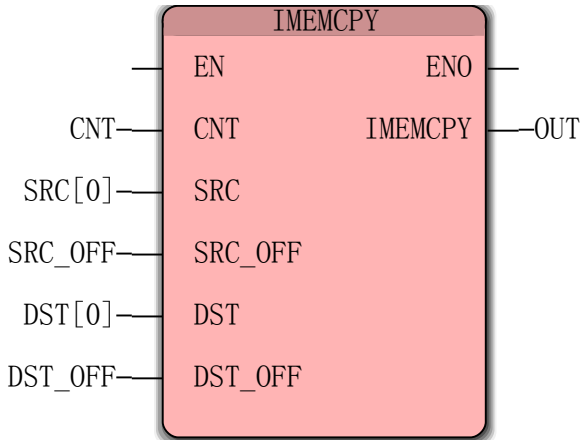
IL 编程语言

```
LD CNT
IMEMCPY SRC, SRC_OFF, DST, DST_OFF
ST OUT
```

ST 编程语言

```
OUT := IMEMCPY ( CNT, SRC, SRC_OFF, DST, DST_OFF ) ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 CNT、SRC 等或使用常量

IMEMCPY 指令处理的数据类型

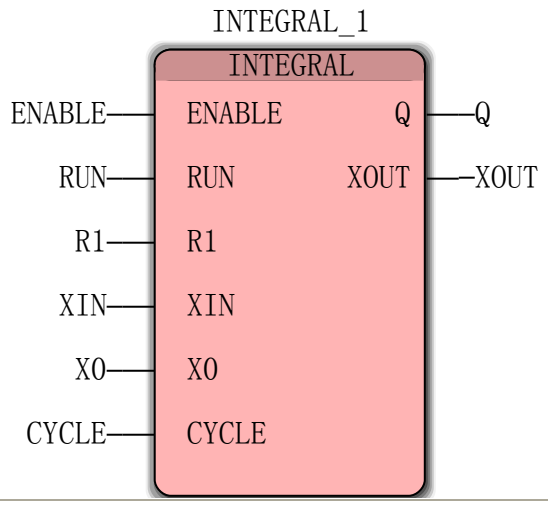
输入和输出	数据类型	描述
CNT（CNT）	INT	将要复制的字节数
SRC[0]（SRC）	BYTE	源数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]
SRC_OFF（SRC_OFF）	INT	源数据区域的起始字节序号，第一个字节的序号为 0
DST[0]（DST）	BYTE	目标数据区域的第一个字节，数据类型为 ARRAY 时写为 DST[0]
DST_OFF（DST_OFF）	INT	目标数据区域的起始字节序号号，第一个字节的序号为 0
OUT（IMEMCPY）	INT	错误代码： 0 复制了数据，无错误发生 14 缓冲区超过数据段 15 目标区域是一个输入组

5.6.15 积分 INTEGRAL

功能

INTEGRAL 指令用于对数据做时间的积分计算，使用积分指令时，需要把 POU 所在的任务类型设置为周期扫描（CYCLIC），并根据自己的需要设置扫描周期，任务类型的设置和扫描周期见本手册编程模型->硬件->Tasks。

用法

IL 编程语言	LD、 FBD 编程语言
见帮助文件 ST 编程语言 INTEGRAL_1 (ENABLE := ENABLE, RUN := RUN, R1 := R1, XIN := XIN, X0 := X0, CYCLE := CYCLE); Q := INTEGRAL_1.Q; XOUT := INTEGRAL_1.XOUT;	

注：IL、ST 语言编程时需在当前 POU 的变量工作单中插入变量 E、RUN 等或使用常量

INTEGRAL 指令处理的数据类型

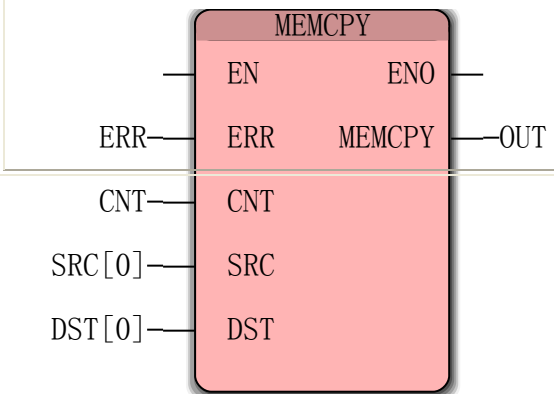
输入和输出	数据类型	描述
ENABLE	BOOL	TRUE 时，执行功能块
RUN	BOOL	TRUE 时，开始积分，FALSE 时，积分暂停，并且输出保持最后的积分值
R1	BOOL	TRUE 时复位，复位值是 X0
XIN	REAL	输入值
X0	REAL	复位值
CYCLE	TIME	时间常数，单位秒，本指令实际上是对 XIN×CYCLE 进行积分
Q	BOOL	Q 等于 R1 取反
XOUT	REAL	输出值，积分结果

5.6.16 数据复制 MEMCPY

功能

MEMCPY 指令用于将数据从源数据区域复制到目标数据区域。

用法

IL 编程语言	LD、 FBD 编程语言
LD ERR MEMCPY CNT, SRC[0], DST[0] ST OUT ST 编程语言	

OUT := MEMCPY (ERR, CNT, SRC[0], DST[0]) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 ERR、CNT、SRC[0]等或使用常量	

MEMCPY 指令处理的数据类型

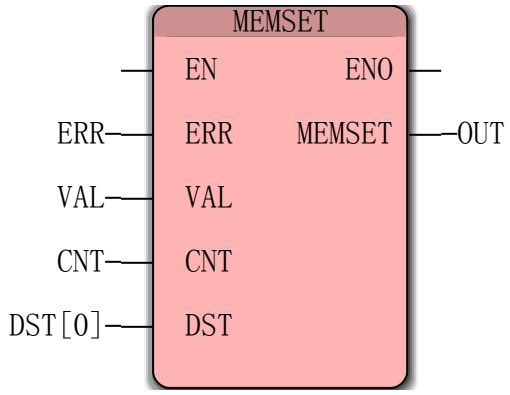
输入和输出	数据类型	描述
ERR（ERR）	INT	错误代码：0——正确复制；14——缓冲区超过数据段；15——目标区域是一个输入数组。注意：这是放置在左边的输出参数！
CNT（CNT）	INT	要复制的字节数
SRC[0]（SRC）	BYTE	源数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]
DST[0]（DST）	BYTE	目标数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]
OUT（IMEMCPY）	WORD	输出，暂没有定义 ，实际上字符是被拷贝到目标数据区 DST 了

5.6.17 数据分发 MEMSET

功能

MEMSET 指令用于将源数据分发到目标数据区域。

用法

IL 编程语言	LD、 FBD 编程语言
LD ERR MEMSET VAL, CNT, DST[0] ST OUT	
ST 编程语言	
OUT := MEMSET (ERR, VAL, CNT, DST[0]) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 ERR、VAL, CNT 等或使用常量	

MEMSET 指令处理的数据类型

输入和输出	数据类型	描述
ERR（ERR）	INT	错误代码：0——正确复制；14——缓冲区超过数据段；15——目标区域是一个输入数组。注意：这是放置在左边的输出参数！
VAL（VAL）	BYTE	源数据——要分发的数据
CNT（CNT）	DINT	要分发的份数，可把一个源数据分发到目标数据区中的 N 个字节中，一个字节一份
DST[0]（DST）	BYTE	目标数据区域的第一个字节，数据类型为 ARRAY 时写为 SRC[0]
OUT（IMEMCPY）	WORD	输出，暂没有定义 ，实际上字符是被拷贝到目标数据区 DST 了

5.6.18 PID

见 FPID。

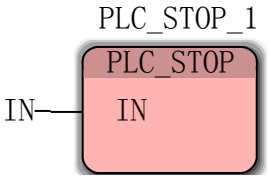
5.6.19 PLC 停止 PLC_STOP

功能

PLC_STOP 指令用于停止 PLC。

在冷再启动过程中，初始化所有数据。如程序发生堆栈溢出、字符串错误或者被 0 除等问题时，可以调用这个指令自动重新启动程序的执行。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN ST PLC_STOP_1.IN CAL PLC_STOP_1	
ST 编程语言	
PLC_STOP_1 (IN);	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN 或使用常量	

PLC_STOP 指令处理的数据类型

输入和输出	数据类型	描述
IN	BOOL	上升沿有效，PLC 停止

5.6.20 读 PDD 变量的值 RD_*_BY_SYM

RD_*_BY_SYM 包括以下指令，分别用于读取 PDD 不同数据类型的变量

RD_BOOL_BY_SYM	RD_BYTE_BY_SYM	RD_WORD_BY_SYM	RD_DWORD_BY_SYM
RD_SINT_BY_SYM	RD_INT_BY_SYM	RD_DINT_BY_SYM	RD_USINT_BY_SYM
RD_UINT_BY_SYM	RD_UDINT_BY_SYM	RD_REAL_BY_SYM	RD_STRING_BY_SYM
RD_TIME_BY_SYM	RD_INPUT_GROUP		

PDD 是通过变量的名称可以访问到该变量值的办法，是在控制器内核层访问 MULTIPROG 定义的 PLC 变量的值时所使用的一种方法，RD_*_BY_SYM 指令暂时不能使用，一般用户直接读写变量就可以了。

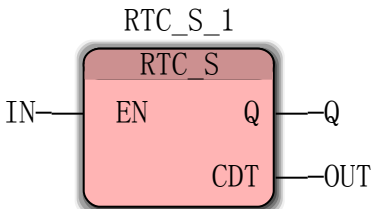
5.6.21 读取 PLC 时钟 RTC_S

功能

RTC_S 指令用于读取 PLC 的时钟放在一个字符串变量中，读取的时钟为 GMT 格式。

IEC 61131-3 规定的日期和时间输出字符串格式为：DT#1998-11-21-15:27:56.46。

用法

IL 编程语言	LD、 FBD 编程语言
LD IN ST RTC_S_1.EN CAL RTC_S_1 LD RTC_S_1.Q ST Q LD RTC_S_1.CDT ST OUT	
ST 编程语言	
RTC_S_1 (EN := (IN)); Q := RTC_S_1.Q; OUT := RTC_S_1.CDT;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 IN、Q、OUT 或使用常量	

RTC_S 指令处理的数据类型

输入和输出	数据类型	描述
IN（EN）	BOOL	如果为 TRUE，则实际的日期和时间被写入所连结的输出字符串中
Q（Q）	BOOL	如果 EN 为 TRUE，Q 为 TRUE，否则 Q 为 FALSE
OUT（CTD）	STRING	取得的日期与时间，如 DT#2011-08-15-10:08:55.19

5.6.22 PLC 暖启动 WARM_RESTART

注：WARM_RESTART 指令暂时不能用。

5.6.23 写 PDD 变量的值 WR_*_BY_SYM

WR_*_BY_SYM 包括以下指令，分别用于写 PDD 不同数据类型的变量

WR_BOOL_BY_SYM	WR_BYTE_BY_SYM	WR_WORD_BY_SYM	WR_DWORD_BY_SYM
WR_SINT_BY_SYM	WR_INT_BY_SYM	WR_DINT_BY_SYM	WR_USINT_BY_SYM
WR_UINT_BY_SYM	WR_UDINT_BY_SYM	WR_REAL_BY_SYM	WR_STRING_BY_SYM
WR_TIME_BY_SYM	WR_OUTPUT_GROUP		

PDD 是通过变量的名称可以访问到该变量值的办法，是在控制器内核层访问 MULTIPROG 定义的 PLC 变量的值时所使用的一种方法，WR_*_BY_SYM 指令暂时不能使用，一般用户直接读写变量就可以了。

5.6.24 PLC 暖启动 WRITE_RETAIN

注：WRITE_RETAIN 指令暂时不能用。

6. 编程语言

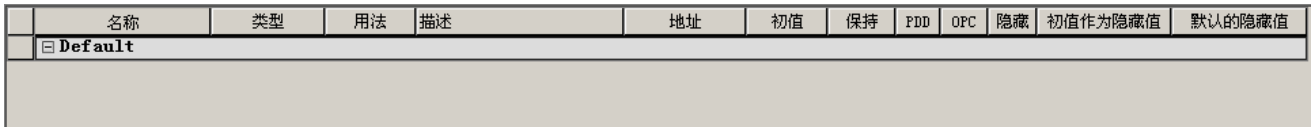
MULTIPROG 支持 IL、ST、FBD、LD、SFC 五种编程语言，其中 IL 和 ST 属于文本类编程语言，FBD、LD 和 SFC 属于图形类编程语言。一个具有独立功能的程序分为代码部分和数据部分，代码要使用 IL、ST、FBD、LD、SFC 中的一种或几种语言的组合来编写，数据要在变量工作单中声明。本章介绍如何声明变量以及如何使用这五种编程语言编程。

IL 是 Instruction List 指令表的缩写，ST 是 Structured Text 结构化文本的缩写，FBD 是 Function Block Diagram 功能图表的缩写，LD 是 Ladder Diagram 的缩写，SFC 是 Sequential Function Chart 是顺序功能图的缩写。图形类编程语言中程序是从上到下、从左到右扫描的，文本类编程语言中程序是从上到下扫描的。

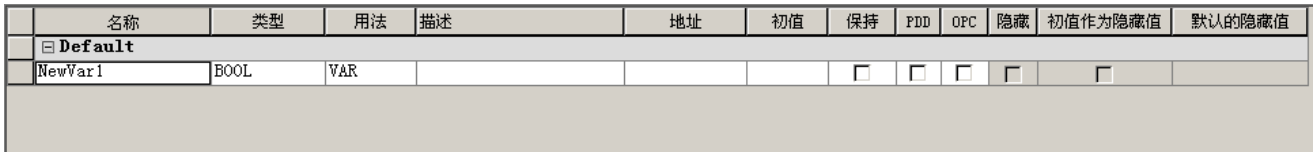
本章分成以下五节：

6.1 变量工作单

IL、ST、FBD、LD、SFC 五种编程语言需要在各程序组织单元 POU 所对应的变量工作单中声明变量，用户选中工程树中的 POU 目录下的程序、功能或功能块，点击菜单栏上的变量工作单，进入变量工作单，如下



右击第一行，选择追加变量，MULTIPROG 自动插入一个缺省变量，如下



其中

- 名称栏是变量的名称，默认 NewVar1，用户可修改变量名，变量名必须以字母开头，可包含字母、数字和下划线；
- 类型栏是数据类型，用户可直接键入用户名，或通过下拉菜单选择；
- 用法栏，表示插入的变量的作用域：
  对于插入的程序，只有 VAR 和 VAR_EXTERNAL 两个选项，VAR 表示内部变量，VAR_EXTERNAL 表示外部变量；
  对于插入的功能，只有 VAR 和 VAR_INPUT 两个选项，VAR 表示内部变量，VAR_INPUT 表示输入变量；
  对于插入的功能块，有 VAR、VAR_EXTERNAL、VAR_IN_OUT、VAR_INPUT 和 VAR_OUTPUT，VAR 表示内部变量，VAR_EXTERNAL 表示外部变量，VAR_IN_OUT 表示输入输出变量，VAR_INPUT 表示输入变量，VAR_OUTPUT 表示输出变量。
- 描述栏是用户输入的文字描述；地址栏是该变量的输入、输出、中间变量；
- 地址栏，标明该变量的地址；
- 初值栏，在 PLC 程序中，第一次使用这个变量时，会使用在这里标明的初始值；
- 保持栏，在 PLC 断电的情况下，该变量的值仍被保存，暖启动之后，将使用变量的最后值；
- PDD 栏，指明变量已写入过程数据目录（PDD）中，仅当用户访问 PLC 上的某个地址对应的变量名称时，才勾选上它。
- OPC 栏，指明变量已写入 OPC 服务器文件中，仅当用户希望通过 OPC 客户端访问该变量时才勾选上它。

FBD、LD 和 SFC 编程语言编程时，也可在编辑区插入变量，插入后的变量自动列入变量工作单中。

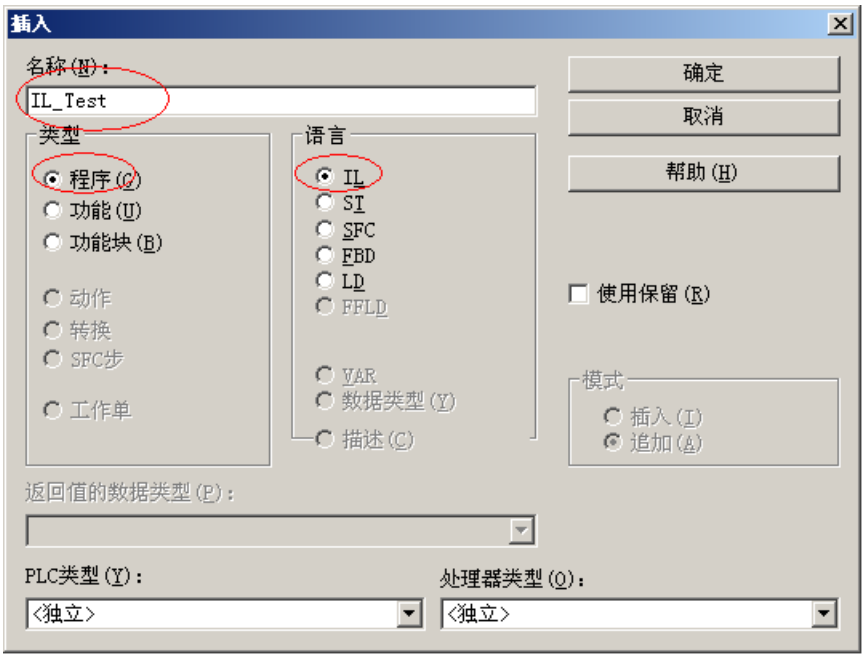
6.2IL 指令表编程语言

指令表编程语言的基本语句是指令表，它是一种底层语言，采用面向机器的操作符，比较容易转换为可编程控制器的机器代码，因为缺乏有效的工具，指令表编程语言比较适合小型的控制程序，不适合大型复杂的控制任务。

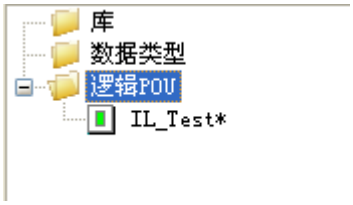
6.2.1 创建一个 IL 程序

使用 IL 编程，用户可在编辑区键入代码，对于指令，可直接键入或在“编辑向导”中拖拽指令到编辑区。下面以写一个求绝对值的程序为例说明 IL 的编程过程：

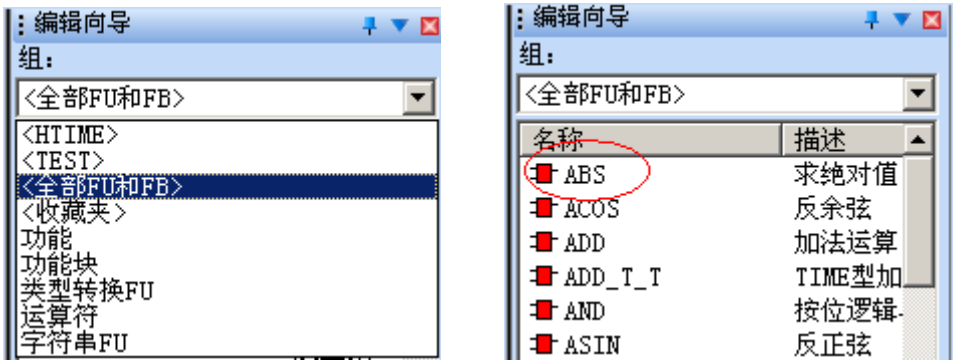
1. 创建一个工程
2. 右击工程树中的“逻辑 POU”，选择插入->程序，弹出对话框，如下图，在对话框中键入程序名，本例中为 IL_Test，在语言栏选择 IL，点击确定。



在工程树中出现了一个程序 POU——IL_Test，如下图。



3. 双击逻辑 POU->IL_Test，然后在编辑向导区选择全部 FU 和 FB，出现全部功能和功能块，如下两图。



4. 双击 ABS，或拖拽 ABS 到编辑区，ABS 功能就出现在编辑区了，如下图。

```
1    LD   (* IN1 作为 ANY_NUM *)
2    ABS
3    ST   (* 结果 作为 ANY_NUM *)
4
```

将图中的绿色字体部分分别改为 IN 和 OUT（用户可自己选择变量名，不一定是 IN 和 OUT），如下图。

