可编程控制器
PROGRAMMABLE LOGIC CONTROLLER

软件手册
EC100/200系列PLC

欧瑞传动
EURA DRIVES

目　　录

第一章　产品概述

本章主要介绍了 EC100/EC200 系列小型一体化 PLC 的上位编程软件 EuraProg，并详细罗列了使用 EuraProg 时的安全注意事项及其特点、创新点等。

1.1 EuraProg 简介

EuraProg 是 EC100/EC200 系列小型一体化 PLC 的编程软件，编程环境符合 IEC 61131-3 标准，是一套功能强大、使用方便、高效的开发系统。

IEC 61131-3 标准，是 IEC 工作组在合理地吸收、借鉴世界范围的各 PLC 厂家的技术、编程语言、方言等的基础之上，形成的一套新的国际编程语言标准 。IEC 61131-3 国际标准随着 PLC 技术、编程语言等的不断进步也在不断补充和完善。它极大提高了工业控制系统的编程软件质量，同时提高了软件开发效率。自发布以来得到所有顶尖 PLC 厂家的认可，各厂家的编程软件也在尽量向 IEC 标准靠拢。

EuraProg 采用了符合 IEC 61131-3 标准的方案，这就使得其简便、易学，使用起来将会得心应手，支持两种编程模式：LD（梯形图）和 IL（指令表），便于用户选用；EuraProg 还提供程序在线编辑、调试、监控，以及 CPU 内部数据的监视、修改功能；EuraProg 同时支持子程序、中断程序的编辑，提供集成库程序功能，以及用户定义的库程序。

- 主要特点
  - 符合 IEC 61131-3 标准
  - 支持 IL（指令表）和 LD（梯形图）两种标准语言
  - 丰富的指令集，内置 IEC 61131-3 定义的标准功能、功能块以及一些特殊的应用指令
  - 支持结构化的编程方式，用户的程序被组织成“工程”
  - 支持中断服务程序，支持用户自定义功能块（子程序）
  - 允许在程序中使用变量名称，便于工程的实施、维护
  - 灵活的硬件配置方式，最大限度地允许用户自定义各种硬件参数
  - 完善的联机功能，包括下载、上载、在线监控、强制、读写实时时钟等
  - 定义了完善的快捷键、右键菜单，方便用户的使用
  - 对用户的错误操作尽可能地予以屏蔽、提示，体贴用户的操作

1.2 使用规范

• **接线**　在给 CPU 进行供电接线时，一定要特别小心分清是哪一种供电方式，如果把 220VAC 接到 24VDC 供电的 CPU 上，或者不小心接到 24VDC 传感器输出电源上，都会造成 CPU 的损坏。

• **电源**　每个 CPU 都有一个 24VDC 电源，它为本机输入点和扩展模块输入点及扩展模块继电器线圈提供 24VDC。当电源要求超出了 CPU 模块的电源定额时，需要增加一个外部 24VDC 电源来提供给扩展模块。

• **通讯**　缺省情况下，EC100/EC200 系列 CPU 的通讯口处于 Modbus 从站模式，地址为 1，通讯速率为 9.6K。要更改通讯口的地址或通讯速率，必须在 EruaProg 的通讯设置对话

框中设置，然后将设置下载到 CPU 中，新的设置才能起作用。

• **冗余**　设计大中型可编程序控制系统时不要耗尽它的硬件和软件资源，对于所设计新的系统，硬件上至少要保留 15%左右的冗余，在软件编制时，同样要估计用户软件对计算机资源的需要与用量，合理的配置可编程序控制器的冗余。

• **通讯连接**　当使用 RS232 接口与 PLC 通讯时，必须先将两端设备断电后方能插拔通讯电缆，确认电缆完全拔下或解除无误后再给设备上电。

• **安装**　安装使用 EC100/EC200 系列 PLC 时，须按照硬件手册使用规范安装。

第二章　　软件的安装与操作

本章对 EuraProg 的安装、卸载、运行环境进行了详细的描述。另外，介绍了 PLC 与计算机连接、界面操作等内容。

2.1　配置要求

EuraProg 软件对计算机及编程设备的最低配置要求如下：

2.1.1　硬件配置

- CPU：主频 133MHz 或更高
- 硬盘：30M 以上空间
- 内存：256M 以上
- 键盘、鼠标、串行通信口（COM 口）或者 USB 口（需另外使用 USB/RS232 转换器）
- 256 色 VGA 或更高