```
1    LD   IN
2    ABS
3    ST   OUT
4
```

5. 点击菜单栏上的变量工作单，声明两个变量 IN 和 OUT，类型为 REAL，用法为 VAR，如下图。

	名称	类型	用法	描述	地址	初值
	Default					
	IN	REAL	VAR			
	OUT	REAL	VAR			

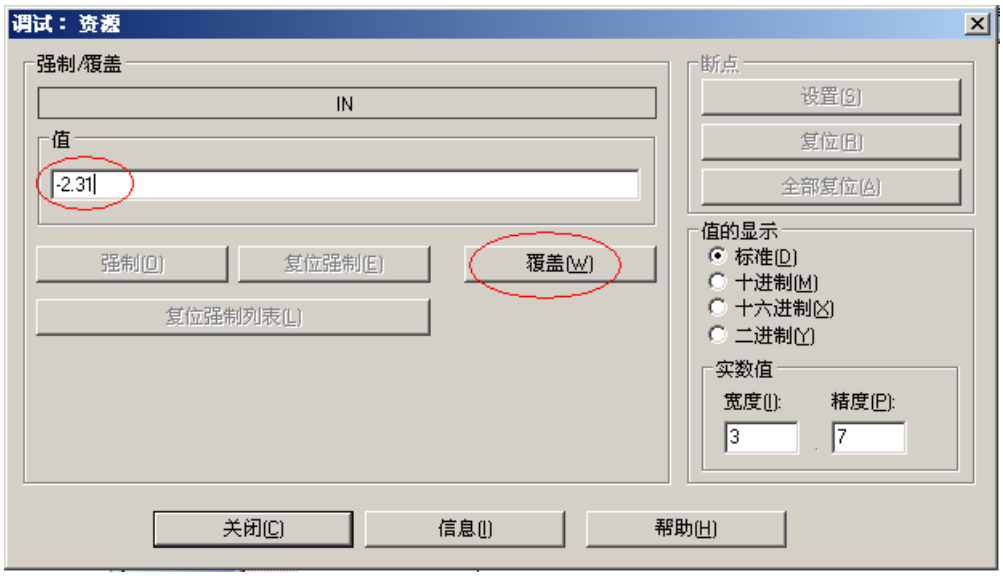
6. 点击制作，并下载，然后点击调试开关，如下图。

```
1    0.0000000E+000   LD   IN
2                     ABS
3    0.0000000E+000   ST   OUT
4
```

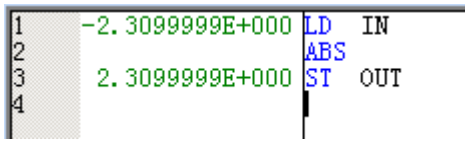
7. 用鼠标分别选中两个操作数，右击，选择添加到监视窗口，监视窗口如下图。

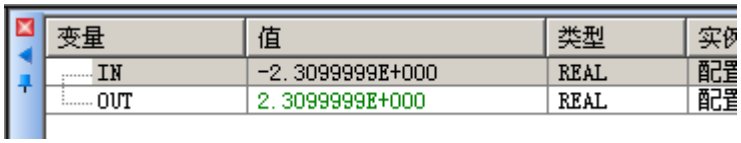
变量	值	类型	实例
IN	0.0000000E+000	REAL	配置.资源.任
OUT	0.0000000E+000	REAL	配置.资源.任

8. 双击监视窗口中的 IN 变量，写入-2.31，然后点击覆盖，如下图。



9. 在编辑区和监视窗口均可显示变量的状态值，如下两图。

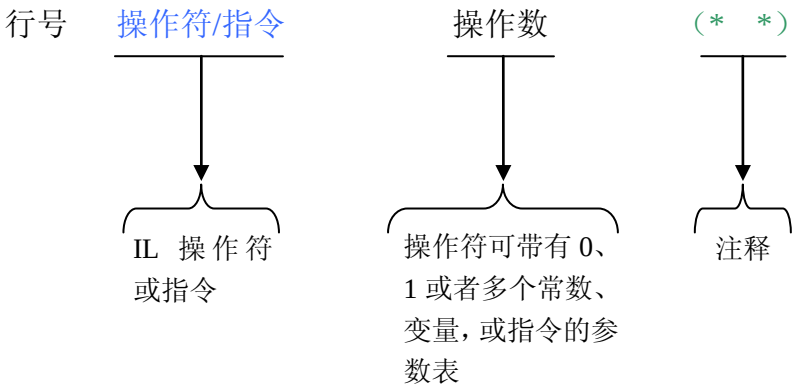




至此，一个完整的简单的 IL 程序完成了，因为实数（REAL）类型精度较低，上图中显示值不是 2.31，而是 2.3099999。

6.2.2 IL 的语句

在指令表语言 IL 的编辑区，可逐行写入 IL 编程语言的语句，每条语句都占据一行，每行语句的前面（非编辑区）是一个行号，每行指令后不带分号。IL 的语句格式如下



IL 语句中以不同颜色来区分各部分：操作符/功能为蓝色，操作数为黑色，注释为绿色。

指令举例：以 ABS 指令为例，解释一下求绝对值的代码，如下

```
LD      IN      (* IN1 作为 ANY_NUM *)
ABS
ST      OUT     (* 结果 作为 ANY_NUM *)
```

其中，LD 是个操作符，LD 后面的变量 IN 是一个操作数，LD 的作用是把其后的操作数 IN 装载到累加器；ABS 是求绝对值指令，求的绝对值后再将结果装载到累加器；ST 后面的变量 OUT 也是一个操作数，作用是把累加器中的数据赋给其后的操作数；绿色字体部分是注释，提

示用户声明一个变量作为操作数，数据类型是 ANY_NUM。

6.2.3 IL 的操作符

IL 编程语言除了指令集中的指令可用以外，IEC 61131-3 标准将以下 24 种指令定为标准指令。

操作符	修饰符	操作数	说明
LD	N	ANY	将其后的操作数装载到累加器
ST	N	ANY	将累加器中的数据赋给其后的操作数
S		BOOL	将操作数
R		BOOL	
AND	N, (	ANY_BIT	逻辑与
&	N, (	ANY_BIT	逻辑与
OR	N, (	ANY_BIT	逻辑或
XOR	N, (	ANY_BIT	逻辑异或
NOT	(	ANY_BIT	逻辑取反
ADD	(	ANY_NUM	加
SUB	(	ANY_NUM	减
MUL	(	ANY_NUM	乘
DIV	(	ANY_NUM	除
MOD	(	ANY_INT	取模
GT	(	ANY_NUM, ANY_BIT	比较，大于，>
GE	(	ANY_NUM, ANY_BIT	比较，大于等于，>=
EQ	(	ANY_NUM, ANY_BIT	比较，等于，=
NE	(	ANY_NUM, ANY_BIT	比较，不等于，<>
LE	(	ANY_NUM, ANY_BIT	比较，小于等于<=
LT	(	ANY_NUM, ANY_BIT	比较，小于<
JMP	C, N	LABAL	跳转到标号处的指令
CAL	C, N	NAME	调用功能块
RET	C, N		从被调用的函数、功能块处返回
)			延迟结束

注：

修饰符 N 表示取反操作，如 ANDN 表示将操作数取反后与；

修饰符 C 表示只有在当前运算结果为真时才执行，如：