2.1.2　软件配置

操作系统：Windows 2000、Windows XP、Windows 7

2.2　EuraProg 的安装与卸载

2.2.1　安装

注意：在安装过程的任何一步单击**【取消】**，都可以终止安装。

1） 运行安装盘中的 EuraProg Vx.x setup.exe（x.x 代表版本号，比如 2.0），弹出安装向导的首页。

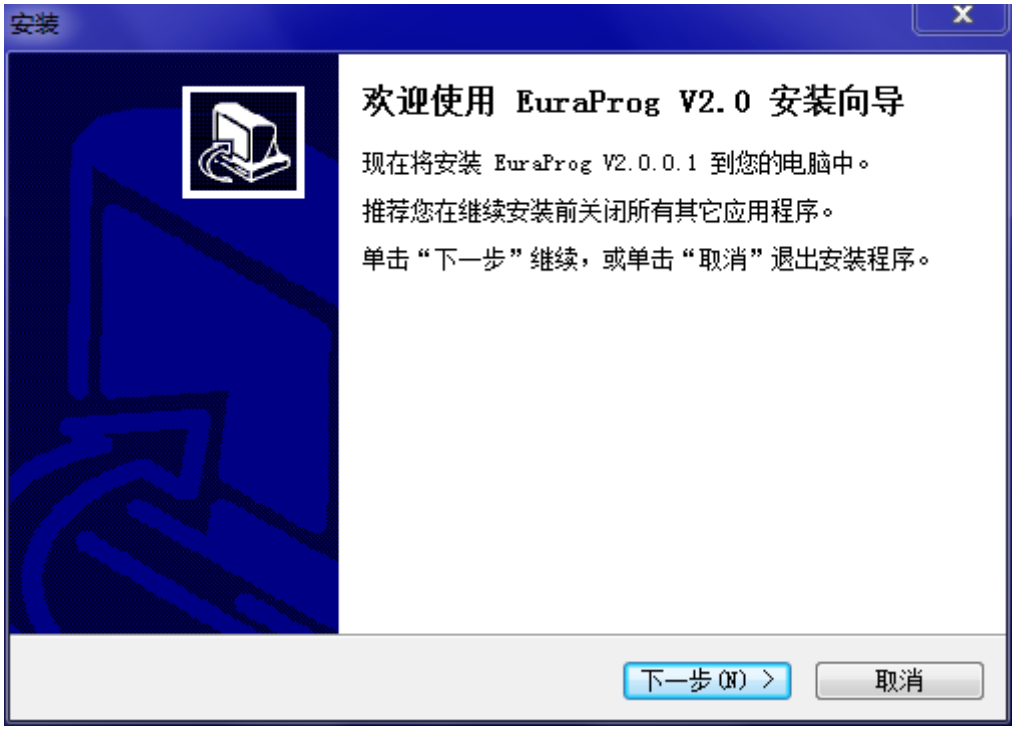


图 2-1　安装向导

2) 单击【下一步】，将进入安装、修复或卸载选项界面，选择“安装”。

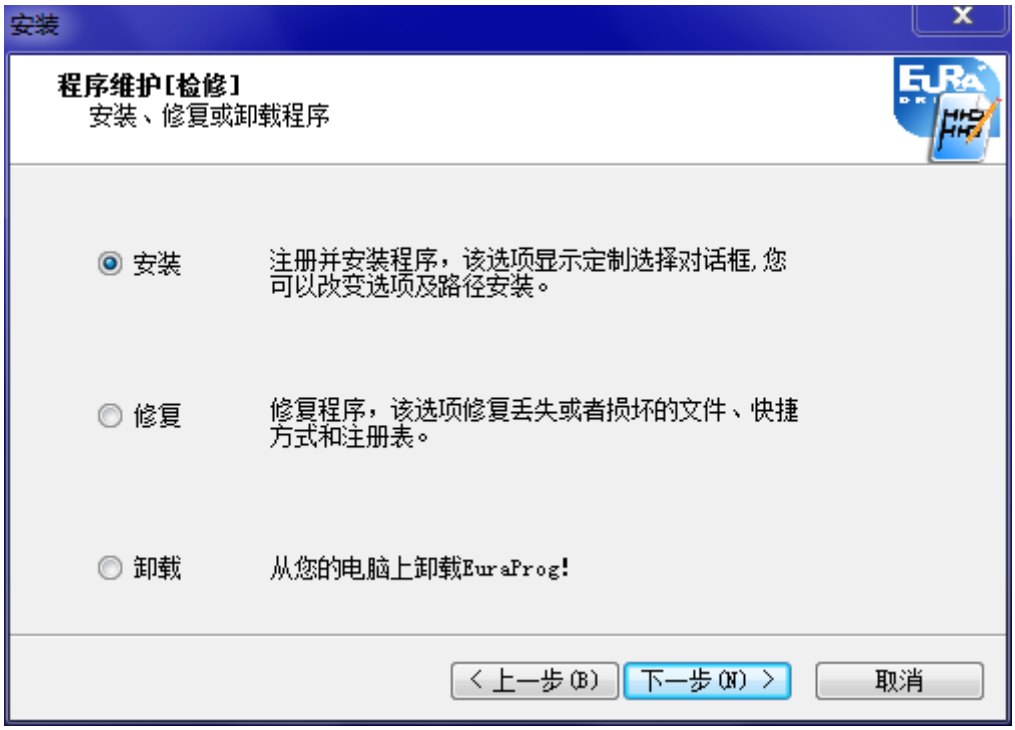


图 2-2　安装向导

3) 单击【下一步】，将确认 EuraProg 的安装协议，选择“我同意此协议”。

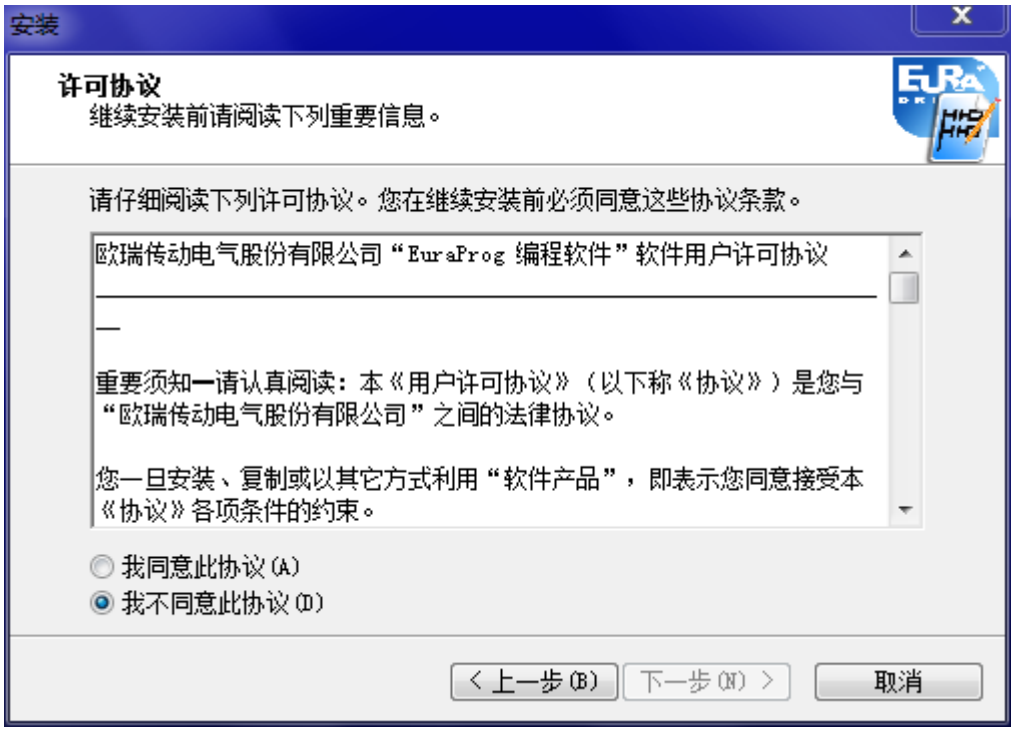


图 2-3　安装协议

4) 单击【下一步】，将确认 EuraProg 的安装路径。用户可使用默认路径，也可以对其进行修改。

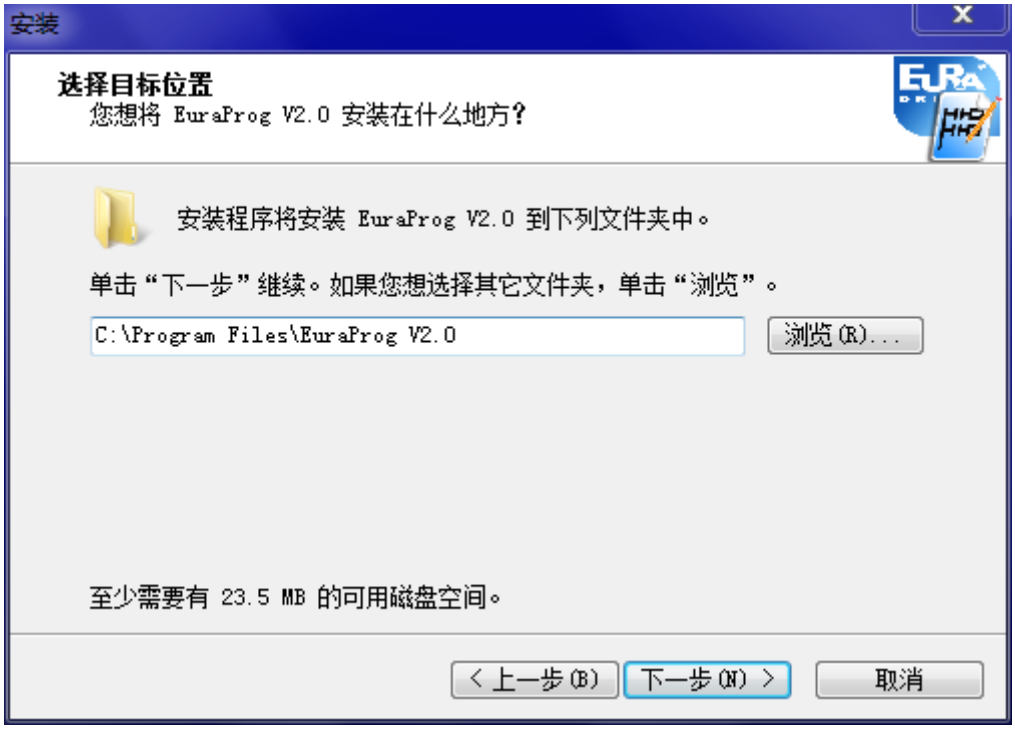