```
LD IN1
AND IN2
ST OUT1
JMPC M2

…

M2:
LD IN3
ST OUT2
```

6.3 ST 结构化文本编程语言

结构化文本编程语言 ST 是高层语言，类似于 Pascal 编程语言，它不采用低层的面向机器的操作符，而是使用类似于日常语言的语句来描述控制命令，能够描述复杂的算法。结构化文本语言的程序由语句组成，语句由表达式和关键字组成，具有以下特点：

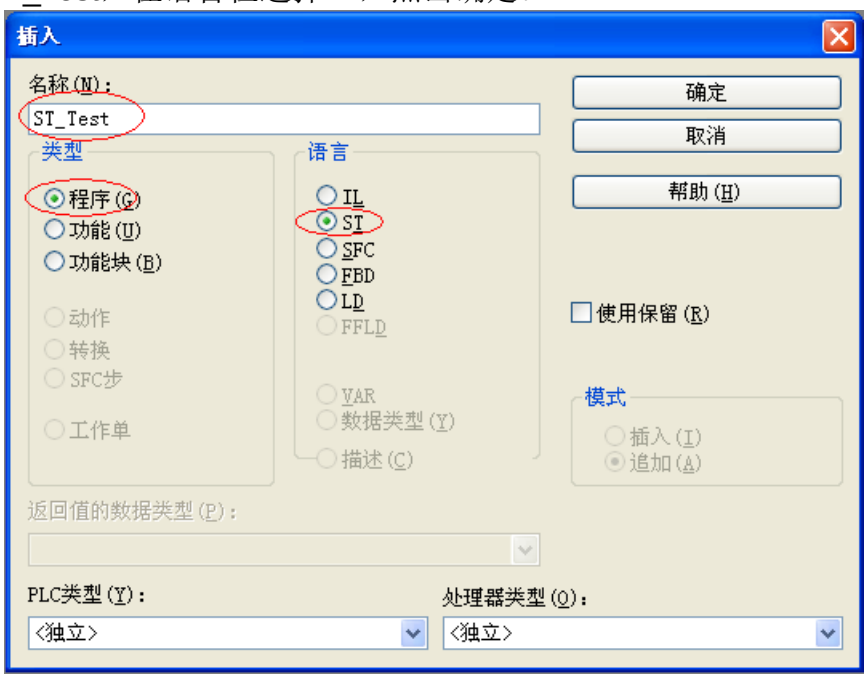
- 没有跳转语句，用条件语句实现程序的分支；
- 每条语句用分号“；”结束；
- 用(*　*) 为程序添加注释，注释不能嵌套，如(* (* *) *).

需要在 POU 对应的变量工作单中申明输入、输出、内部、外部、全局变量。

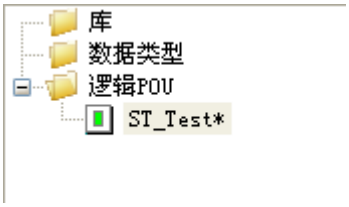
6.3.1 创建一个 ST 程序

使用 ST 语言编程，用户可在编辑区键入一条语句，也可直接在“编辑向导”中拖拽或双击指令到编辑区，再键入操作数和结束符“；”。下面以写一个求绝对值的程序为例说明 TS 的编程过程：

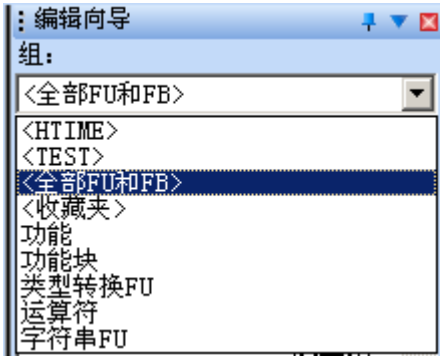
1. 创建一个工程
2. 右击工程树中的“逻辑 POU”，选择插入->程序，弹出对话框，在对话框中键入程序名，本例中为 IL_Test，在语言栏选择 IL，点击确定。

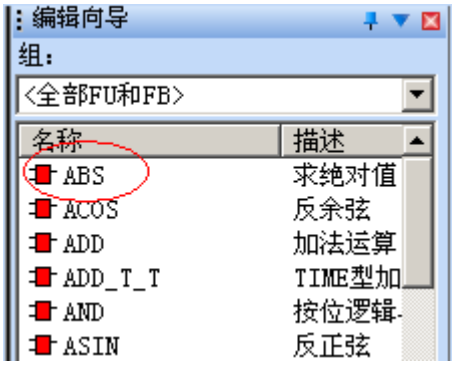


在工程树中出现了一个程序 POU，如下图。

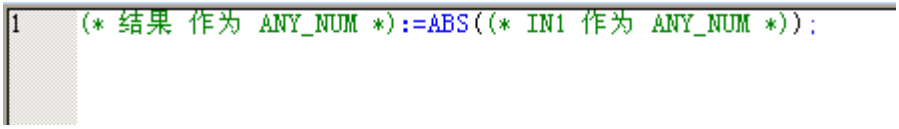


3. 双击逻辑 POU->ST_Test，然后在编辑向导区选择全部 FU 和 FB，出现全部功能和功能块，如下两图。

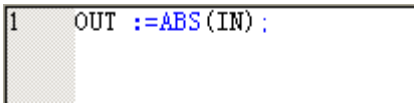




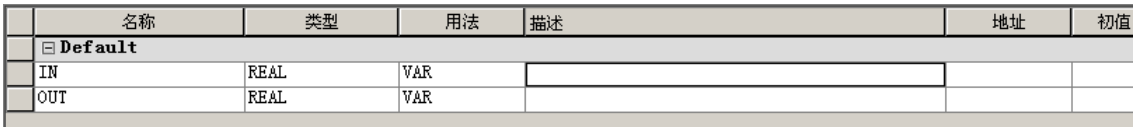
4. 双击 ABS，或拖拽 ABS 到编辑区，ABS 功能就出现在编辑区了，如下图



将图中的绿色字体部分分别改为 IN 和 OUT（用户可自己选择变量名，不一定是 IN 和 OUT），如下图。

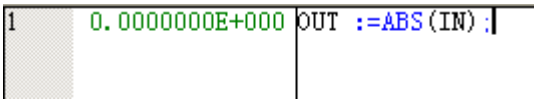


5. 点击菜单栏上的变量工作单，声明两个变量 IN 和 OUT，类型为 REAL，用法为 VAR，如下图。

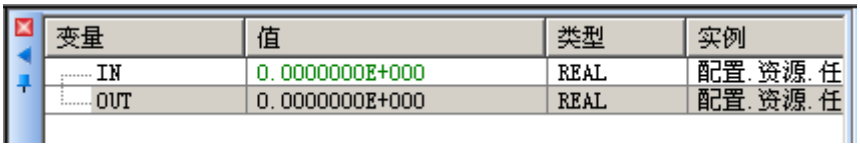


	名称	类型	用法	描述	地址	初值
	Default					
	IN	REAL	VAR			
	OUT	REAL	VAR			

6. 点击制作，并下载，然后点击调试开关，如下图。

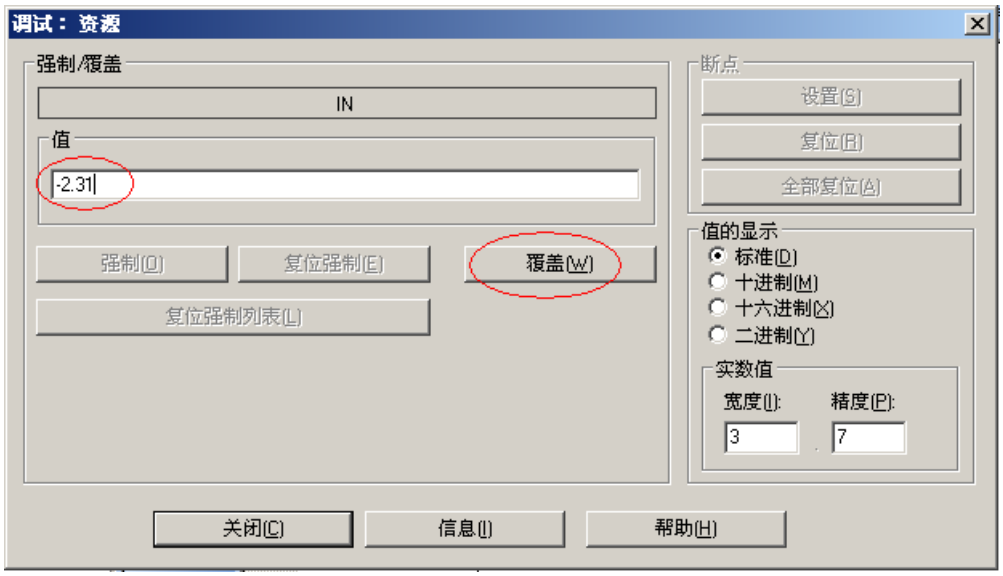


7. 用鼠标分别选中两个操作数，右击，选择添加到监视窗口，监视窗口如下图。

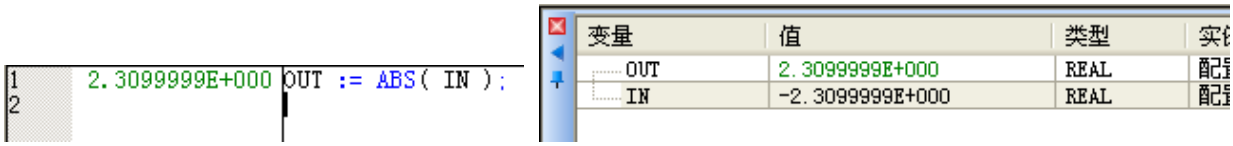


变量	值	类型	实例
IN	0.0000000E+000	REAL	配置.资源.任
OUT	0.0000000E+000	REAL	配置.资源.任

8. 双击监视窗口中的 IN 变量，写入-2.31，然后点击覆盖，如下图。



9. 在编辑区和监视窗口均可显示变量的状态值，如下两图。



至此，一个完整的简单的 ST 程序完成了。

6.3.2 ST 的语句

在 ST 编程语言中，程序由语句组成，语句由表达式和关键字组成。

在结构化文本编程语言 ST 的编辑区，可逐行写入 ST 编程语言的语句，每条语句都以分号结束，多条语句可占据一行，每行语句的前面（非编辑区）是一个行号。

如一条赋值语句，它由变量、赋值关键字和表达式组成，作用是将表达式的运算结果赋给赋值关键字左侧的变量：

变量名 := 表达式;

赋值关键字两侧的数据类型必须一致。

6.3.3 ST 的表达式

表达式由操作数和运算符组成，操作数可以是直接量、变量或功能名称。

可以使用的运算符如下表：

操作符	举例	例子的值	描述	优先级
( )	(2+3)* (4+5)	45	括号	高
**	3.0**4	81.0	乘幂运算	
-	-10	-10	求相反数	
NOT	NOT TRUE	false	按位取反	
*	10*3	30	乘法运算	
/	6/2	3	除法运算	

MOD	17 **MOD** 10	7	模数运算	
+	2**+**3	5	加法运算	
-	4**-**2	2	减法运算	
<, >, <=, >=	4 **>** 12	false	比较	
=	T#26h **=** T#1d2h	true	相等	
<>	8 **<>**16	true	不相等	
&, AND	TRUE **&** FALSE	false	布尔与	
OR	TRUE **OR** FALSE	true	布尔或	
XOR	TRUE **XOR** FALSE	true	布尔异或	低

ST 语言常用关键字

说明	**关键字**	**举例**	**描述**
赋值操作符	:=	OUT := IN	将 IN 赋给 OUT
返回	RETURN	RETURN;	退出被调用的功能、功能块或程序， 返回到调用它的语句处。
选择	IF	IF a < b THEN c:=1; ELSIF a=b THEN c:=2; ELSE c:=3; END_IF;	当 IF 后的表达式‘a<b’为 TRUE 时，执行 THEN 后的一条语句（以分号为界），否则不执行 THEN 后的语句，而继续判断 ELSIF 或 ELSE。
	CASE	CASE F OF 1: a:=3; 2: a:=4; 3: a:=2; ELSE a:=0; END_CASE;	根据 CASE 关键字后面的表达式的值，执行一组语句。变量或表达式‘F’必须为 INT 数据类型。

循环	FOR	FOR a:=1 TO 10 BY 3 DO
		f[a] :=b;
		END_FOR;
			变量‘a’从 1 开始，FOR 和 END_FOR 之间的语句被反复执行，没执行一次，a 增加 3，到 10 结束。所有值必须具有 ANY_INT 数据类型。

	WHILE	WHILE b > 1 DO
		b:= b/2;
		END_WHILE;
			当表达式‘b>1’的值为 TRUE 时，WHILE 和 END_WHILE 之间的语句被反复执行，直到‘b>1’的值为 FALSE。

	REPEAT	REPEAT
		a := a*b;
		UNTIL a <10000
		END_REPEAT;
			REPEAT 和 END_REPEAT 之间的语句被反复执行，直到所表达式‘a<10000’的值为 TRUE。

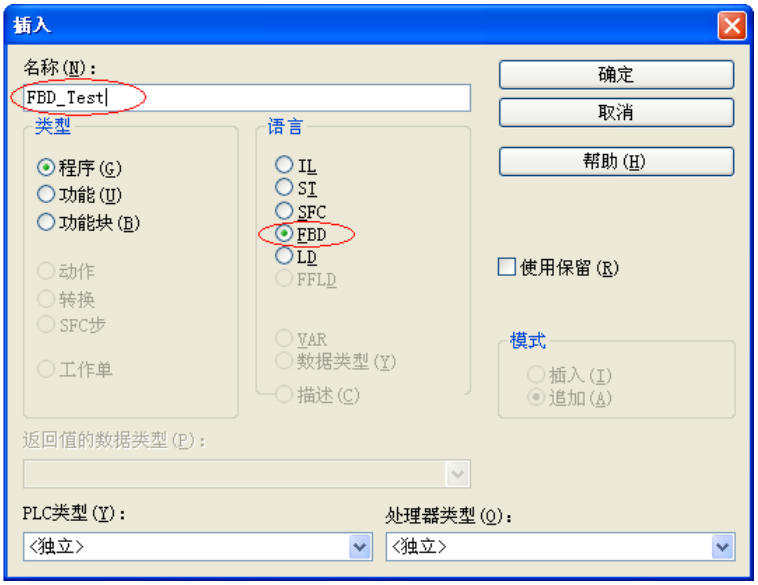
循环结束	EXIT	FOR a:=1 TO 2 DO IF flag THEN EXIT; END_IF SUM:= SUM + a END_FOR	退出语句可被用于中止循环语句的执行。
语句结束	;		放在语句后面，表示语句的结束，也可单独存在。

6.4 FBD 功能块图编程语言

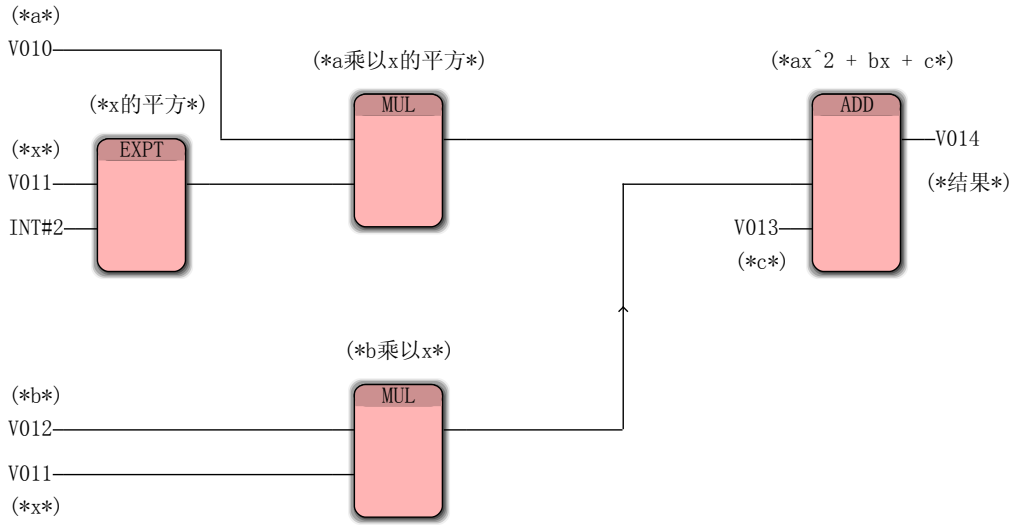
功能块图编程语言源于信号处理领域，它是 IEC 61499 标准的基础，一个功能块图编程语言的程序是将各种功能块连接起来，编程语言的元素是函数、功能块和连接符号。

创建一个 FBD 程序

创建一个工程后，右击工程树中的“逻辑 POU”，选择插入->程序，在弹出的对话框中键入程序名 FBD_Test、选择类型和编程语言，如下图

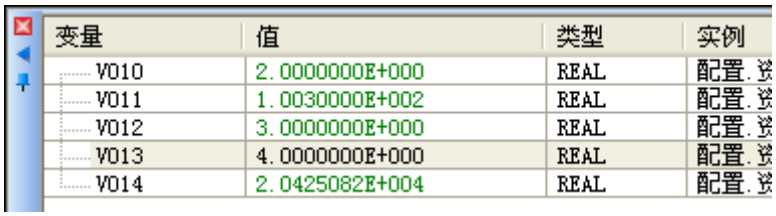


双击工程树中创建的 FBD_Test，在编辑向导中找到 ADD、MUL 和 EXPT 功能块，将他们拖拽到 FBD_TesT 的编辑区，我们要用它们完成一个一元二次函数的计算功能，即 $y=ax^2+bx+c$，建立的 FBD 程序如下图。



其中功能块 EXPT 完成 x 的平方运算，功能块 MUL 完成 a 与 x^2 的乘法，另一个 MUL 完成 b 与 x 的乘法，功能块 ADD 完成 ax^2 与 bx 与 c 的加法，其中，变量 V011 为 x，变量 V012 为 b，变量 V010 为 a，V013 为 c，变量 V014 为最后结果。变量 V010~V014 为实数类型。

点击制作，然后下载，再点击调试开关，将 V010~V014 添加到监视窗口，将 V010 改为 2.0，V011 改为 100.3，V012 改为 3.0，V013 改为 4.0，则程序自动计算出 ax^2+bx+c 的结果，放在变量 V014 中，如下图



变量	值	类型	实例
V010	2.0000000E+000	REAL	配置.
V011	1.0030000E+002	REAL	配置.
V012	3.0000000E+000	REAL	配置.
V013	4.0000000E+000	REAL	配置.
V014	2.0425082E+004	REAL	配置.

在 FBD 中，是不需要额外在变量工作单中建立变量的，所以，在插入变量并双击变量后，这些变量就出现在变量工作单中了。

FBD 功能块图编程语言与 LD 梯形图编程语言相似。

至此，一个完成的 FBD 完成了。

6.5 LD 梯形图编程语言

梯形图编程语言是历史最久远的一种编程语言，梯形图源于电气系统的逻辑控制图，逻辑图采用继电器、触点、线圈和逻辑关系图等表示它们之间逻辑关系，梯形图编程语言可采用的图形元素有梯形图网络、电源轨线、连接导线、触点、线圈和功能块等，触点和线圈的数据类型可以是 BOOL、BYTE、WORD 和 DWORD。

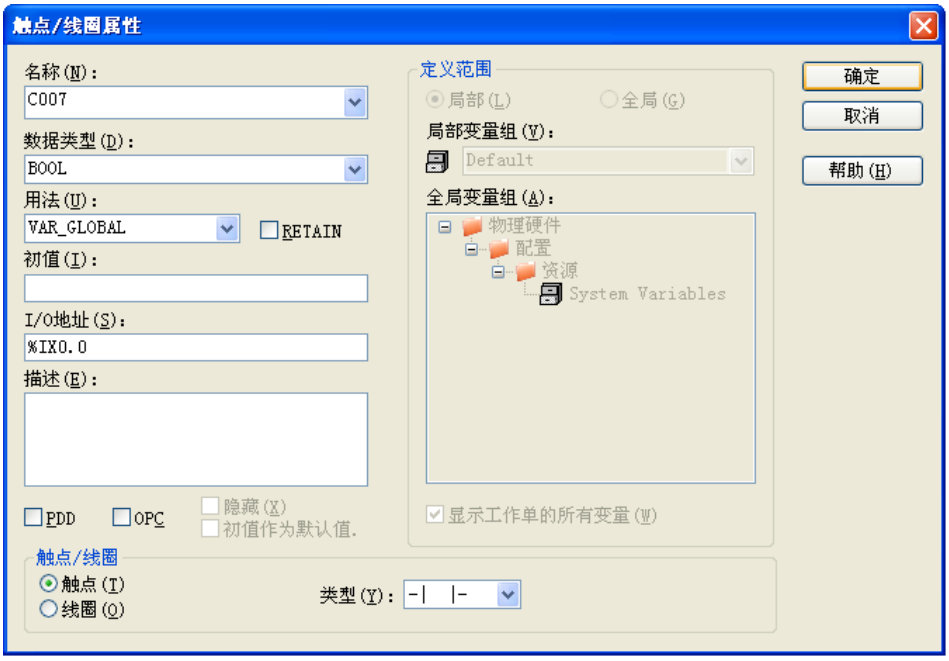
6.5.1 创建一个 LD 程序

创建一个工程后，右击工程树中的“逻辑 POU”，选择插入->程序，在弹出的对话框中键入程序名 LD_Test、选择类型和编程语言，如下图

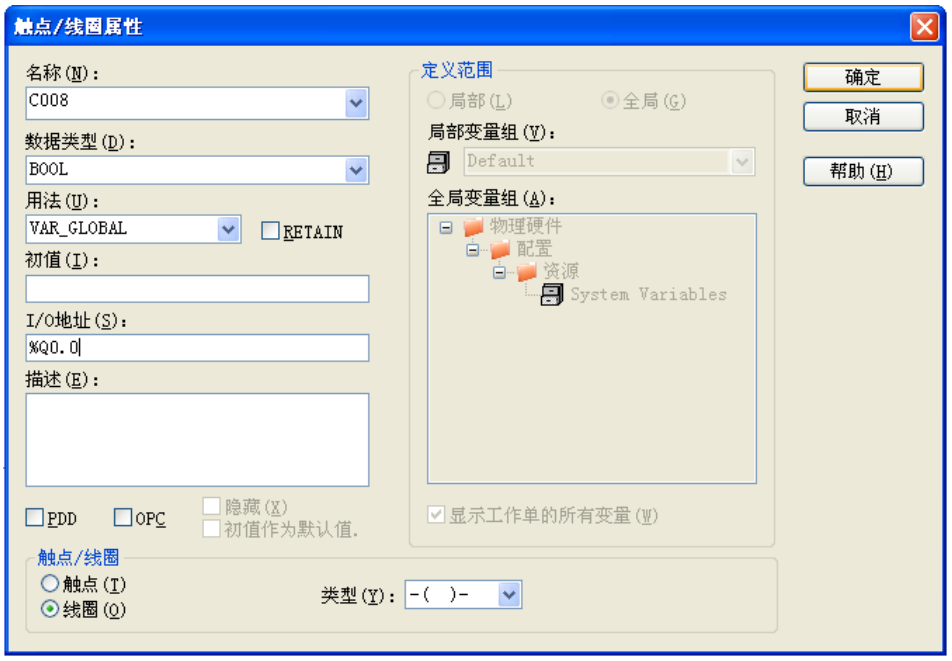
点击 MULTIPROG 编程软件的编辑区域，然后点击编辑区域左侧的

图标，在编辑区域中出现了一个梯形图的简单网络，左侧是一个常开触点，变量名是 C007，右侧是一个线圈，变量名是 C008，如下图。

双击常开触点 C007，出现触点/线圈属性对话框，在 I/O 地址(S)栏，输入%IX0.0，表示 PLC 本机的第一个数字量输入通道，点击“确定”，如下图。



双击梯形图中线圈 C008，出现触点/线圈属性对话框，在 I/O 地址(S)栏，输入%QX0.0，表示 PLC 本机的第一个数字量输出通道，点击“确定”，如下图。



在 LD 中，是不需要额外在变量工作单中建立变量的，所以，在插入一个梯形图网络，并双击触点和线圈后，这两个变量就出现在变量工作单中了，也可以在编辑区右击鼠标，选择“变量(V)”插入一个变量，这个插入的变量必须要连接到功能块引脚端。

在上图中，C007 和 C008 的变量工作范围（用法）设为 VAR_GLOBAL，说明这两个变量是全局变量，可在本工程中的其他程序里使用。

至此，一个完整的 LD 程序就完成了，当连接到第一个数字量输入通道 I0.0 的触点闭合时，

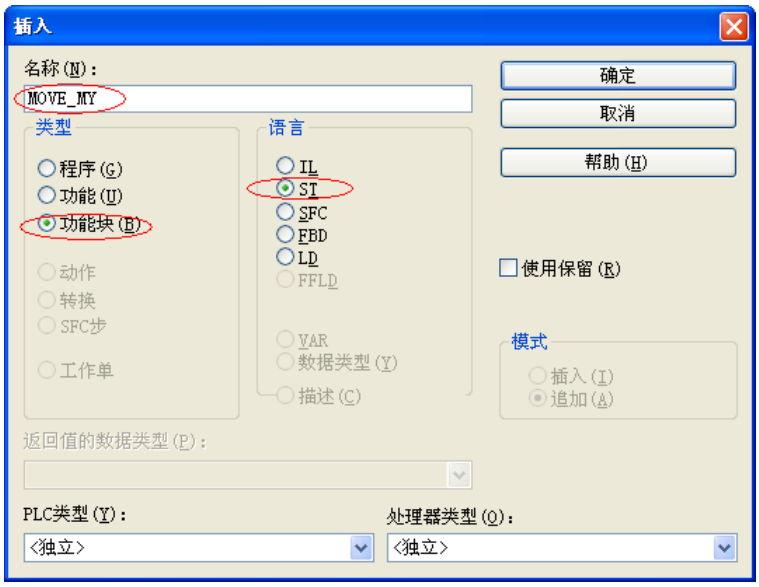
与第一个数字量输出通道 Q0.0 连接的线圈导通。

6.5.2 在 LD 中插入一个 FB

使用 LD 编程时，有时会需要一个特殊的功能块，而编辑向导中却没有，此时用户可以在 LD 中创建自己的功能块，下面说明如何在 LD 中创建一个 MOVE 功能块。

在工程树中右击用创建的 LD 程序名，选择插入->功能块，如下图

弹出一个对话框，如下图，在这个对话框中键入要创建的 FB 的名称、制定类型和所用语言，这里分别是 MOVE_MY、功能块和 ST，点击确定后，就进入 ST 语言编程了。



我们要创建的 MOVE 功能块，有输入引脚 EN、IN、Mode1、Mode2，有输出引脚 ENO 和 MODE，这个功能块要在前面所创建的 LD 程序中使用。

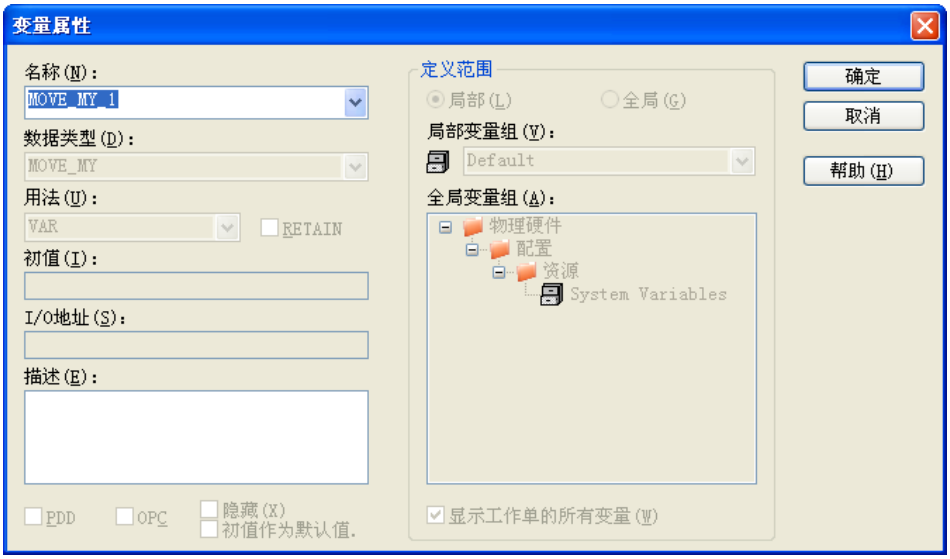
与使用 ST 创建一个程序时一样，首先要在变量工作单中（注意这里是 MOVE_MY 的变量工作单，要双击工程树中的 MOVE_MY 功能块，再点击工具栏中的变量工作单）建立 MOVE_MY 的变量，如下图

名称	类型	用法	描述
Default			
EN	BOOL	VAR_INPUT	
IN	BOOL	VAR_INPUT	
Mode1	WORD	VAR_INPUT	
Mode2	WORD	VAR_INPUT	
ENO	BOOL	VAR_OUTPUT	
MODE	WORD	VAR_OUTPUT	

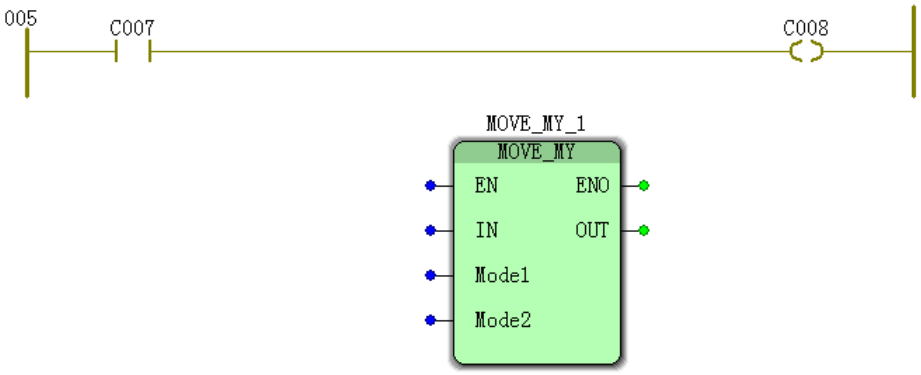
然后回到 MOVE_MY 的编辑区，键入以下代码