图 2-4　安装路径

5) 单击【下一步】，确认是否在桌面、运行栏中生成快捷方式。

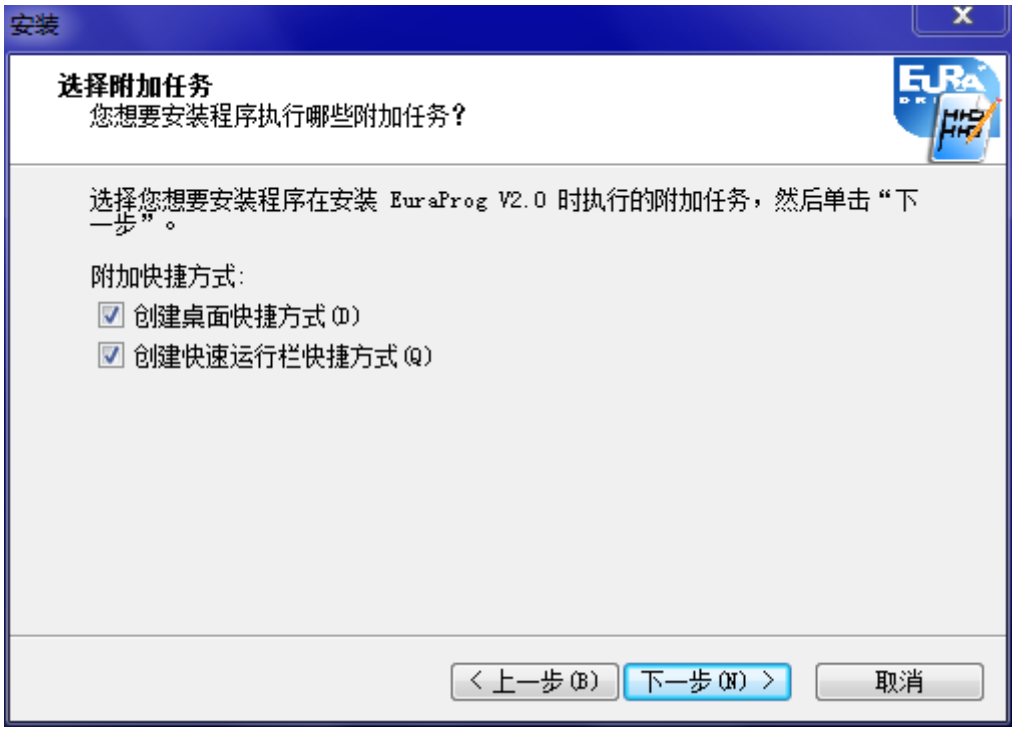


图 2-5　附加任务

6）用户选择完是否在桌面、运行栏中创建快捷方式后，单击【下一步】，将提示确认准备开始安装。

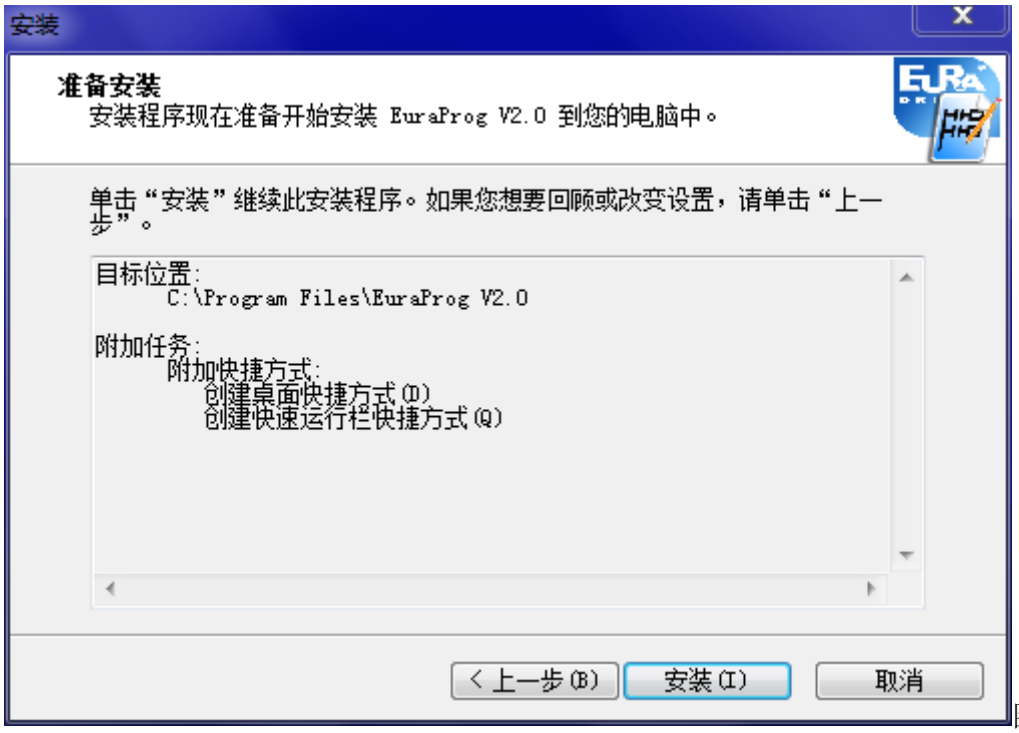


图 2-6　准备安装

7）单击【安装】，将正式开始安装过程，安装结束后的提示，如图 2-7。