```
IF EN THEN
    IF IN THEN
        MODE := Mode1;
    ELSE
        MODE := Mode2;
    END_IF;
    ENO := TRUE;
ELSE
    ENO := FALSE;
END_IF;
```

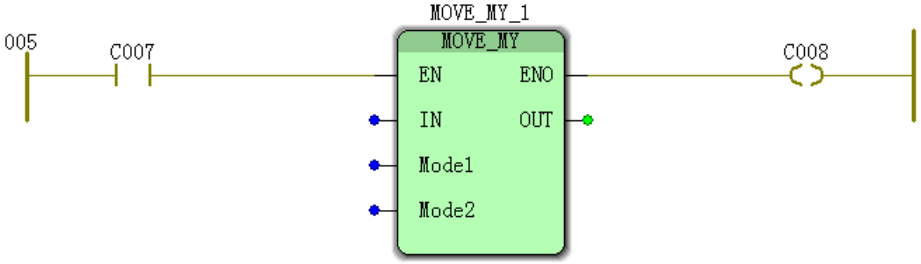
然后点击“制作”，编译通过后，回到 LD 编辑区，在编辑向导中将刚刚创建的 MOVE_MY 拖拽到 LD 的编辑区，如下图



这里使用默认名称 MOVE_MY_1，点击确定，在 LD 编辑区出现了这个功能块，如下图



把这个 MOVE_MY_1 功能块拖拽到前面插入的梯形图网络中，使得功能块的 EN 连接着触点 C007，ENO 连接着线圈 C008，如下图



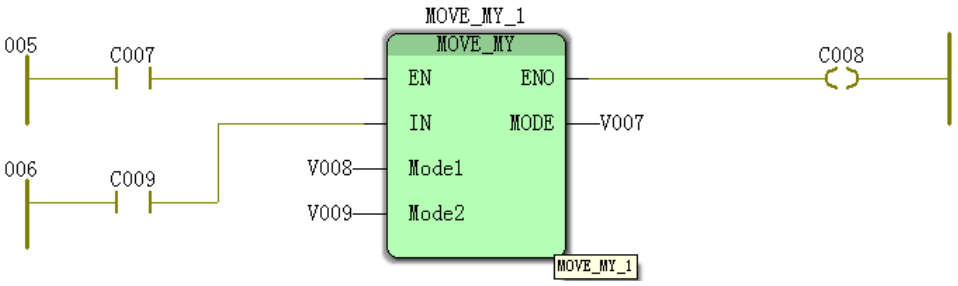
这里假定要控制一个设备的起停，它有两种工作模式，当触点 C007 闭合时，线圈 C008 导通，设备运行，设备的运行方式通过 MOVE_MY_1 功能块的 IN 引脚选择，并通过

MOVE_MY_1 功能块的 MODE 引脚连接变量 V007 送给设备：

当 MOVE_MY_1 功能块的 IN 引脚连接的变量 C009 为 1 时，运行方式 MODE 为 Mode1 引脚连接的变量 V008；

当 MOVE_MY_1 功能块的 IN 引脚连接的变量 C009 为 0 时，运行方式 MODE 为 Mode2 引脚连接的变量 V009。

最终的梯形图如下



6.6 SFC 顺序功能图编程语言

SFC 顺序功能图编程语言可将一个复杂的控制程序分成若干个小的任务，每个小任务以次序顺序执行。

SFC 顺序功能图编程语言中将每个小任务称为一个“步”，“步”与“步”之间的承接关系称为“转换”，每个“步”带有一个动作，“步”、“转换”和“动作”之间用“连线”连接。

一个“步”可以关联多个动作。

动作由一个动作本体和动作限定符组成，动作限定符说明动作是如何与步关联的，当 SFC 步变为活动状态时，所关联的动作会根据动作限定符而被执行。动作既可以连接一个布尔变量，也可以是一个 IL、ST、LD、FBD 程序(被称为'明细')。

转换代表了通过下一步继续处理的情况。如果一个转换变为 TRUE，则前一个步被再执行一次，并且后一个的步成为活动状态。转换既可以是一个布尔变量，也可以是使用 FBD 或 LD 编写的直接连接的布尔表达式。也可以在另一个被称为明细的程序中，编辑将要执行的代码。

所连接对象的集合被称为 SFC 网络。一个 SFC 网络必须具有一个初始步，当调用该 SFC POU 时，这个初始步是第一个要被执行的步。可以在 SFC 网络内插入并行分支(同步地执行)或选择分支(可选择地执行)。

6.6.1 创建一个 SFC 程序

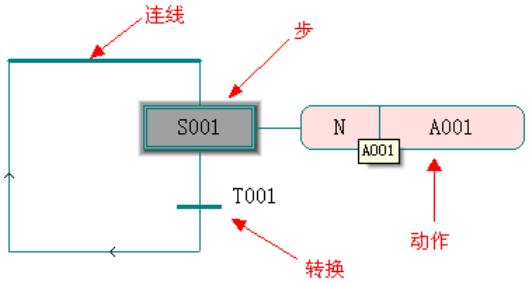
下面我们要创建一个红绿灯的控制程序。

6.6.1.1 创建 SFC 网络

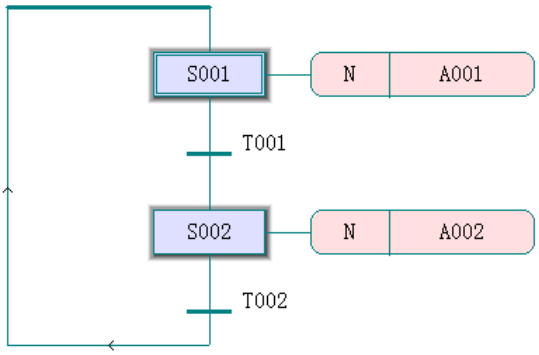
创建一个工程，在创建过程中，选择程序语言是 SFC，程序名是 TrafficLight。

进入 TrafficLight 的编辑区，然后点击编辑区左侧的“创建步转换序列”，如下图

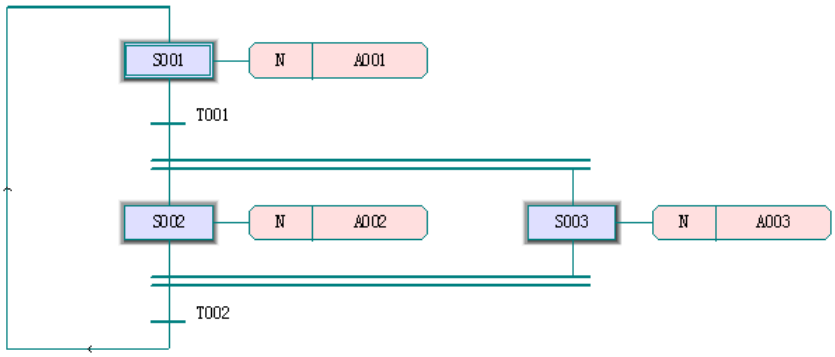
在编辑区就出现了一个 SFC 的“步”S001，如下图



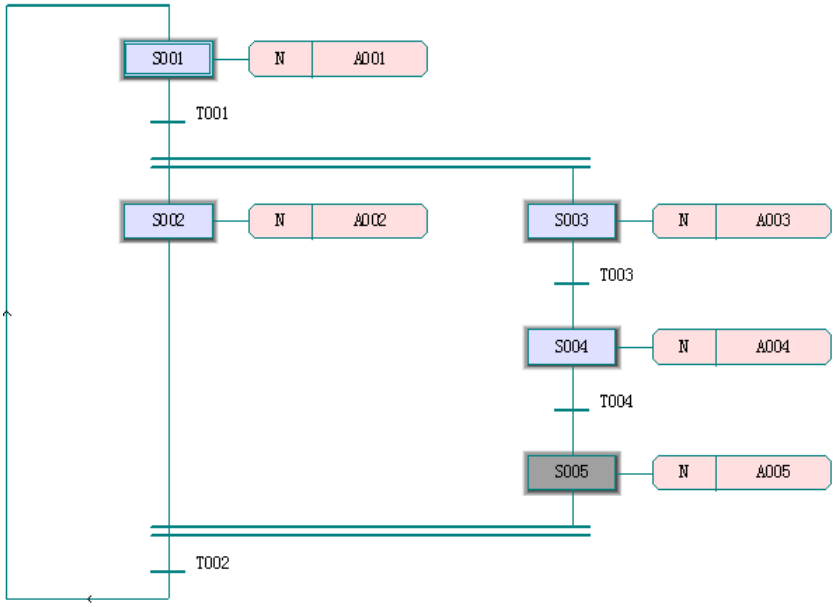
上图中，每个“步”、“转换”和“动作”都有唯一的名称，接着，点击上图中的“转换”T001，再点击“创建步转换序列”，则在这个步的下面增加了一个“步”S002，如下图。



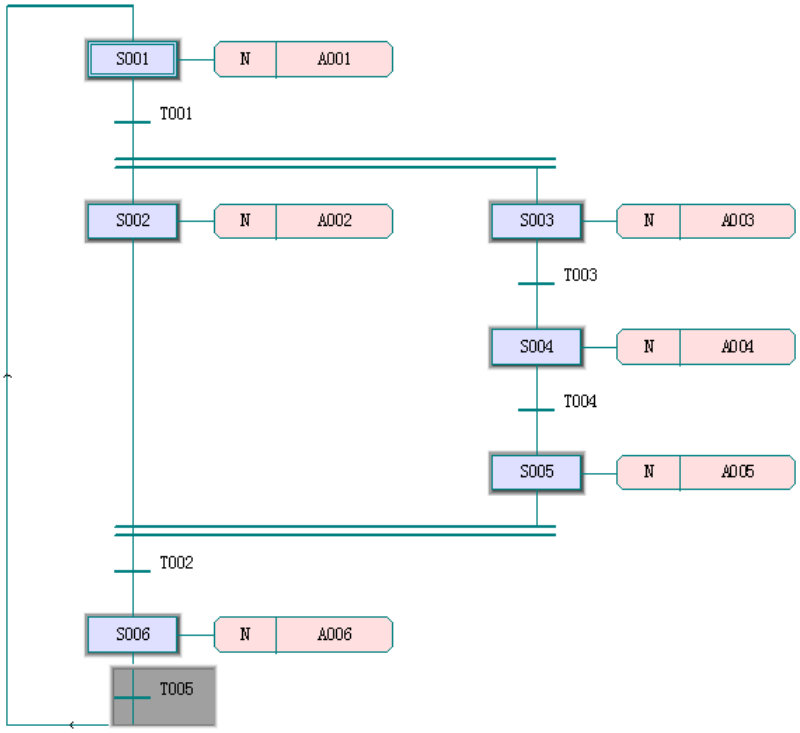
点击“步”S002，再点击“插入 SFC 分支”，则在“步”S002 的右侧插入了一个分支，分支带有一个“步”S003，如下图。



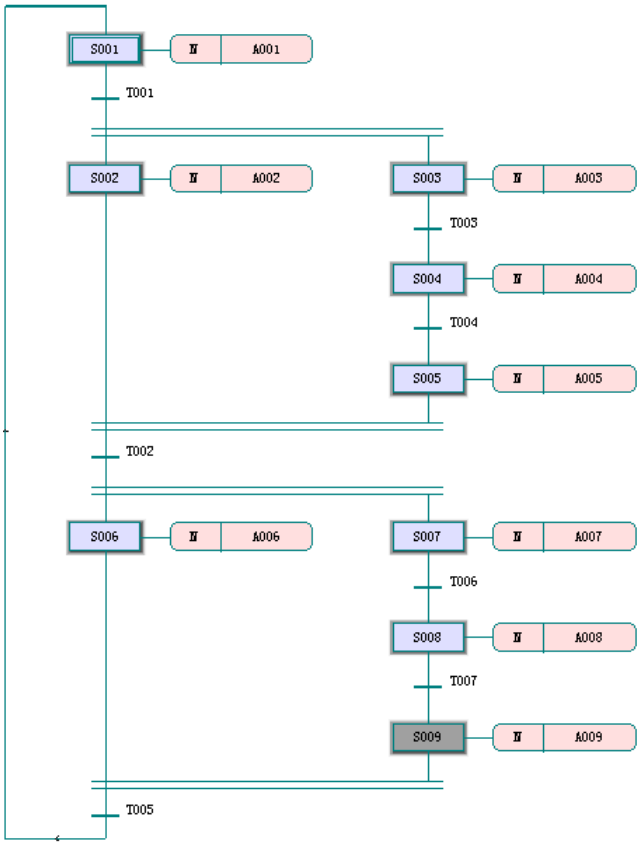
点击“步”S003，再点击“创建步转换序列”，则在 S003 的下面插入了一个“步”S004，依次，再插入一个“步”S005，如下图



点击“转换”T002，再点击“创建步转换序列”，插入 1 个步，如下图



点击“步”S006，再点击“插入 SFC 分支”，则在“步”S006 的右侧插入了一个分支，分支带有一个“步”S007，然后点击“步”S007,再插入两个“步”S008 好 S009，如下图



至此，一个 SFC 网络创建完毕了，其中，S001 是起始步。

6.6.1.2 编写程序

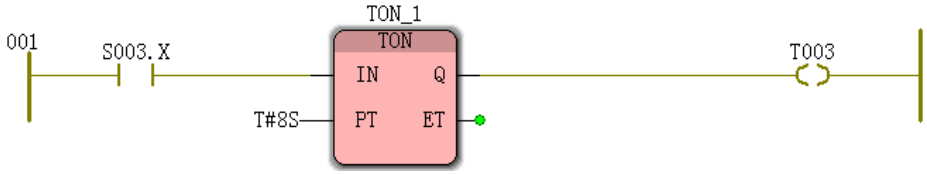
转换条件

双击“转换”T001，在弹出的对话框中选择 LD 编程语言，如下图

点击确定，进入 T001 的编程，在编辑区插入一个网络，并把触点的变量名改为 C000，线圈的变量名改为 T001，如下图

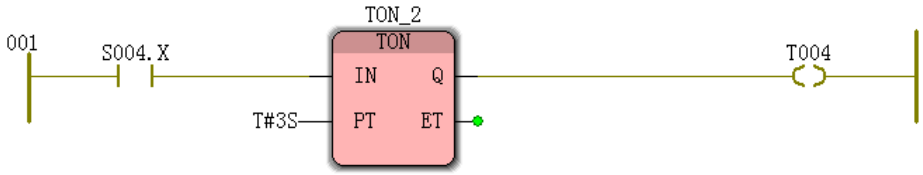
这个转换条件是启动 SFC 程序，因为“步”S001 是起始步，始终为 1，当全局变量 C000 置 1 时，“转换”T001 为 1，进入“步”S002 和 S003。

双击“转换”T003，选择 LD 编程语言，插入如下程序



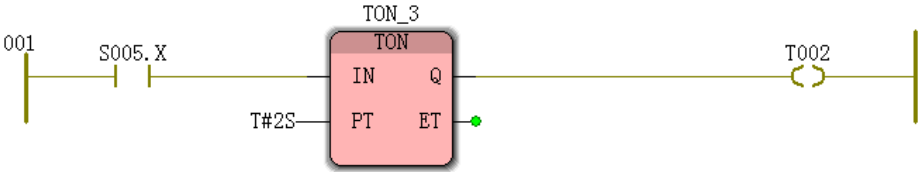
这个转换条件用于计时，当“步”S003 激活 8 秒后，“转换”T003 置 1，进入“步”S004。

双击“转换”T004，选择 LD 编程语言，插入如下程序



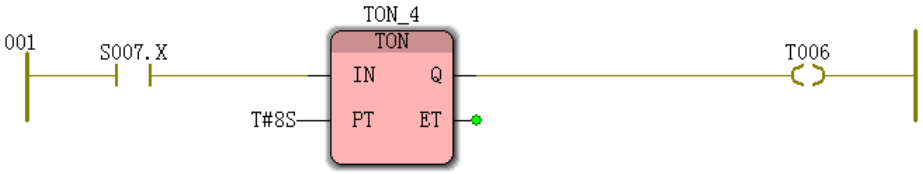
这个转换条件用于计时，当“步”S004 激活 3 秒后，“转换”T004 置 1，进入“步”S004。

双击“转换”T002，选择 LD 编程语言，插入如下程序



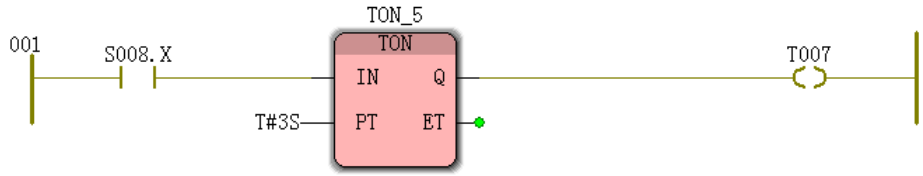
这个转换条件用于计时，当“步”S005 激活 2 秒后，“转换”T002 置 1，进入“步”S006 和 S007。

双击“转换”T006，选择 LD 编程语言，插入如下程序



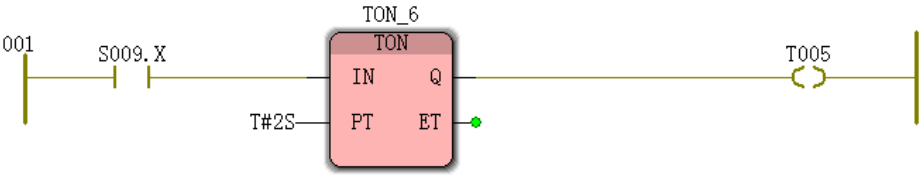
这个转换条件用于计时，当“步”S007 激活 8 秒后，“转换”T006 置 1，进入“步”S008。

双击“转换”T007，选择 LD 编程语言，插入如下程序



这个转换条件用于计时，当“步”S008 激活 3 秒后，“转换”T007 置 1，进入“步”S009。

双击“转换”T005，选择 LD 编程语言，插入如下程序



至此，SFC 的转换条件编程完毕。

动作

删除“动作”A001；

将“动作”A002 名称改为 NS_RED；

将“动作”A003 名称改为 EW_GREEN；

将“动作”A004 名称改为 EW_GREEN_F；

将“动作”A005 名称改为 EW_YELLOW；

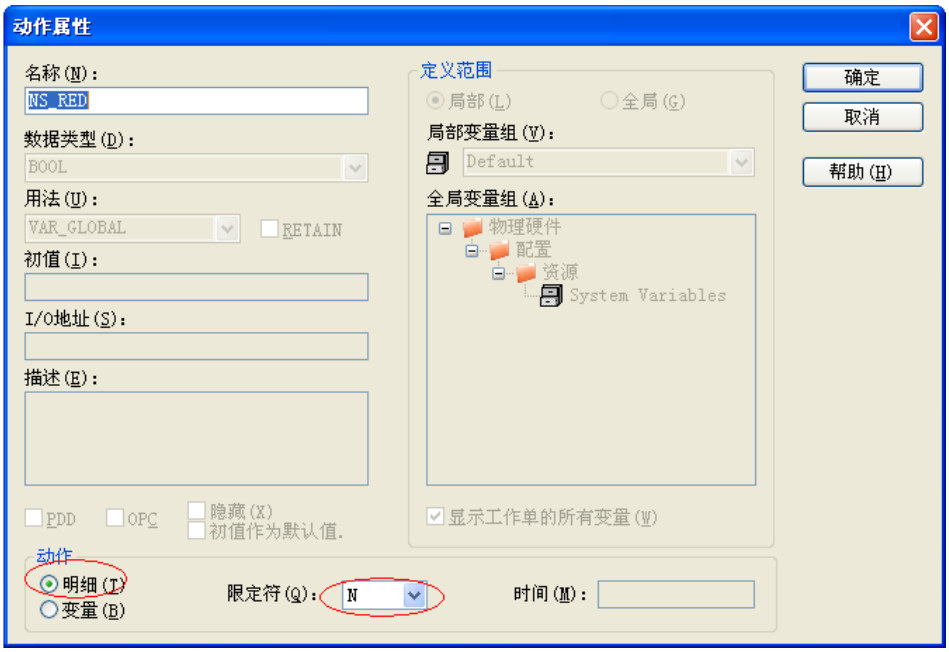
将“动作”A006 名称改为 EW_RED；

将“动作”A007 名称改为 NS_GREEN；

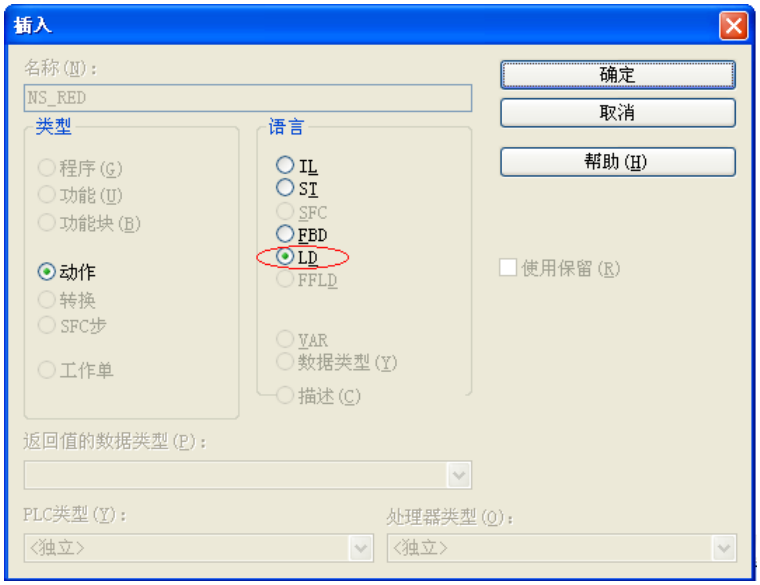
将“动作”A008 名称改为 NS_GREEN_F；

将“动作”A009 名称改为 NS_YELLOW。

双击“动作”NS_RED，在弹出对话框中选择动作类型为明细，限定符为 N，如下图



点击确定，再双击“动作”NS_RED，弹出对话框中选择 LD 编程语言，如下图



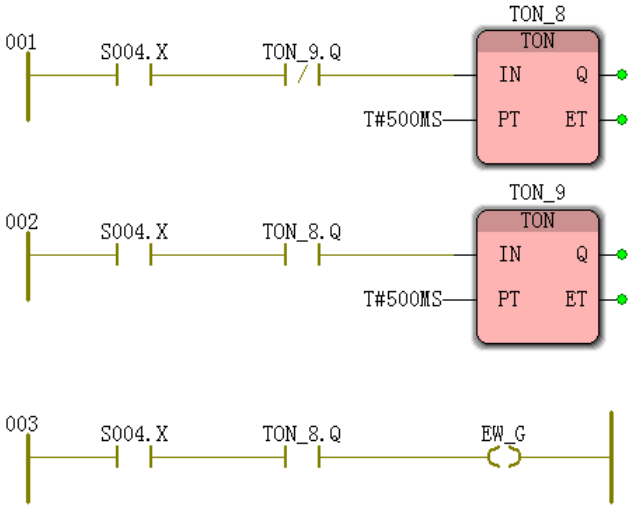
点击确定进入 NS_RED 的编程，在编辑区插入如下程序

其中，线圈 NS_R 的 I/O 地址为%Q0.0，表示“步”S002 激活时南北方向亮红灯。

将 EW_GREEN 改为明细，选择 LD 编程，进入 EW_GREEN 的编程，在编辑区插入程序

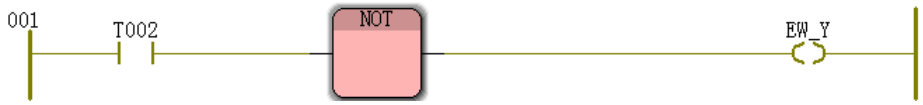
其中，线圈 EW_G 的 I/O 地址为%Q0.5，表示“步”S003 激活时东西方向亮绿灯。

将 EW_GREEN_F 改为明细，选择 LD 编程，进入 EW_GREEN_F 的编程，在编辑区插入如下程序



表示“步”S004 激活时东西方向绿灯闪烁

将 EW_YELLOW 改为明细，选择 LD 编程，进入 EW_YELLOW 的编程，在编辑区插入程序



其中，线圈 EW_Y 的 I/O 地址为%Q0.6，表示“转换”T002 激活时东西方向亮黄灯。

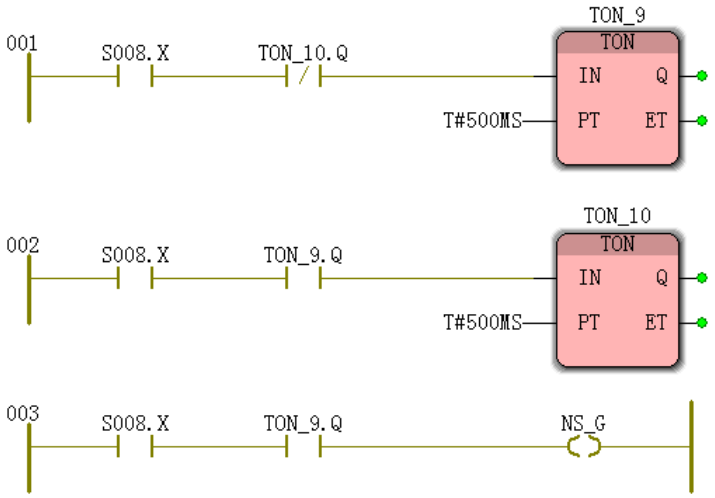
将 EW_RED 改为明细，选择 LD 编程，进入 EW_RED 的编程，在编辑区插入程序

其中，线圈 EW_R 的 I/O 地址为%Q0.4，表示“步”S006 激活时东西方向亮红灯

将 NS_GREEN 改为明细，选择 LD 编程，进入 NS_GREEN 的编程，在编辑区插入程序

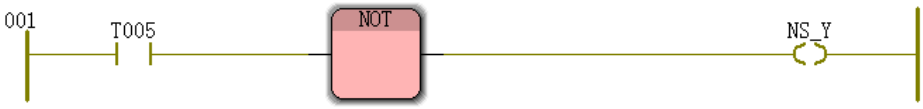
其中，线圈 NS_G 的 I/O 地址为%Q0.1，表示“步”S007 激活时东西方向亮绿灯。

将 NS_GREEN_F 改为明细，选择 LD 编程，进入 NS_GREEN_F 的编程，在编辑区插入程序



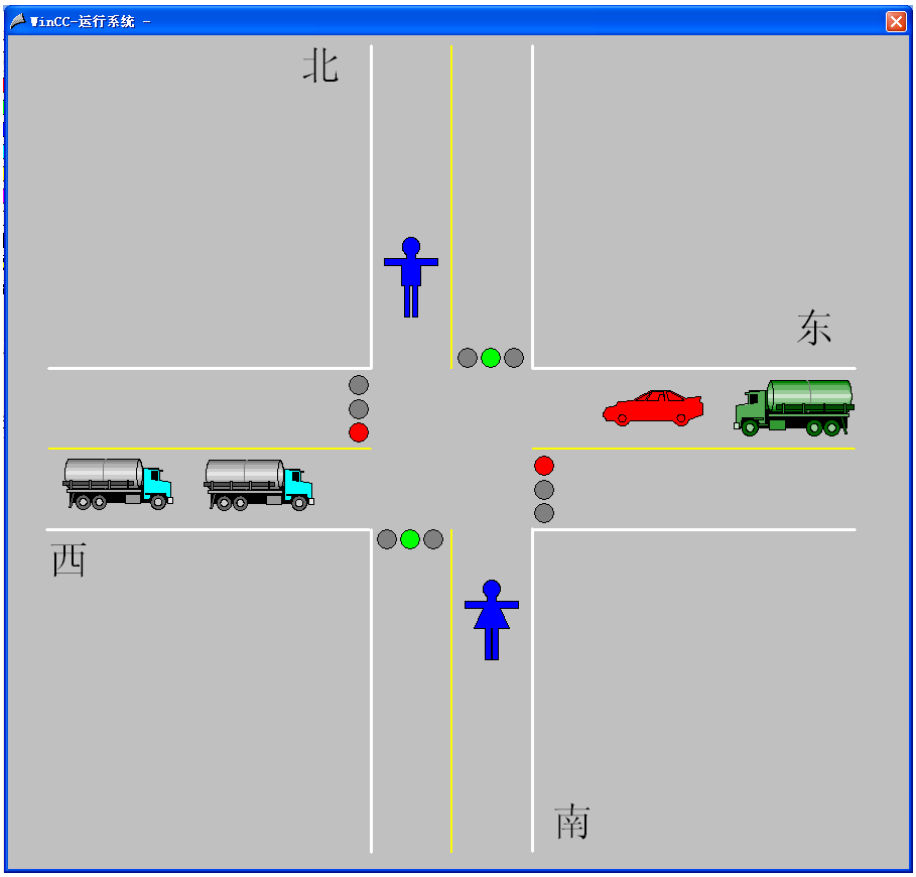
表示“步”S008 激活时南北方向绿灯闪烁

将 NS_YELLOW 改为明细，选择 LD 编程，进入 NS_YELLOW 的编程，在编辑区插入程序



其中，线圈 EW_Y 的 I/O 地址为%Q0.6，表示“转换”T005 激活时南北方向亮黄灯

至此，一个完整的十字路口红绿灯控制程序完成了，点击“制作”并下载到 PLC 后，程序开始工作。这个程序是模拟的交通十字路口的红绿灯的自动控制，即当南北方向红灯亮时，保持 13 秒，东西方向绿灯亮，保持 8 秒，再闪烁 2 秒，然后黄灯亮，保持 2 秒，然后，东西方向红灯亮，南北方向绿灯亮，下图是用 WinCC 做的一个交通画面，连接到 PLC 后，可以看到红绿灯工作过程。



6.6.2 SFC 的动作限定符

SFC 动作包括动作限定符和动作本体，动作限定符说明动作是如何与步关联的，以下的动作限定符都是可用的。

限定符	描述	功能
N	不保存的	只要步是活动的，就执行动作代码本体或置位布尔变量。
R	超越复位	不再执行动作代码本体或者复位布尔变量。在使用‘S’限定符之前，必须要置位动作。
S	置位（保存的）	执行动作代码本体或者置位布尔变量。一旦(相关联的)步变为活动状态，就保存这个(置位)状态。该置位状态，仅能通过使用‘R’限定符，将该动作关联到另一个不同的步上，来显式地复位。
L	限时的	只要步是活动的，就执行动作代码本体或置位布尔变量，但最多能持续所设定的时间间隔这样一段时间。
D	延时的	在所设定的延迟时间流逝之后，再执行动作代码本体或置位布尔变量。只要步是活动的，动作就保持活动状态。如果步处于活动状态的时间比所设定的延迟时间短，则动作不会变为活动状态。
P	脉冲	步一变为活动状态，动作代码本体就执行一个操作周期，或布尔变量就置位一个操作周期。
SD	保存并延时	在步被激活之后，当又流逝了所设定的延迟时间之时，就执行动作代码本体或保存并置位布尔变量，即使步又变为不活动状态。该动作在被复位之前，将一直保持活动状态。如果步处于活动状态的时间比所设定的延迟时间还短，则该动作无论如何也会变为活动状态的。
DS	延迟并保	在步被激活之后，当又流逝了所设定的延迟时间之时，就执行动作代码本体或保存

	存	并置位布尔变量。该动作在被复位之前，将一直保持活动状态。如果步处于活动状态的时间比所设定的延迟时间短，则动作不会变为活动状态。
SL	保存并限时	只要步是活动的，就在一个固定时间间隔内，执行动作代码本体或置位并保存布尔变量。如果步处于活动状态的时间比该时间间隔短，则该动作也无论如何会在整个时间间隔内处于活动状态。如果在该时间间隔内，动作被复位，该动作马上就会变为不活动状态。

7. 腾控开发的指令

包括 FileAccess、HTIME 和 TCNETLIB 共三组指令集，使用这些指令时，需要插入这些固件库。

7.1 文件读写 FileAccess

使用以下指令，可以将在 PLC 中建立一个.txt、.csv 或其它格式的文件，对文件进行打开、读、写、查找、移除等操作，此外，用户还可使用“CuteFTP 8 Professional”软件浏览 PLC。

7.1.1 FileOpen

功能

FileOpen 指令用于打开 PLC 用户存储区的一个文件，准备读写，当文件不存在时就在 PLC 用户存储区创建一个文件并打开。

用法

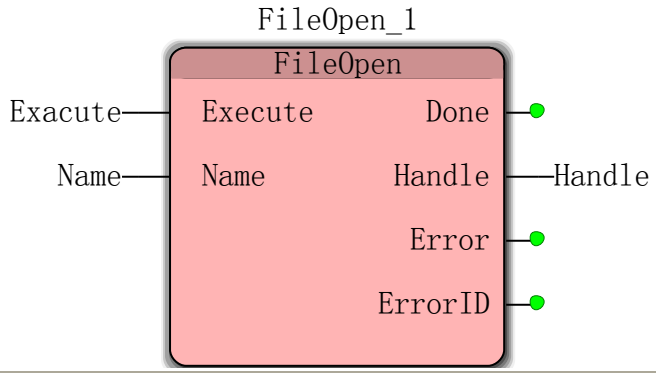
IL 编程语言

```
LD Execute
ST FileOpen_1.Execute
LD Name
ST FileOpen_1.Name
CALFileOpen_1
```

ST 编程语言

```
FileOpen_1(Execute := Execute,
Name := Name ) ;
Handle := FileOpen_1.Handle;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 Execute、Name 或使用常量

FileOpen 指令处理的数据类型

参数	数据类型	描述
Execute	BOOL	上升沿，打开/创建文件，文件操作期间需要保持 ON 的状态
Name	STRING	将要打开/创建的文件的名称和后缀名，如’my.txt’
Done	BOOL	0: 功能块不能被执行；1: 功能块已经被执行
Handle	UINT	打开/创建文件的文件句柄
Error	BOOL	0: 当打开/创建文件时，没有错误 1: 当打开/创建文件时，产生错误。
ErrorID	UINT	错误编号： 0: 文件正常打开 2: 创建文件失败 4: 文件已经被打开 5: 文件被写保护或者访问被拒绝

		6: 文件名错误

注意：

- 在同一时间内，最大限度可以打开 8 个文件。
- 文件名称的数据类型必须是 STRING。文件名称的长度包括路径在内必须不能超过 80 个字符。不允许用户定义字符串。
- 输出 Done、Error 和 ErrorID 的状态一直保留到输入 Execute 处检测到一个下降沿为止。

7.1.2 FileSeek

功能

FileSeek 指令用于在已打开的文件中将光标移动到某个位置。

用法

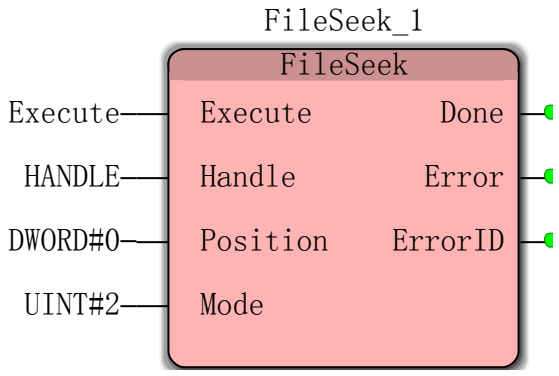
IL 编程语言

```
LD Execute
ST FileSeek_1.Execute
LD Handle
ST FileSeek_1.Handle
LD DWORD#8
ST FileSeek_1.Position
LD UINT#0
ST FileSeek_1.Mode
CALFileSeek_1
```

ST 编程语言

```
FileSeek_1(Execute := Execute, Handle := Handle, Position := DWORD#8, Mode := UINT#0) ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 Execute、Handle 或使用常量

FileSeek 指令处理的数据类型

输入	数据类型	描述
Execute	BOOL	上升沿有效
Handle	UINT	打开/创建文件的文件句柄
Position	DWORD	将光标移动到文件中某个字符处，0——第一个字符，1——第二个字符，…
Mode	UINT	0——从文件的开始处，1——从当前位置，2——从文件的结尾处
输出	数据类型	描述
Done	BOOL	0: 功能块不能被执行；1: 功能块可以被执行
Error	BOOL	
ErrorID	UINT	错误编号： 1: 句柄错误 13: 模式错误 24: Fileseek 失败

注意：

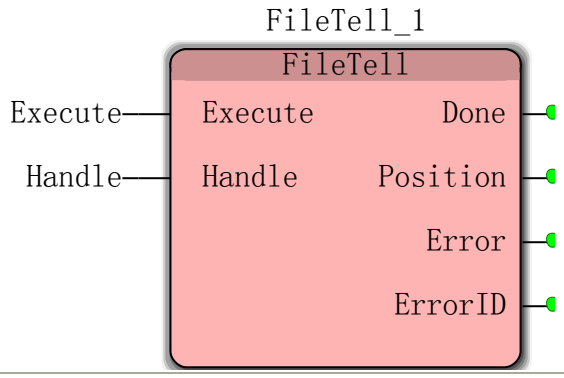
- 文件中的空格、回车、换行，分别视为一个字符：

7.1.3 FileTell

功能

FileTell 指令用于在已打开的文件中找出光标所在的位置。

用法

IL 编程语言	LD、 FBD 编程语言
LD Execute ST FileTell_1.Execute LD Handle ST FileTell_1.Handle CAL FileSeek_1	
ST 编程语言	
FileTell_1(Execute := Execute, Handle := Handle) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 Execute、Handle 或使用常量

FileTell 指令处理的数据类型

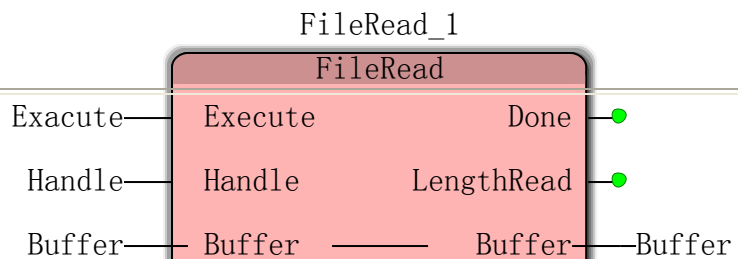
参数	数据类型	描述
Execute	BOOL	上升沿有效
Handle	UINT	打开/创建文件的文件句柄
输出	数据类型	描述
Done	BOOL	0: 功能块不能被执行；1: 功能块可以被执行
Position	DWORD	光标在文件中的位置（某个字符处），0——第一个字符，1——第二个字符，…
Error	BOOL	
ErrorID	UINT	错误编号： 1: 句柄错误

7.1.4 FileRead

功能

FileOpen 指令用于读取已打开的文件中的数据，读出的数据以 ASCII 码的形式存放在数据缓冲区中。

用法

IL 编程语言	LD、 FBD 编程语言
LD Execute ST FileRead_1.Execute LD Handle	

ST FileRead_1.Handle LD Buffer ST FileRead_1.Buffer LD MaxLength ST FileRead_1.MaxLength CALFileOpen_1	
ST 编程语言	
FileRead_1(Execute := Execute, Handle := Handle, Buffer := Buffer, MaxLength := MaxLength) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 Execute、Handle 等或使用常量	

FileRead 指令处理的数据类型

参数	数据类型	描述
Execute	BOOL	上升沿有效，从已打开的文件中光标起始处读数据
Handle	UINT	由 FileOpen 指令打开的文件句柄
Buffer	ANY	从文件中读出的数据要放在一个数据缓冲区中，Buffer 就是这个缓冲区，使用中一般将数据类型设为数组
MaxLength	UDINT	要读的最大字符数
Done	BOOL	0: 功能块不能被执行 1: 功能块可以被执行
LengthRead	UDINT	已读出字符的数量，包括换行符，一个字符占用一个字节，
Error	BOOL	0: 读的时候，没有错误产生 1: 读的时候，产生一个错误
ErrorID	UINT	当读的时候，产生的错误的错误编号： 0: 没有错误 1: 句柄错误 10: 已达到数据的末尾 12: 要读字符的数量比数据缓冲区的字符数量大 22: 读取失败

注意：

- 可以用不同方法来声明要读数据的数据缓冲区。数据缓冲区类型是一个用户定义的数据类型，例如，Byte Array。数据类型如以下所声明为例：
  TYPE
  FileBuffer :ARRAY [1..100] OF BYTE;
  END_TYPE
  在这种情况下，数据缓冲区长度为 100 个字符。
- 字符串不能直接用作数据缓冲区。如果不能将读数据处理成字符串，那么首先必须在数组中存储数据，然后使用 ProConOS 功能块 BUF_TO_STRING 将其转换成字符串。
- 输出 Done、LengthRead、Error 和 ErrorID 的状态一直保留到输入 Execute 处检测到一个下降沿为止。

7.1.5 FileWrite

功能

FileWrite 指令用于向已打开的文件中的写数据，将要写的数据应以 ASCII 码的形式存放在数据缓冲区中。

用法

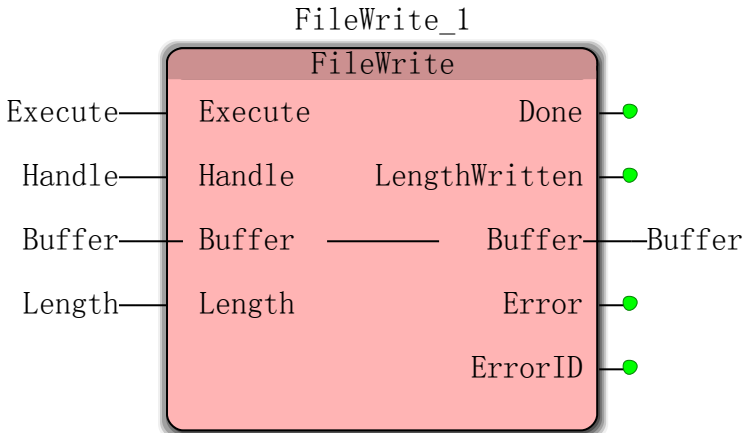
IL 编程语言

```
LD Execute
ST FileWrite_1.Execute
LD Handle
ST FileWrite _1.Handle
LD Buffer
ST FileWrite _1.Buffer
LD Length
ST FileWrite _1.Length
CALFileOpen_1
```

ST 编程语言

```
FileWrite _1(Execute := Execute, Handle := Handle, Buffer := Buffer, Length := Length) ;
```

LD、 FBD 编程语言



注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 Execute、Handle 等或使用常量

FileWrite 指令处理的数据类型

参数	数据类型	描述
Execute	BOOL	引脚 Execute 有上升沿时，向已打开的文件起始处写数据
Handle	UINT	由 FileOpen 指令打开的文件句柄
Buffer	ANY	将要向文件中写的数据应先放在一个数据缓冲区中，Buffer 就是这个缓冲区，使用中一般将数据类型设为数组
Length	UDINT	要写的字符数
Done	BOOL	0: 功能块不能被执行。 1: 功能块可以被执行。
LengthWriten	UDINT	已写入文件的字符数量，包括回车、换行符，一个字符占用一个字节，
Error	BOOL	0: 写的时候，没有错误产生。 1: 写的时候，产生一个错误。
ErrorID	UINT	当写的时候，产生的错误的错误编号： 0: 没有错误 1: 句柄错误 10: 没有内存可供写数据使用 12: 要写字符的数量比数据缓冲区的字符数量大 22: 没有数据能被写 23 写长度错误

注意：

- 可以用不同方法来声明要写数据的数据缓冲区。数据缓冲区类型是一个用户定义的数据类型，例如，Byte Array。数据类型如以下所声明为例：