图 2-7　安装完成

2.2.2　修复

打开安装程序，进入安装向导界面，选择【修复】，单击【下一步】，其余步骤同“安装”过程。

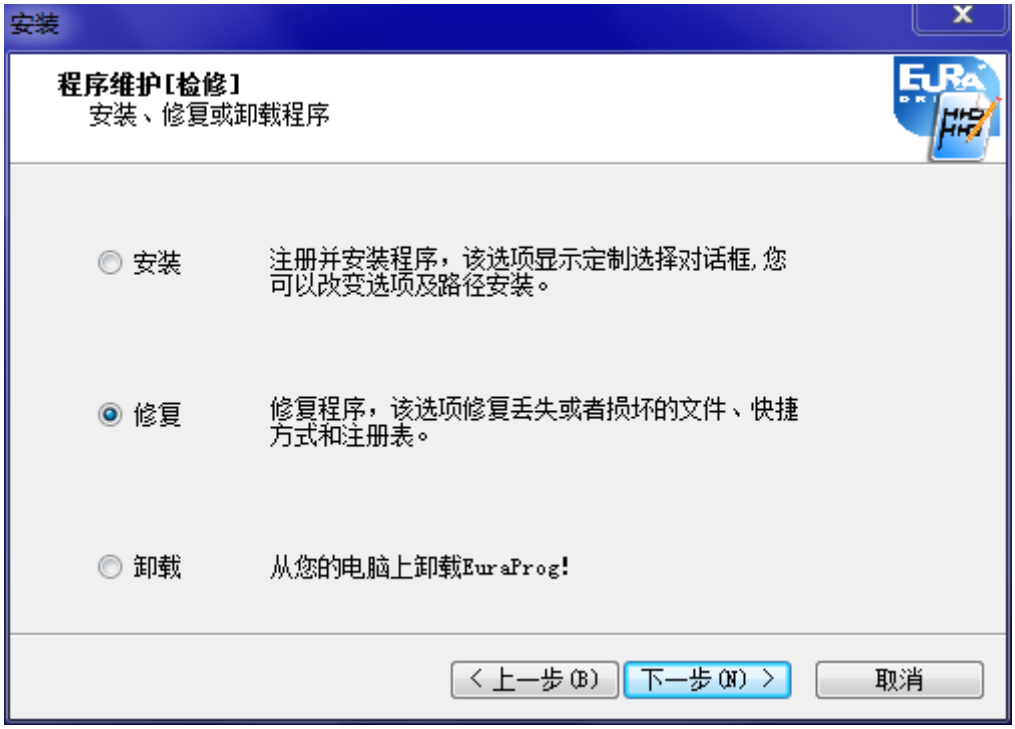


图 2-8　修复向导

2.2.3 卸载

执行卸载前，请先退出 EuraProg 程序。卸载 EuraProg 软件共有三种方法：

· 执行【开始】→【程序】中 EuraProg 快捷方式组中的【卸载 EuraProg】命令，将开始卸载过程。