```
TYPE
    FileBuffer : ARRAY [1..100] OF BYTE;
END_TYPE
```

在这种情况下，数据缓冲区长度为 100 个字符。

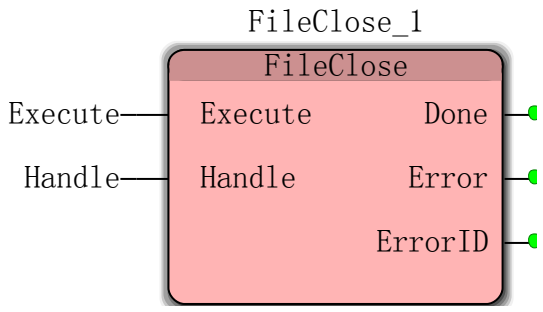
- 字符串不能直接用作数据缓冲区。如果字符串能够被存储在一个文件夹里，那么首先必须使用 ProConOS 功能块 STRING_TO_BUF 将它存储在一个数组中。
- 输出 Done、LengthWritten、Error 和 ErrorID 的状态一直保留到输入 Execute 处检测到一个下降沿为止。
- Length 值不能太大，否则出错，如 5000 是可以的，8000 就可能出错了。

7.1.6 FileClose

功能

FileClose 指令用于关闭已经打开的文件。

用法

IL 编程语言	LD、 FBD 编程语言
LD Execute ST FileClose_1.Execute LD Handle ST FileClose_1.Handle CALFileClose_1	
ST 编程语言	
FileClose_1(Execute := Execute, Handle := Handle) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 Execute、Handle 或使用常量

FileOpen 指令处理的数据类型

参数	数据类型	描述
Execute	BOOL	引脚 Execute 有上升沿时，关闭文件
Handle	UINT	要关闭的文件句柄（这个文件由 FileOpen 已经打开）
Done	BOOL	0: 功能块不能被执行；1: 功能块已经被执行
Error	BOOL	0: 当关闭文件时，没有错误 1: 当关闭文件时，产生错误
ErrorID	UINT	错误编号： 0: 文件正常关闭 1: 句柄错误 20: 文件不能被关闭

注意：

- 当终止应用程序时，要想自动关闭已打开的文件，那么对于系统程序中的每个已打开的文件，必须调用功能块 FileClose。

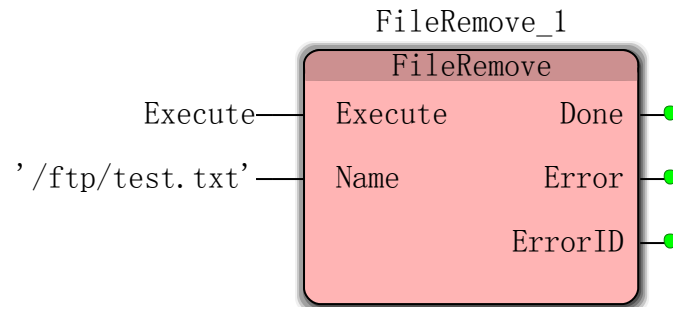
- 输出 Done、Error 和 ErrorID 的状态一直保留到输入 Execute 处检测到一个下降沿为止。

7.1.7 FileRemove

功能

FileRemove 指令用于删除 PLC 中存在的一个文件。

用法

IL 编程语言	LD、 FBD 编程语言
LD Execute ST FileClose_1.Execute LD Handle ST FileClose_1.Handle CALFileClose_1	
ST 编程语言	
FileClose_1(Execute := Execute, Handle := Handle) ;	

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量 Execute、Handle 或使用常量

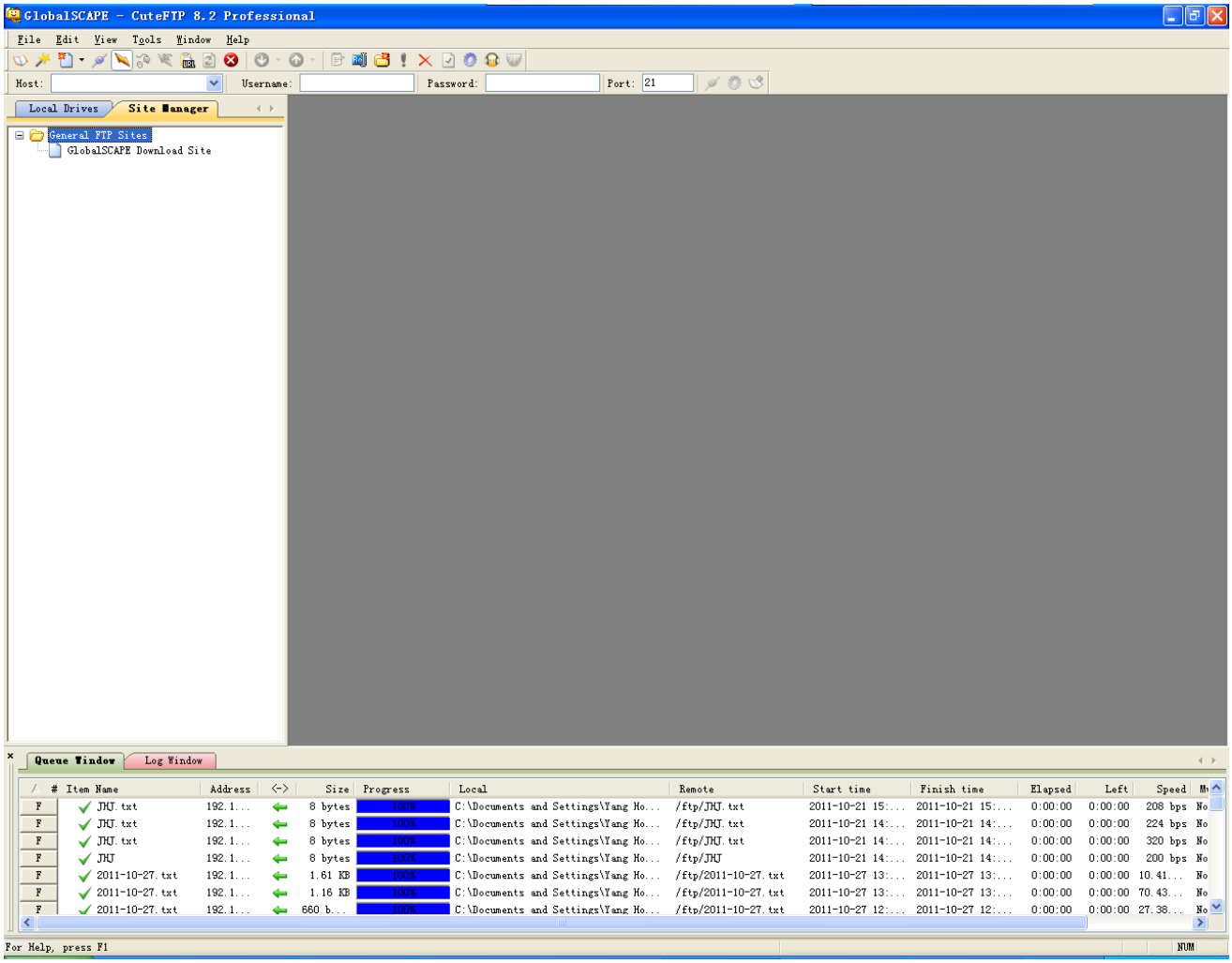
FileOpen 指令处理的数据类型

参数	数据类型	描述
Execute	BOOL	上升沿有效
Name	STRING	字符串要用一对单引号，即前缀“/ftp/”+ “将要删除的文件的名称和后缀名”，如’/ftp/my.txt’
参数	数据类型	描述
Done	BOOL	0: 功能块不能被执行；1: 功能块已经被执行
Error	BOOL	0: 当关闭文件时，没有错误 1: 当关闭文件时，产生错误
ErrorID	UINT	错误编号： 21:删除失败

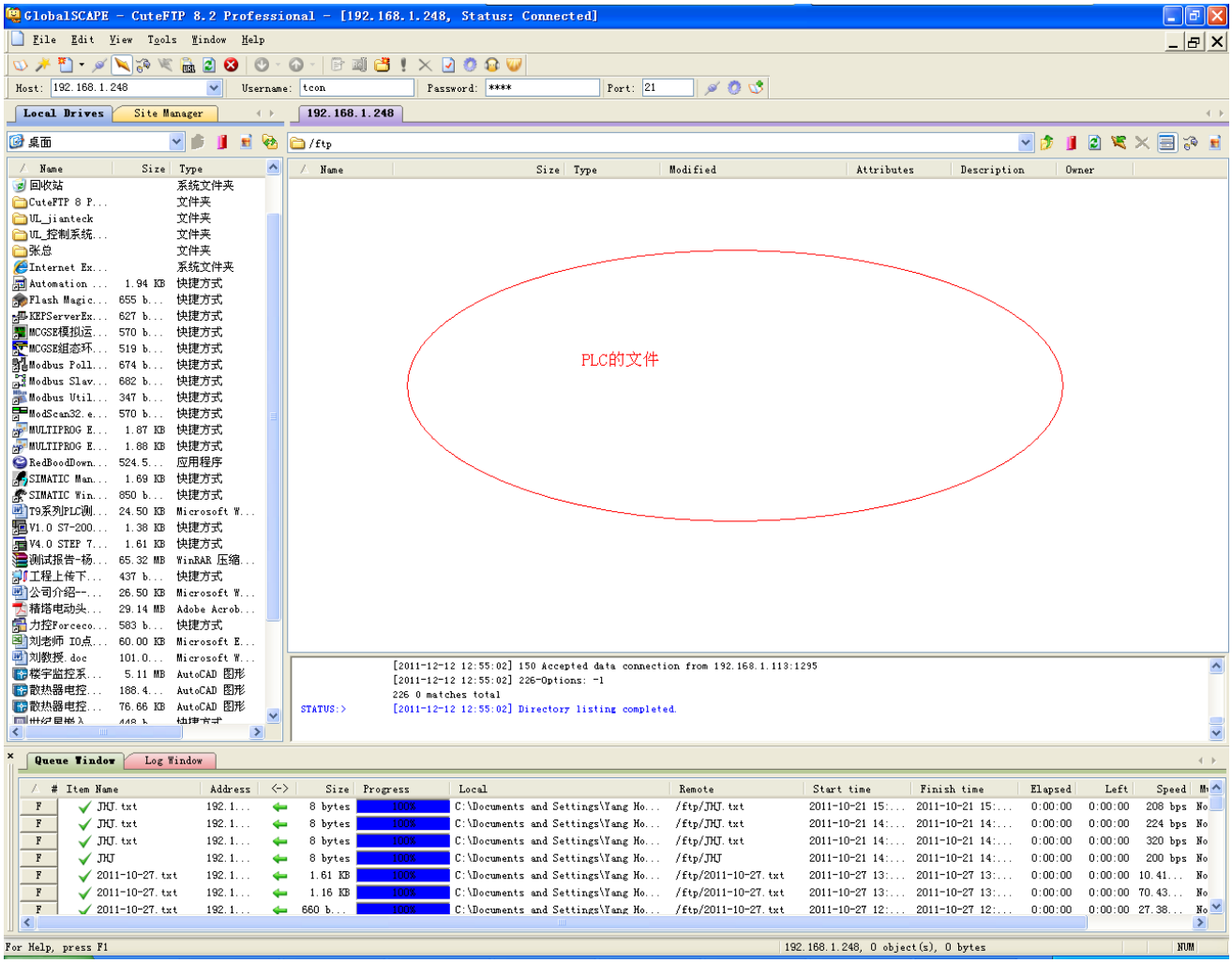
7.1.8 CuteFTP 8 Professional

CuteFTP 8 Professional 是一个 FTP 客户端程序，通过它可以访问对 PLC 进行文件传输访问。在 CuteFTP 8 Professional 文件夹中，双击可执行程序，cteftppro.exe，如下

软件启动后如下



在工具栏中的 Host 栏，填入要访问的 PLC 的 IP 地址，如 192.168.1.99；在 Username 栏填入用户名——tcon；在 Password 栏填入用户密码——tcon，点击连接即可，如下图



7.2 高速计数 HTIME

编码器（或其它高速脉冲发生装置）接入 PLC 的高速脉冲计数通道后，需要设置高速计数通道的工作方式，才能计数。

HTIME 有 HTIME_SET、HTIMEA_SET、HTIMEB_SET、HTIMEAB_SET 共 4 个指令，分别用于两相高速脉冲计数工作方式设置、第 1 路单相高速脉冲计数清零、第 2 路单相高速脉冲计数清零、两相高速脉冲计数清零。

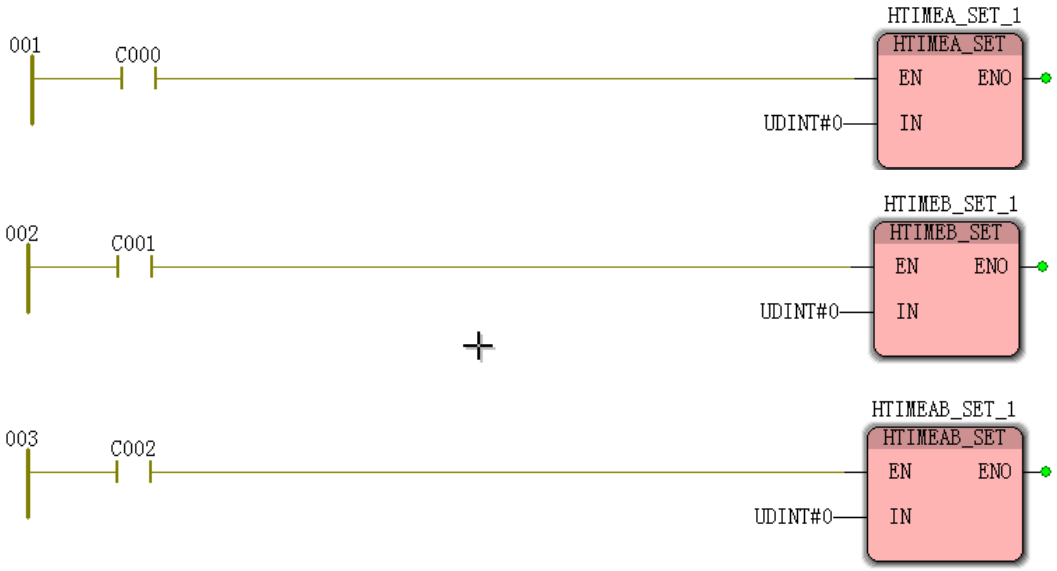
编程过程：

首先，要建立一个工程，创建一个 LD 程序，在“编辑向导”中选择“HTIME”，把 HTIME_SET 拖拽到编辑区，并建立一个 BOOL 变量用于使能 HTIME_SET，和一个 BYTE 变量存储高速计数通道的工作方式，如下

其中，BOOL 变量 C003 从 0 变为 1 后（上升沿有效）启动 HTIME_SET 指令，BYTE 变量 V006 存储着高速脉冲计数的工作方式：0 为普通 DI，1 为单相高速脉冲计数，2 为两相高速脉冲计数。

设置高速脉冲计数的共组方式，也可通过 ModScan 修改 400047 号保持寄存器地址的内容：0 为普通 DI，1 为单相高速脉冲计数，2 为两相高速脉冲计数。

其次，建立一个 UDINT 型变量，变量的地址为%ID192，通过这个变量就能获得第 1 路高速脉冲计数值，如果需要第 2 路高速脉冲计数值，把地址改为%ID196 即可，两相高速脉冲计数值的地址为%ID200（见硬件手册）。如果需要对第 1、第 2 和两相高速脉冲计数值清零，则需添加如下程序



上述 3 个指令分别给第 1 路、第 2 路和两相高速脉冲计数值清零，指令的 EN 引脚均是上升沿有效。

再次，如果需要获得频率值，则建立一个 UDINT 型变量，变量的地址为%ID208，通过这个变量就能获得第 1 路高速脉冲计数的频率值，如果需要第 2 路高速脉冲计数的频率值，把地址改为%ID212 即可，两相高速脉冲计数的频率值的地址为%ID216（见硬件手册）。如果需要修改频率的单位，则可通过 ModScan 修改 400048 和 400049 号保持寄存器地址的内容（分别对应第 1 路和第 2 路高速脉冲计数的闸门时间），默认值为 100，单位为 10ms，即实际频率单位为 100×10=1s。

7.3 通讯 TCNETLIB

包括自由口编程、TCNET、校验等三种指令。

7.3.1 自由口通讯

自由口通讯是指用户的通讯协议不是标准的，而只用户自定义的各种通讯协议，PLC 可以完成与第三方设备的数据通信，对自由口通讯的要求如下：

- 第三方设备具有 RS232 或 RS485 口
- 如果是通过 PLC 的 RS232 口通讯，则要与第三方设备的 RS232 口直连，即 2 接 2，3 接 3，5 接 5
- 如过是通过 PLC 的 RS485 口通讯，则 A 接 A，B 接 B
- 设置 PLC 与第三方设备的波特率、数据位、校验位、停止位等信息，使其保持一致
- 理解第三方设备的数据格式，包括起始字符、数据内容，消息的长度、结束字符等

自由口通讯编程时，首先是打开 PLC 的串口，然后是发送读或写命令，北京腾控科技的

自由口通讯包括 PORT_OPEN、PORT_WRITE 和 PORT_READ 共 3 个指令。

通过 485 口，最多支持 32 个从站的自由口编程。

7.3.1.1 打开串口 PORT_OPEN

功能

PORT_OPEN 指令用于打开 PLC 的一个串口并设置串口的通讯速率，在通过串口读写过程中，PORT_OPEN 指令需要一致有效。

用法

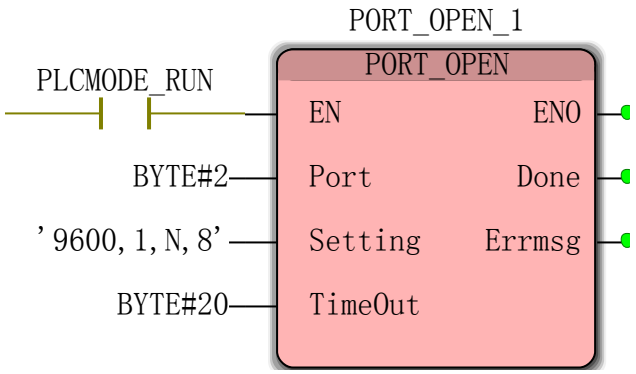
IL 编程语言

```
LD PLCMODE_RUN
ST PORT_OPEN_1.EN
LD BYTE#2
ST PORT_OPEN_1.Port
LD '9600,1,N,8'
ST PORT_OPEN_1.Setting
LD BYTE#20
ST PORT_OPEN_1.TimeOut
CALPORT_OPEN_1
```

ST 编程语言

```
PORT_OPEN_1(EN:=  PLCMODE_RUN,
Port:=BYTE#2,
Setting:='9600,1,N,8',TimeOut:=BYTE#20);
```

LD、 FBD 编程语言



注：各引脚也可使用变量，也可使用常量

PORT_OPEN 指令处理的数据类型

输入引脚	数据类型	描述
EN	BOOL	为 TRUE 时，打开串口并建立 Modbus 连接,其它程序不可使用此串口；为 FALSE 时，关闭此串口
Port	BYTE	要打开的串口号，可选值：1、2、3，分别对应 COM1、COM2、COM3 口
Setting	STRING	通讯参数，要与从站的通讯格式一致，如'9600,1,N,8'是指 9600 BPS，无校验、8 位数据位、1 位停止位
TimeOut	BYTE	串口操作超时时间，单位：10ms
输出引脚	数据类型	描述
ENO	BOOL	输出使能
DONE	BOOL	为 1 时串口打开成功、为 0 时串口打开失败
Errmsg	WORD	错误信息 0：串口工作正常 1：当前串口无效(注:已被其它程序占用) 2：串口参数格式错误(9600,1,N,8) 3：接收数据奇偶校验错误 4：串口设备忙

		5：串口没打开

7.3.1.2 写数据 PORT_WRITE

功能

PORT_WRITE 指令用于向打开的 PLC 串口上写数据，被写的数据存放在数据缓冲区中，该指令每调用一次写一次数据。

用法

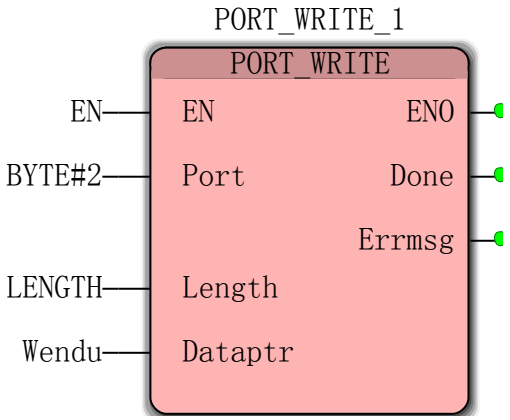
IL 编程语言

```
LD EN
ST PORT_WRITE_1.EN
LD BYTE#2
ST PORT_WRITE_1.Port
LD LENGTH
ST PORT_WRITE_1.Length
LD Wendu
ST PORT_WRITE_1.Dataptr
CALPORT_WRITE_1
```

ST 编程语言

```
PORT_WRITE_1(EN:= EN, Port:=BYTE#2,
Length:=LENGTH,Dataptr:=Wendu);
```

LD、 FBD 编程语言



注：各引脚也可使用变量，也可使用常量

PORT_WRITE 指令处理的数据类型

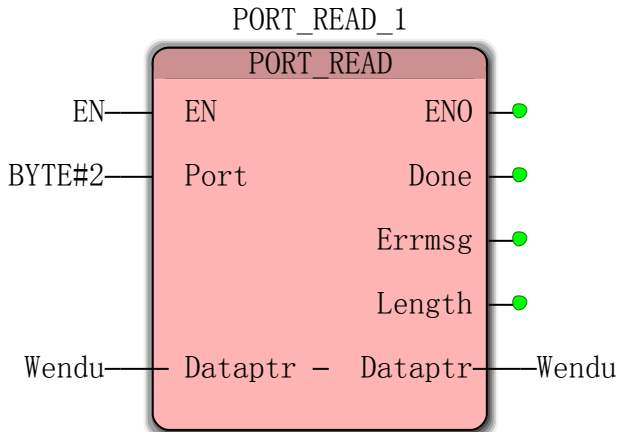
输入引脚	数据类型	描述
EN	BOOL	写命令使能，上升沿有效
Port	BYTE	由 PORT_OPEN 打开的串口号
Length	WORD	需要写入的数据长度（字节数）
Dataptr	ANY	存放数据的缓冲区，一般为数组，也可以是 ANY_BIT 或 ANY_INT
输出引脚	数据类型	描述
ENO	BOOL	输出使能
DONE	BOOL	为 1 时写数据、为 0 时没有写数据
Errmsg	WORD	错误信息 0：串口工作正常 1：当前串口无效(注:已被其它程序占用) 2：串口参数格式错误(9600,1,N,8) 3：接收数据奇偶校验错误 4：串口设备忙 5：串口没打开

7.3.1.3 读数据 PORT_READ

功能

PORT_READ 指令用于从打开的 PLC 串口上读数据。

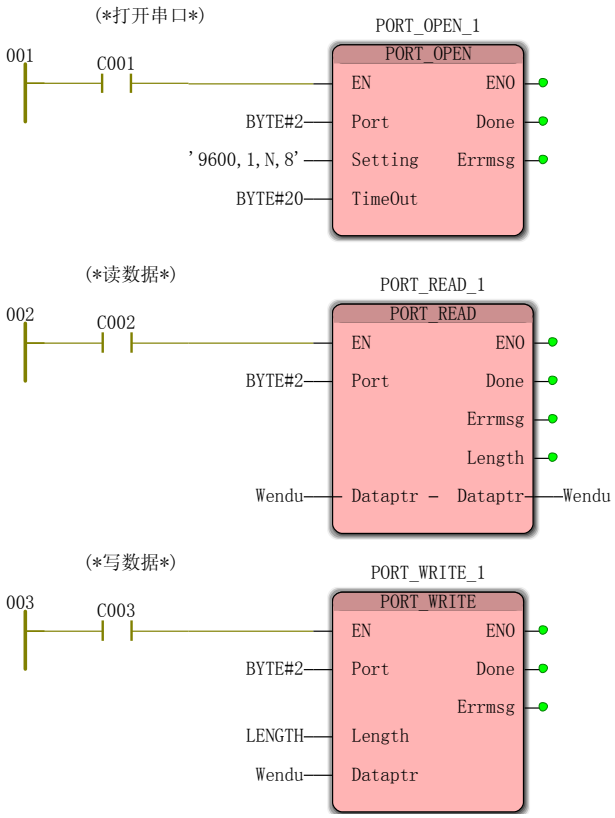
用法

IL 编程语言	LD、 FBD 编程语言
LD EN ST PORT_READ_1.EN LD BYTE#2 ST PORT_READ_1.Port LD Wendu ST PORT_READ_1.Dataptr CALPORT_READ_1	
ST 编程语言	
PORT_READ_1(EN:= EN, Port:=BYTE#2, Dataptr:=Wendu);	
注：各引脚也可使用变量，也可使用常量	

PORT_READ 指令处理的数据类型

输入引脚	数据类型	描述
EN	BOOL	读命令使能，TRUE 有效
Port	BYTE	由 PORT_OPEN 打开的串口号
Dataptr	ANY	存放数据的缓冲区，一般为数组，也可以是 ANY_BIT 或 ANY_INT
输出引脚	数据类型	描述
ENO	BOOL	输出使能
DONE	BOOL	为 1 时读数据、为 0 时没有读数据
Errmsg	WORD	错误信息 0：串口工作正常 1：当前串口无效(注:已被其它程序占用) 2：串口参数格式错误(9600,1,N,8) 3：接收数据奇偶校验错误 4：串口设备忙 5：串口没打开 6：没有收到数据
Length	WORD	被读出的数据长度

一个完整的自由口编程例子如下



7.3.2 CAN 通讯

T9 系列 PLC 能够最多连接 32 个 CAN 总线设备，支持标准帧和扩展帧，能够自由向 CAN 总线中的任何一个 CAN 设备发送数据、接收数据（0 至 8 个字节均可）。

使用时，先用 CAN_Open 指令打开 PLC 的串口：

如果需要向 CAN 总线网络中的某个 CAN 设备发送数据，就用 CAN_Write 指令，并在 CAN_Write 指令中指定本机的 ID，则 CAN 总线中具有这个 ID 号的 CAN 设备就会收到数据；

如果需要接收 CAN 总线网络中某个设备的数据，就用 CAN_Read 指令，在 CAN_Read 指令中指定做为发送方的 CAN 设备的 ID，当 CAN 总线中具有这个 ID 号的 CAN 设备发送数据时，PLC 会收到数据；

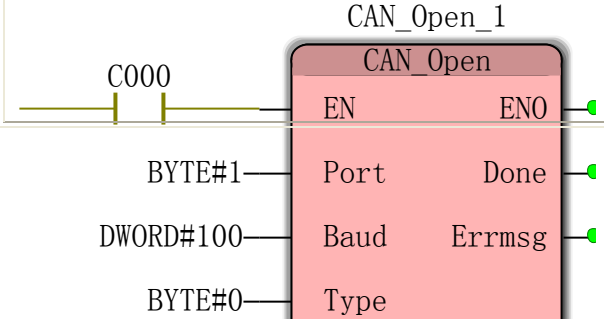
CAN_Open 指令必须是打开的，否则，CAN_Read 和 CAN_Write 指令将无法发送、接收数据。

7.3.2.1 打开 CAN 口 CAN_Open

功能

Can_Open 指令用于打开 PLC 的 CAN 口。

用法

IL 编程语言	LD、 FBD 编程语言
LD C000 ST CAN_Open_1.EN LD BYTE#1 ST CAN_Open_1.Port	

```
LD DWORD#100
ST CAN_Open_1.Baud
LD BYTE #0
ST CAN_Open_1.Type
CAL CAN_Open_1
```

ST 编程语言

```
CAN_Open_1 (EN := C000, Port := BYTE#1, Baud:= DWORD#100, Type := BYTE #0) ;
```

注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量或使用常量

CAN_Open 指令处理的数据类型

输入	数据类型	描述
EN	BOOL	高电平有效，打开 CAN 口
Port	BYTE	PLC 的 CAN 口，如果 PLC 只有 1 个 CAN 口，则这里为 1
Baud	DWORD	波特率，DWORD#1 代表 100kbps，其它波特率如 200kps，400kbps …
Type	BYTE	0——标准帧，1——扩展帧
输出	数据类型	描述
ENO	BOOL	使能输出
Done	BOOL	0: 功能块不能被执行；1: 功能块可以被执行
Errmsg	WORD	错误编号： 1: 波特率错误 2: 端口错误 3: 包类型错误 4: 波特率不支持

7.3.2.2 写数据 CAN_Write

功能

Can_Write 指令用于向 PLC 的 CAN 口写数据。

用法

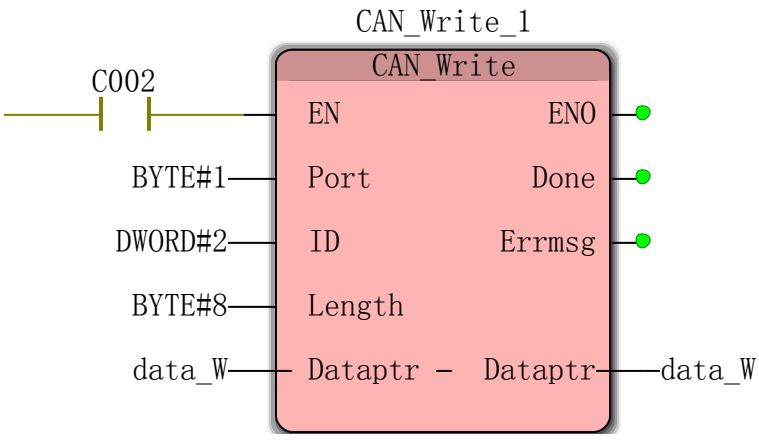
IL 编程语言

```
LD C002
ST CAN_Write_1.EN
LD BYTE#1
ST CAN_Write_1.Port
LD DWORD#2
ST CAN_Write_1.ID
LD BYTE#8
ST CAN_Write_1.Length
LD data_W
ST CAN_Write_1.Dataptr
CAL CAN_Write_1
```

ST 编程语言