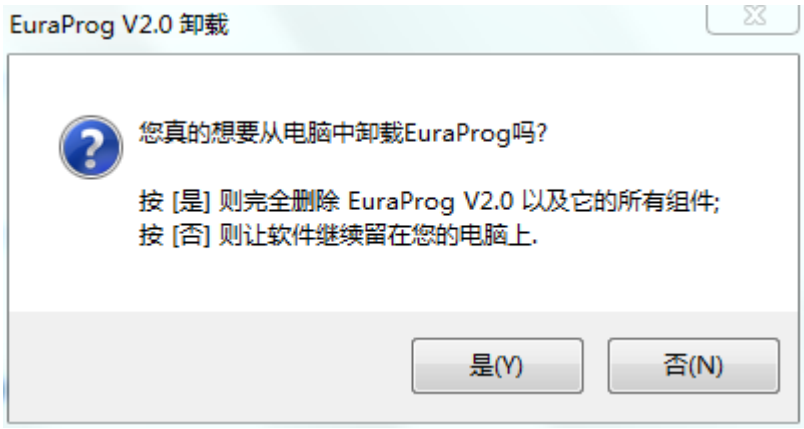


图 2-9　卸载确认

· 打开安装程序，进入安装向导，选择【卸载】选项，点击下一步进行卸载。

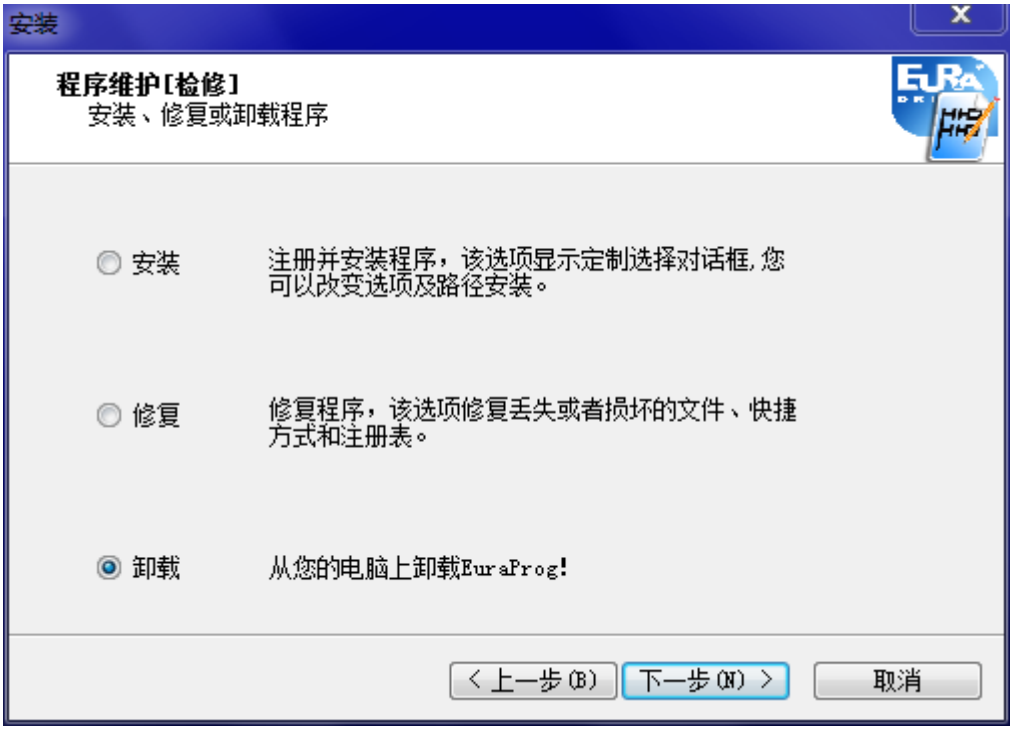


图 2-10　卸载向导

· 单击【开始】→【设置】→【控制面板】，进入控制面板，执行其中的【添加或删除程序】命令，继而在弹出的“添加/删除程序”对话框中选择“EuraProg Vx.x”（x.x 代表版本号，如 2.0），然后单击【更改/删除】按钮。

2.2.4 启动和退出

1）启动 EuraProg

用户可以通过下面两种简单的方法启动 EuraProg：

· 执行【开始】→【程序】中 EuraProg 快捷方式组中的【EuraProg】命令。

· 若您在安装时选中了“在桌面上创建快捷方式”，则双击桌面上的  图标即可。

2）退出 EuraProg

启动 EuraProg 后，有三种方式可以退出软件：

· 执行【文件】→【退出】菜单命令

· 使用 Alt+F4 快捷键

· 单击 EuraProg 主窗口右上角的  图标。

2.3 EuraProg 界面详细介绍

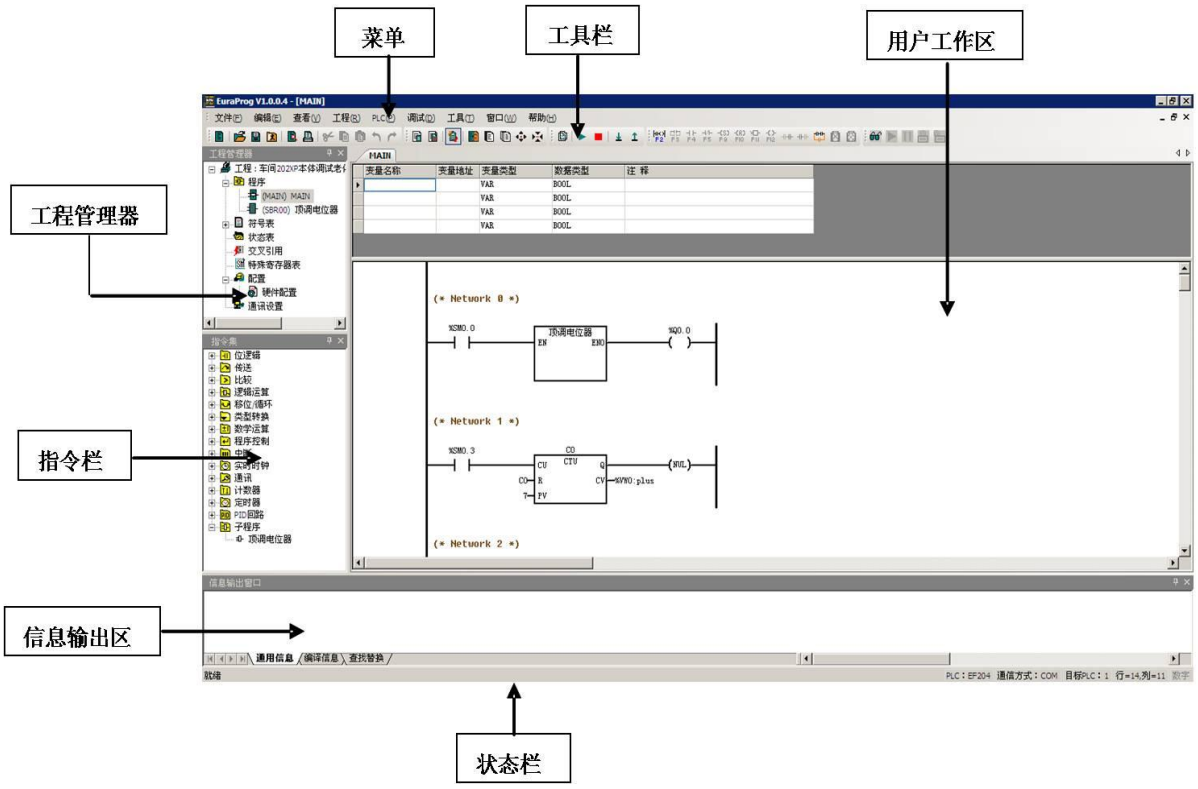


图 2-11　软件界面

- 菜　　单：菜单中包含了 EuraProg 软件所有的操作命令。
- 工　具　栏：工具栏中包含了用户使用频度较高的一些操作命令。
- 状　态　栏：状态条提供了软件当前的状态信息和操作命令的提示信息。
- 工程管理器：工程管理器是界面中的主要窗口之一，以树状列表的形式直观地显示出了当前工程的所有组成部分，包括程序、硬件配置、状态表、全局变量表等。用户可以在

此窗口中对当前打开的工程进行管理、操作、维护。工程管理器的各个树节点均支持右键，用右键单击某个节点将会弹出相应的右键菜单。

- 用户工作区：这是用户使用的主要区域，用户可以在此打开硬件配置窗口、全局变量表、编辑器等窗口，完成配置硬件、声明全局变量、编辑程序等任务。

  上图显示的编辑器包括区域上部的变量定义表格和区域下部的程序编辑器。编辑器又分为 LD 编辑器、IL 编辑器。
- 指　令　集：以树状列出了 EC100/EC200 系列 PLC 支持的所有指令、功能、功能块和用户自己编写的子程序。指令集又分为 LD 指令集、IL 指令集。
- 信息输出区：用于显示 EuraProg 软件的各种提示信息。其中“编译信息”窗口显示了用户最近一次的编译信息，而“通用信息”窗口显示了最近一些操作的提示信息。

2.4 EuraProg 中应用程序的组织

2.4.1 工程的组织结构

在 EuraProg 中应用程序被组织成“工程（Project）”，工程中包含了用户程序、硬件配置等应用程序的所有信息。在本手册中，“用户程序”和“用户工程”这两个词的含义是相同的。