```
CAN_Write_1 (EN := C002, Port := BYTE#1, ID := DWORD#2, Length := BYTE#8, Dataptr :=
```

LD、 FBD 编程语言



data_W) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量或使用常量	

CAN_Write 指令处理的数据类型

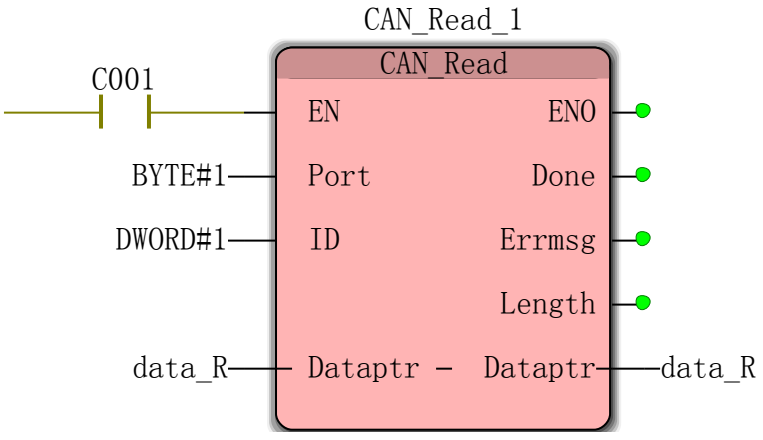
输入	数据类型	描述
EN	BOOL	上升沿有效，每来一个上升沿就向 PLC 的 CAN 口写一次数据
Port	BYTE	PLC 的 CAN 口，如果 PLC 只有 1 个 CAN 口，则这里为 1
ID	DWORD	帧 ID，CAN_Write 指令的 ID 必须和接收方的帧 ID 一致，如果为 0 则表示向所有节点发送数据
Length	BYTE	发送的数据长度，单位——字节
Dataptr	ANY	一般为字节型数组，用于存放将要写到 CAN 口的数据
输出	数据类型	描述
ENO	BOOL	使能输出
Done	BOOL	0: 功能块不能被执行；1: 功能块可以被执行
Errmsg	WORD	错误编号： 1: 端口错误 2: 端口未打开 3: 长度错误 4: 功能块数量超限

7.3.2.3 读数据 CAN_Read

功能

Can_Read 指令用于读 PLC 的 CAN 口数据。

用法

IL 编程语言	LD、 FBD 编程语言
LD C001 ST CAN_Read_1.EN LD BYTE#1 ST CAN_Read _1.Port LD DWORD#1 ST CAN_Read _1.ID LD data_R ST CAN_Read _1.Dataptr CAL CAN_Read _1	
ST 编程语言	
CAN_Read_1 (EN := C001, Port := BYTE#1, ID := DWORD#1, Dataptr := data_R) ;	
注：IL、ST 语言编程时需要在当前 POU 的变量工作单中插入变量或使用常量	

CAN_Read 指令处理的数据类型

输入	数据类型	描述
EN	BOOL	高电平有效，发送方每发一次数据就将其读取到缓冲区中——Dataptr 引脚
Port	BYTE	PLC 的 CAN 口，如果 PLC 只有 1 个 CAN 口，则这里为 1

ID	DWORD	帧 ID，CAN_Read 指令的 ID 必须和发送方的帧 ID 一致
Dataptr	ANY	一般为字节型数组，用于存放从 CAN 口读上来的数据
输出	数据类型	描述
ENO	BOOL	使能输出
Done	BOOL	0: 功能块不能被执行；1: 功能块可以被执行
Length	BYTE	读取到的数据的长度，单位——字节
Errmsg	WORD	错误编号： 1: 端口错误 2: 端口未打开 4: 功能块数量超限

7.3.3 Modbus 通讯 TCMODBUS

TCMODBUS 指令封装了标准的 Modbus RTU 通讯协议，用于 T9 系列 PLC 与 STC、SDP 等 I/O 模块通讯，也可以与其它支持 Modbus RTU 协议的第三方设备通讯，T9 系列 PLC 为主站，I/O 模块或第三方设备为从站，对 Modbus RTU 通讯的要求如下：

- 连接 STC、SDP 等 I/O 模块的 Modbus 通讯口，或第三方设备的支持 Modbus RTU 通讯协议的 RS232、RS485 口
- 如果是通过 PLC 的 RS232 口通讯，则要与第三方设备的 RS232 口直连，即 2 接 2，3 接 3，5 接 5
- 如过是通过 PLC 的 RS485 口通讯，则 A 接 A，B 接 B
- 设置 PLC 与第三方设备的波特率、数据位、校验位、停止位等信息，使其保持一致

使用 TCMODBUS 指令通讯编程时，首先是打开 PLC 的串口并设置通讯格式，然后是发送读写命令，TCMODBUS 通讯包括建立连接指令 TCMODBUSRUN 和读写指令 TCMODBUS 共 2 个指令。

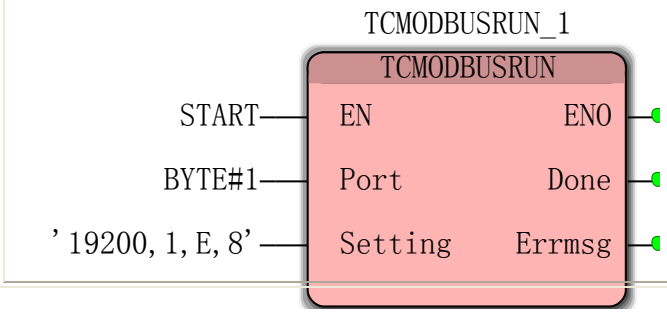
通过 485 口，最多支持 32 个从站的编程。

7.3.3.1 建立连接 TCMODBUSRUN

功能

TCMODBUSRUN 指令用于打开 PLC 的一个串口并设置串口的通讯格式，在通过串口读写过程中，TCMODBUSRUN 指令需要一致有效。

用法

IL 编程语言	LD、 FBD 编程语言
LD START ST TCMODBUSRUN_1.EN LD BYTE#1 ST TCMODBUSRUN_1.Port LD '19200,1,E,8' ST TCMODBUSRUN_1.Setting CALTCMODBUSRUN_1	

ST 编程语言	
TCMODBUSRUN_1(EN:= START, Port:=BYTE#1, Setting:='19200,1,E,8');	
注：各引脚也可使用变量，也可使用常量	

TCMODBUSRUN 指令处理的数据类型

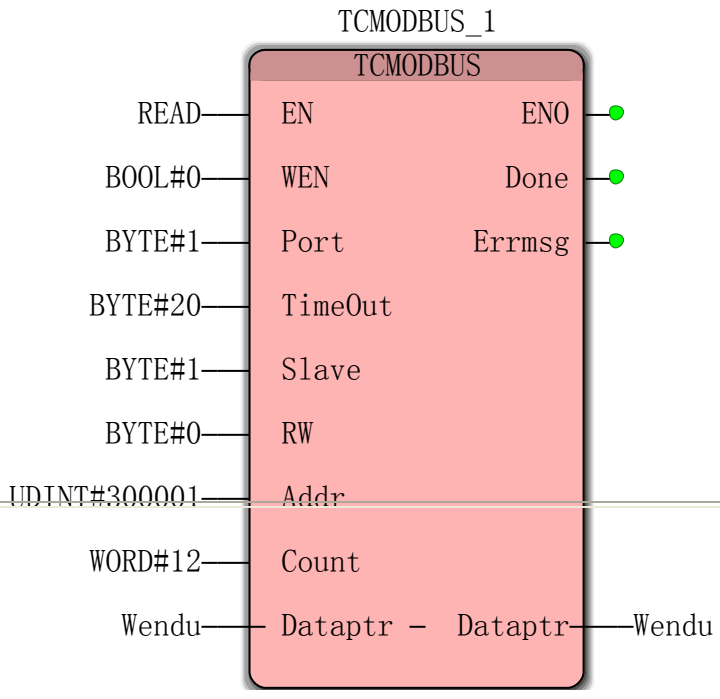
输入引脚	数据类型	描述
EN	BOOL	为 TRUE 时，打开并独占此串口并建立 Modbus 连接；为 FALSE 时，关闭此串口
Port	BYTE	要打开的串口号，可选值：1、2、3，分别对应 COM1、COM2、COM3 口
Setting	STRING	通讯参数，要与从站的通讯格式一致，如'19200,1,E,8'是指 19200 BPS，偶校验、8 位数据位、1 位停止位
输出引脚	数据类型	描述
ENO	BOOL	输出使能
DONE	BOOL	为 1 时表示通讯初始化完成，为 0 时表示未完成
Errmsg	WORD	错误信息，0 表示无错，1 表示出错

7.3.3.2 读写 TCMODBUS

功能

MODBUSRUN 指令用于向已打开的 PLC 串口中读或写数据，在读写过程中，TCMODBUS 指令需要一致有效。

用法

IL 编程语言	LD、 FBD 编程语言
LD READ ST TCMODBUS_1.EN LD BOOL#0 ST TCMODBUS_1.WEN LD BYTE#1 ST TCMODBUS_1.Port LD BYTE#20 ST TCMODBUS_1.TimeOut LD BYTE#1 ST TCMODBUS_1.Slave CALTCMODBUS_1	
ST 编程语言	

TCMODBUS_1(EN:= READ, WEN:=BOOL#0,Port:=BYTE#1, TimeOut:=BYTE#20, Slave:=BYTE#1);	
注：各引脚也可使用变量，也可使用常量	

TCMODBUS 指令处理的数据类型

输入引脚	数据类型	描述
EN	BOOL	使能端，为 TRUE 时有效
WEN	BOOL	写使能，0 为读，1 为写
Port	BYTE	由 TCMODBUSRUN 打开的串口号
TimeOut	BYTE	超时时间：本条报文接收超时时间
Slave	BYTE	从站地址：可选择的范围 1~247
RW	BYTE	读写操作：0 为读，1 为写 注：开关量输出和保持寄存器支持读和写功能，开关量输入和模拟量输入只支持读功能
Addr	UDINT	读写从站的数据地址： 00001 至 0xxxx－开关量输出 10001 至 1xxxx－开关量输入 30001 至 3xxxx－模拟量输入 40001 至 4xxxx－保持寄存器
Count	WORD	通讯的数据个数——BOOL 量的个数或字的个数 注意： Modbus 主站每次可读/写的最大数据量为 120 个字，对于 BOOL 量的读写，推荐值为“8”个，对于字的读写，推荐值为“1”个
DType		只能为 0
Dataptr	ANY	数据指针： 1. 如果是读指令，读回的数据放到这个数据区中 2. 如果是写指令，要写出的数据放到这个数据区中
输出引脚	数据类型	描述
ENO	BOOL	输出使能
DONE	BOOL	读写功能完成位：当指令被正确执行后(PLC 发出报文后从站返回报文并 CRC 校验正确后)置 1，在下一扫描周期会被自动置零，可以用此位去驱动数据处理或其它函数功能块。
Errmsg	WORD	错误代码：只有在 Done 位为 1 时，错误代码才有效 0＝无错误 1＝TCMODBUS 功能块数量太多 2＝未用 3＝接收超时（从站无响应） 4＝请求参数错误（slave address,Modbus address,count,RW） 5＝Modbus/自由口未使能 6＝Modbus 正在忙于其它请求 7＝响应错误（响应不是请求的操作） 8＝响应 CRC 校验和错误

		101＝ 从站不支持请求的功能 102＝ 从站不支持数据地址 103＝ 从站不支持此种数据类型 104＝ 从站设备故障 105＝ 从站接受了信息，但是响应被延迟 106＝ 从站忙，拒绝了该信息 107＝ 从站拒绝了信息 108＝ 从站存储器奇偶错误

7.3.4 PLC 与 PLC 之间通讯 TCNET

TCNET 指令是北京腾控科技基于 TCP/IP 协议开发的在 T9 系列 PLC 之间互联实现通讯的一种网络通讯协议，TCNET 具有以下特点：

- 采用 UDP 方式
- 可以一次完成对所有变量的读写：包括读中间变量，写中间变量，读写中间变量，读输入，写输出，读写输入输出
- 本地的调用者为服务器端，被调用者为客户端
- 每台 PLC 都可以作为服务器和客户端
- 以太网远程模块只能作为服务器
- 提供非常简单的编程接口
- 只需指定服务器的 IP 地址，读写地址，读写长度和自身保持的地址和长度即可
- 用户只需熟悉一个功能块，不需要了解协议细节

通过一根以太网线，将两个 PLC 连接起来，就可以使用 TCNET 指令了，或者通过交换机，将多个 PLC 连接起来，也可使用 TCNET 指令。

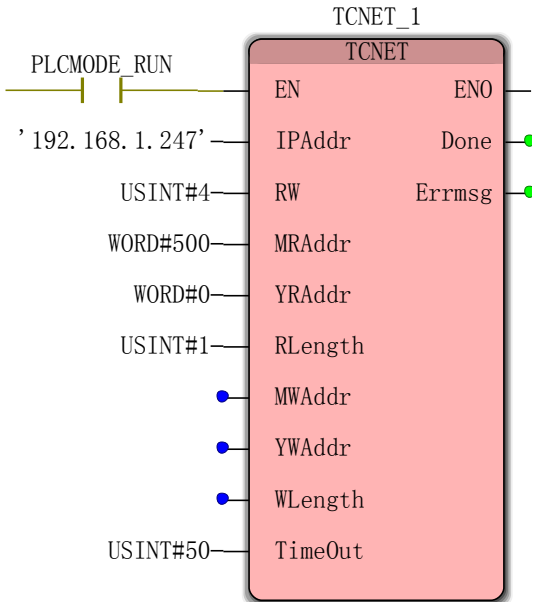
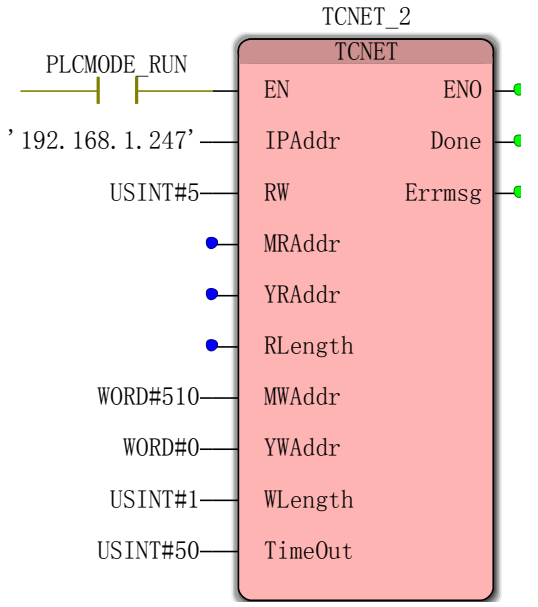
通过以太网口，最多支持 16 个从站的编程。

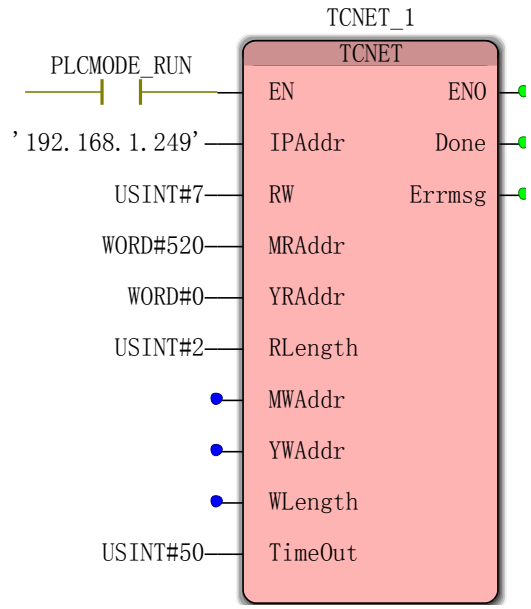
TCNET 指令处理的数据类型

输入引脚	数据类型	描述
EN	BOOL	通讯使能端，为 TRUE 时，启动通讯，为 FALSE 时，断开通讯
IPAddr	STRING	要读写的 PLC 的 IP 地址，如‘192.168.1.249’
RW	USINT	通讯使用的功能码 1：读中间变量区 2：写中间变量区 3：同时读写中间变量区 4：读开关量输入 5：写开关量输出 6：同时读写开关量输入输出 7：读模拟量输入 8：写模拟量输出 9：同时读写模拟量输入输出
MRAddr	WORD	本机读取数据后存放在本机的存放地址（起始地址）
YRAddr	WORD	远程被读数据的起始地址
RLength	USINT	要读取的数据长度
MWAddr	WORD	在本机上要写的数据的存放地址（起始地址）
YWAddr	WORD	在远程 PLC 上的数据起始地址（要写到这里）
WLength	USINT	要写的数据长度

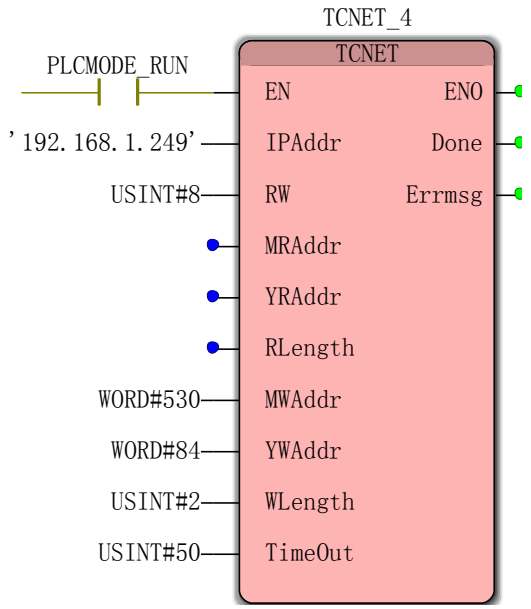
输出引脚	数据类型	描述
ENO	BOOL	输出使能
DONE	BOOL	为 1 时表示已建立通讯，为 0 时表示未通讯
Errmsg	WORD	通讯状态码 1：通讯超时 2：IP 地址格式错误 4：读写地址出错 8：功能码不存在 16：读写长度出错 32：读错误 64：写错误 128：读写错误 256：资源溢出（超出最大支持的功能快数量） 512：网络连接建立失败 1024：端口获取失败 2048：目标不可达

用法

读从站 PLC 的数字量输入	写从站 PLC 的数字量输出
 引脚 RW 的值为 4，表示读取从站 PLC 的数字量输入（以字节为最小单位）；引脚 MRAddr 的值为 500，表示读到的数据存放在主站 PLC 的 IB500；引脚 YRAddr 的值为 0，表示读取从站 PLC 的输入映射区 IB0，1 表示 IB1，以此类推；引脚 RLength 的值为 1，表示读一个字节的数字量输入。	 引脚 RW 的值为 5，表示写从站 PLC 的数字量输出（以字节为最小单位）；引脚 MWAddr 的值为 510，表示要写的数据存放在主站 PLC 的 QB510；引脚 YWAddr 的值为 0，表示写从站 PLC 的输出映射区 QB0，1 表示 QB1，以此类推；引脚 RLength 的值为 1，表示写一个字节的数字量输入。
读从站 PLC 的模拟量输入	写从站 PLC 的模拟量输出

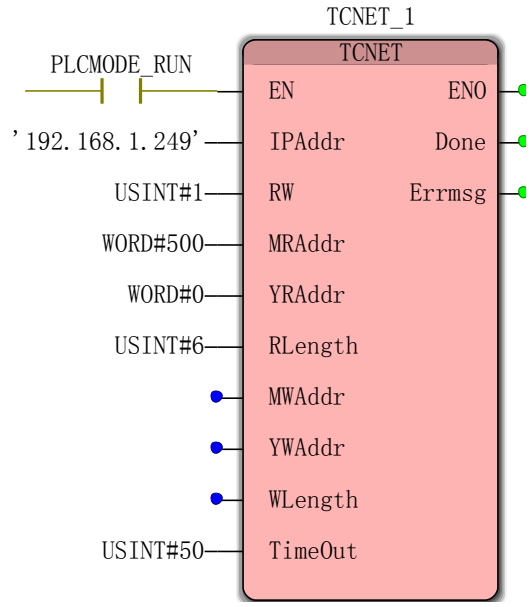


引脚 RW 的值为 7，表示读取从站 PLC 的模拟量输入；引脚 MRAddr 的值为 520，表示读到的数据存放在主站 PLC 的 IW520；引脚 YRAddr 的值为 0，表示读取从站 PLC 的第 1 路 AI，2 表示第 2 路 AI，以此类推；引脚 RLength 的值为 2（以字节为最小单位），表示读一个 AI(一个 AI 为一个字)。



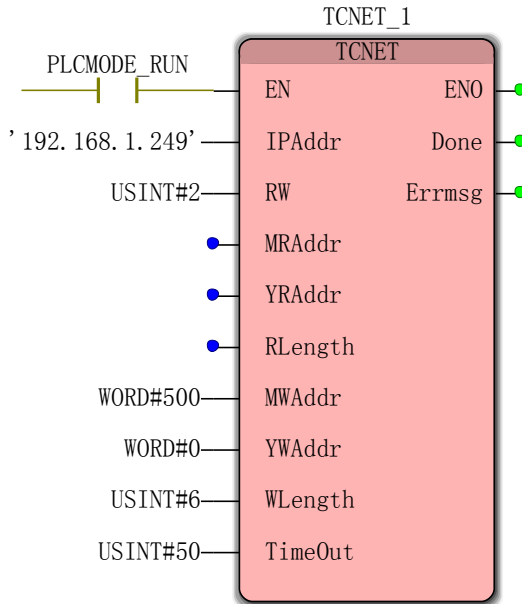
引脚 RW 的值为 8，表示写从站 PLC 的模拟量输出（以字为最小单位）；引脚 MWAddr 的值为 530，表示要写的数据存放在主站 PLC 的 QW530；引脚 YWAddr 的值为 84，表示写从站 PLC 的第 1 路 AO（QW64）；86 表示第 2 路 AO（QW66），以此类推；引脚 WLength 的值为 2，表示写一个 AO。

读从站 PLC 的中间变量区

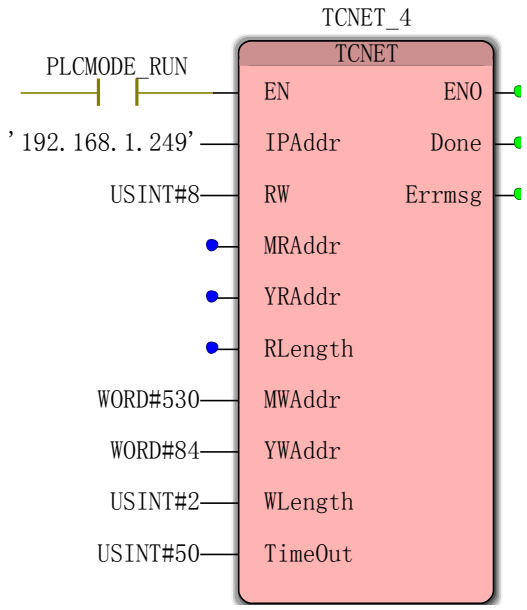


引脚 RW 的值为 1，表示读取从站 PLC 的中间变量区；引脚 MRAddr 的值为 500，表示读到的数据存放在主站 PLC 的 MB3.500；引脚 YRAddr 的值为 0，表示读取从站 PLC 的 MB3.0，1 表示 MB3.1，以此类推；引脚 WLength 的值为 6（以字节为最小单位），表示读中间变量区的 6 个字节。

写从站 PLC 的中间变量区



引脚 RW 的值为 2，表示写从站 PLC 的中间变量区；引脚 MWAddr 的值为 500，表示要写的数据存放在主站 PLC 的 MB3.500；引脚 YWAddr 的值为 0，表示把数据写到从站 PLC 的 MW3.0；1 表示 MB3.1，以此类推；引脚 WLength 的值为 6，表示写中间变量区的 6 个字节。



8. PLC 地址与 Modbus 地址的对应

8.1 中间变量映射区

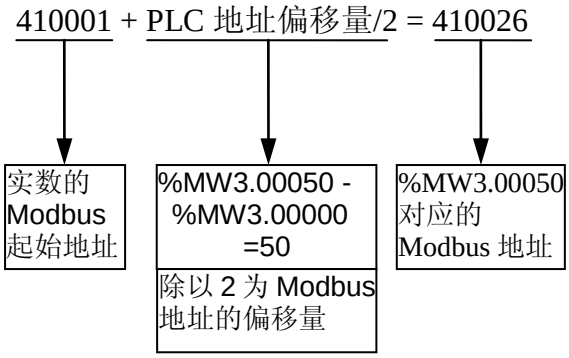
在中间变量区，可以定义 BOOL、BYTE、WORD 和 DWORD 型变量，也可以定义为 SINT、INT、DINT、REAL 等类型。

	PLC 地址				对应
序号	位	字节	字	双字	Modbus 地址
0	%MX3.00000.0	%MB3.00000	%MW3.00000		410001
1	%MX3.00001.0	%MB3.00001			
2	%MX3.00002.0	%MB3.00002	%MW3.00002	%MD3.00002	410002
3	%MX3.00003.0	%MB3.00003			
4	%MX3.00004.0	%MB3.00004	%MW3.00004		410003
5	%MX3.00005.0	%MB3.00005			
6	%MX3.00006.0	%MB3.00006	%MW3.00006	%MD3.00006	410004
7	%MX3.00007.0	%MB3.00007			
8	%MX3.00008.0	%MB3.00008	%MW3.00008		410005
9	%MX3.00009.0	%MB3.00009			
10	%MX3.00010.0	%MB3.00010	%MW3.00010	%MD3.00010	410006
11	%MX3.00011.0	%MB3.00011			
12	%MX3.00012.0	%MB3.00012	%MW3.00012		410007
13	%MX3.00013.0	%MB3.00013			
14	%MX3.00014.0	%MB3.00014	%MW3.00014	%MD3.00014	410008
15	%MX3.00015.0	%MB3.00015			
16	%MX3.00016.0	%MB3.00016	%MW3.00016		410009
17	%MX3.00017.0	%MB3.00017			
…	…	…	…	…	…

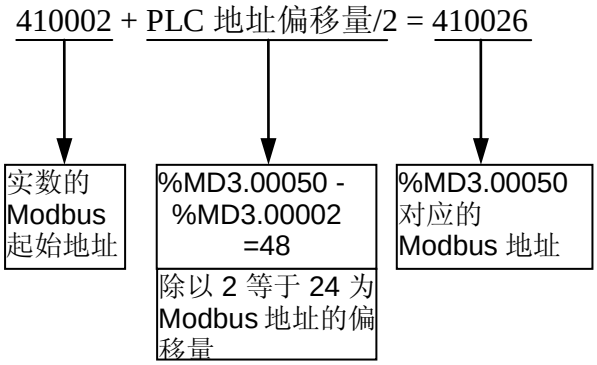
在中间变量区，基本的 PLC 地址是按字节型变量定义的，即一个 BYTE 型变量占据一个 PLC 地址，一个 BOOL 型变量也占据一个 PLC 地址，一个 WORD 型变量占据两个字节但其地址是低字节的 PLC 地址，一个 DWORD 型变量占据 4 个字节但其地址是最低字节的 PLC 地址。如果要定义一个 SINT、INT、DINT 型变量，则它们对应的地址分别是字节型（%MB）、字（%MW）和双字（%MD），如果要定一个 REAL 型变量，则对应的地址是双字（%MD）。

Modbus 地址的确定：

- 字变量，假设其地址为%MW3.00050，则对应的 Modbus 地址为



- 实数变量，必须从%MB3.00002 开始使用，假设其地址指定为%MD3.00050，则对应的 Modbus 地址为

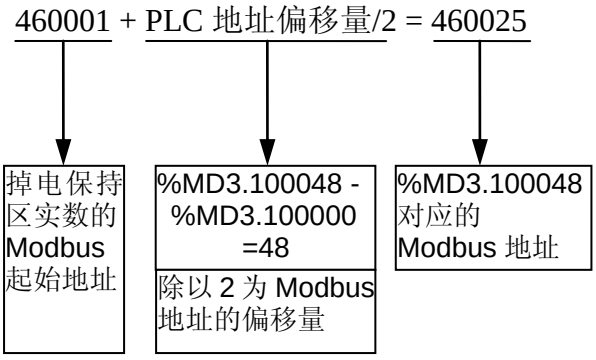


在 MULTIPROG5.35 中，中间变量区有 1000 个字节的掉电保持区（但不能冷起 PLC，否则数据会丢失），可以定义 BOOL、BYTE、WORD 和 DWORD 型变量，也可以定义为 SINT、INT、DINT、REAL 等类型。

	PLC 地址				对应
序号	位	字节	字	双字	Modbus 地址
0	%MX3.100000	%MB3.100000	%MW3.100000	%MD3.100000	460001
1	%MX3.100001	%MB3.100001			
2	%MX3.100002	%MB3.100002	%MW3.100002		460002
3	%MX3.100003	%MB3.100003			
4	%MX3.100004	%MB3.100004	%MW3.100004	%MD3.100004	460003
5	%MX3.100005	%MB3.100005			
6	%MX3.100006	%MB3.100006	%MW3.100006		460004
7	%MX3.100007	%MB3.100007			
8	%MX3.100008	%MB3.100008	%MB3.100008	%MD3.100008	460005
9	%MX3.100009	%MB3.100009			

10	%MX3.100010	%MB3.100010	%MB3.100010		460006
11	%MX3.100011	%MB3.100011			
12	%MX3.100012	%MB3.100012	%MB3.100012		460007
13	%MX3.100013	%MB3.100013			
14	%MX3.100014	%MB3.100014	%MB3.100014		460008
15	%MX3.100015	%MB3.100015			
16	%MX3.100016	%MB3.100016	%MB3.100016		460009
17	%MX3.100017	%MB3.100017			
…	…	…	…	…	…

1000 个字节的掉电保持区地址计算：



8.2 数字量输入输出映射区

数字量输出			数字量输入		
序号	PLC 地址(位)	对应 Modbus 地址	序号	PLC 地址(位)	对应 Modbus 地址
1	%QX0.0	000001	1	%IX0.0	100001
2	%QX0.1	000002	2	%IX0.1	100002
3	%QX0.2	000003	3	%IX0.2	100003
4	%QX0.3	000004	4	%IX0.3	100004
…	…	…	…	…	…

8.3 模拟量输入输出映射区

模拟量输出			模拟量输入		
序号	PLC 地址(字)	对应 Modbus 地址	序号	PLC 地址(字)	对应 Modbus 地址
1	%QW64	400043	1	%IW64	300001
2	%QW66	400044	2	%IW66	300002
…	…	…	…	…	…

9. 编程注意事项

- 使用梯形图编程语言时，尽量避免用同一个指令对同一个变量进行多处操作。
- 梯形图编程，程序的扫描顺序是从上到下，从左到右。
- 编程时，不要在编辑区拷贝粘贴指令（功能块），否则编译出错，可以从右侧编辑向导中拖入编辑区，或在编辑区直接键入指令（功能块）的名称。
- 以下指令暂时不支持：
  GET_CHAR
  GET_ERROR
  GET_ERROR_CATALOG
  GET_SYM
  CLR_OUT
  COLD_RESTART
  CONTINUE
  HOT_RESTART
  WRITE_RETAIN
  WARM_RESTART
  读写 PDD 变量的指令，如 WR_BOOL_BY_SYM
- FOR 循环：步数不能太长，如 15000，否则可能出错误，一个程序中，也不能出现多个较大的 FOR 循环（如 10000 步），否则也可能出错。
- 数组 ARRAY 的最大长度是 32767

10. ASCII 码表

ASCII 的全称是美国信息交换标准码，是 American Standard Code for Information Interchange 的缩写，ASCII 码定义从 0 到 127 的一百二十八个数字所代表的英文字母或数字或符号，所有使用 ASCII 的计算机之间可以互相读取同一份文件而不会有不一样的结果与意义。由于只使用 7 个位 (BIT) 就可以表示从 0 到 127 的数字，大部分的计算机都使用 8 个位来存取字符集 (CHARACTER SET)，见表 1 和表 2。从 128 到 255 之间的数字可以用来代表另一组一百二十八个符号, 称为 EXTENDED ASCII，见表 3。表 1 中的十进制的 0~31 和 127 为控制字符，表 2 中的 32~126 为可打印字符；表 3 中 128~255 为扩展 ASCII 码。

表 1 控制字符（表中 Hx 表示十六进制，Dec 表示十进制）

Dec	Hx	字符	描述	Dec	Hx	字符	描述
0	0	NULL	Null/空字符	17	11	DC1	Device Control 1/设备控制 1
1	1	SOX	Start Of Heading/标题开始	18	12	DC2	Device Control 1/设备控制 2
2	2	STX	Start Of Text/正文开始	19	13	DC3	Device Control 1/设备控制 3
3	3	ETX	End Of Text/正文结束	20	14	DC4	Device Control 1/设备控制 4
4	4	EOT	End Of Transmission/传输结束	21	15	NAK	Negtive Acknowledge/拒绝接收
5	5	ENQ	Enquiry/请求	22	16	SYN	Synchronous Idle/同步空闲
6	6	ACK	Acknowledge/收到通知	23	17	ETB	End of Trans. Block/传输块结束

7	7	BEL	Bell/响铃	24	18	CAN	Cancel/取消
8	8	BS	Backspace/退格	25	19	EM	End of Medium/介质中断
9	9	HT	Horizontal Tab/水平制表符	26	1A	SUB	Substitute/替补
10	0A	LF	NL line feed, new line/换行	27	1B	ESC	Escape/溢出
11	0B	VT	Vertical Tab/垂直制表符	28	1C	FS	File Separator/文件分割符
12	0C	FF	NP form feed, new page/换页	29	1D	GS	Group Separator/分组符
13	0D	CR	Carriage Return/回车	30	1E	RS	Record Separator/记录分离符
14	0E	SO	Shift Out/停止切换	31	1F	US	Unit Separator/单元分隔符
15	0F	SL	Shift In/启用切换	127	7F	DEL	Delete/删除
16	10	DLE	Data Link Escape/数据链路转义				

表 2 可打印字符（表中 Hx 表示十六进制，Dec 表示十进制）

Dec	Hx	字符	Dec	Hx	字符	Dec	Hx	字符	Dec	Hx	字符
32	20	Space	56	38	8	80	50	P	104	68	h
33	21	!	57	39	9	81	51	Q	105	69	i
34	22	“	58	3A	:	82	52	R	106	6A	j
35	23	#	59	3B	;	83	53	S	107	6B	k
36	24	$	60	3C	<	84	54	T	108	6C	l
37	25	%	61	3D	=	85	55	U	109	6D	m
38	26	&	62	3E	>	86	56	V	110	6E	n
39	27	‘	63	3F	?	87	57	W	111	6F	o
40	28	(	64	40	@	88	58	X	112	70	p
41	29	)	65	41	A	89	59	Y	113	71	q
42	2A	*	66	42	B	90	5A	Z	114	72	r
43	2B	+	67	43	C	91	5B	[	115	73	s
44	2C	,	68	44	D	92	5C	\	116	74	t
45	2D	-	69	45	E	93	5D	]	117	75	u
46	2E	.	70	46	F	94	5E	^	118	76	v
47	2F	/	71	47	G	95	5F	_	119	77	w
48	30	0	72	48	H	96	60	`	120	78	x
49	31	1	73	49	I	97	61	a	121	79	y
50	32	2	74	4A	J	98	62	b	122	7A	z
51	33	3	75	4B	K	99	63	c	123	7B	{
52	34	4	76	4C	L	100	64	d	124	7C	|
53	35	5	77	4D	M	101	65	e	125	7D	}
54	36	6	78	4E	N	102	66	f	126	7E	~
55	37	7	79	4F	O	103	67	g			

表 3 扩展 ASCII 码（表中 Hx 表示十六进制，Dec 表示十进制）

Dec	Hx	字符	Dec	Hx	字符	Dec	Hx	字符	Dec	Hx	字符
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ

欢迎您宝贵的意见！

我们想竭诚为您服务，我们在努力完善自己。我们希望您阅读本书、使用产品时，如发现错误、论述不详或使用问题找不到相应解释时，请您致电我们或者填写意见表邮寄给我们，我们真诚期待您的宝贵意见。

来电请致： 010-59790086-233
北京腾控科技有限公司 市场部
北京市海淀区紫竹园路广源闸 5 号广源大厦 320
邮编： 100086
传真： 010-68726710-214

文字错误：

论述不详：

使用问题找不到相应解释：

其他问题：

（可复印此表传真我们）

如果方便，请留下您的联系方式

联系人：
职务：
公司：
地址：
电话：
E-mail：

地址：北京市海淀区紫竹院路广源闸5号广源大厦3层

邮编：100086

电话：+86-10 59790086

传真：+86-10 68703551