下表详细描述了工程的组织结构。表中注明“可选”的项表示此项并非工程的必要元素，用户在工程中可以忽略它们。

表 2-1 工程的组织结构

符号表	初始化数据表（可选）	用户可在此为 V 区中 BYTE、WORD、DWORD、INT、DINT、REAL 类型的数据指定初始值。初始化数据表是工程中的可选部分，用户也可以不在表中输入任何内容。 在冷启动时 CPU 进入主循环之前，初始化数据表被处理一次。
	全局变量表（可选）	用户在此声明工程中需要的全局变量。
程序	主程序	主程序是在 CPU 内运行的主循环任务，在 CPU 的每个扫描周期内都会被执行一次。 在工程中有且仅有 1 个主程序。

	中断服务程序（可选）	响应某个中断事件而被执行的程序称为中断服务程序。 中断服务程序不需要在主程序中调用。用户只需要通过 ATCH 指令将其与某个预定义的中断事件连接起来，那么当该中断事件发生时，CPU 就会自动调用该中断服务程序进行处理。 一个工程中最多允许有 100 个中断服务程序。
	子程序（可选）	子程序必须被主程序或者中断服务程序调用后才能被执行。 子程序是可重用的代码段，用户可以将常用的控制功能编写成一个个子程序。使用子程序可以更好地组织应用程序的结构，方便调试和维护，有效地缩短程序代码的长度。 一个工程中最多允许有 100 个子程序。
配置	硬件配置	用户在此对工程中用到的 PLC 模块及其参数进行配置。CPU 将在冷启动时处理一次硬件配置，然后再执行其它任务。

2.4.2 工程的存储目录

在建立工程时 EuraProg 软件将会提供一个默认的工程存放路径，用户可以修改此路径，点击“确定”就会生成一个空的工程文件（扩展名为.eup）并存放于此路径下。

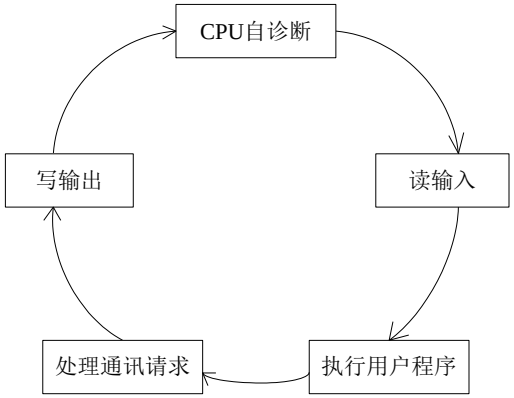
例如，用户若确定在 c：\ test 目录下建立一个名为 “项目 1” 的工程，则工程文件的路径是 c：\test\项目 1.eup。

2.5 程序的执行

CPU 循环连续执行一系列任务的过程称为扫描。扫描是 CPU 的主要工作，通常也被称为主循环。在一个扫描周期内 CPU 将执行如下任务：

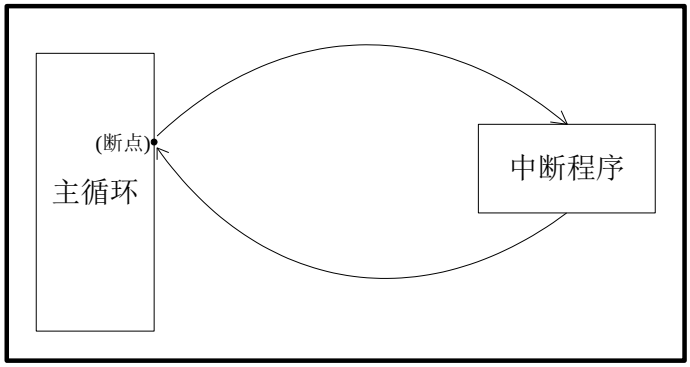
- CPU 自诊断
- 读输入：读取物理输入通道的信号数据并将其写至输入映像区

- 执行用户程序：直接执行用户工程中的主程序
- 处理通讯请求
- 写输出：将输出映像区中的数据写至物理输出通道



另外，需要注意的是，CPU 在扫描周期中执行的任务取决于它的工作状态。CPU 具有两种工作状态：运行（RUN）状态、停止（STOP）状态。这两种状态的主要差别就是：在 RUN 状态下 CPU 将执行用户程序，而在 STOP 状态下 CPU 不执行用户程序。

EC100/EC200 系列 PLC 也支持中断，用户编写的中断服务程序将在指定连接（ATCH）的中断事件发生时执行。中断事件可能发生在扫描过程中的任一时刻，这时候 CPU 将会先暂时中断主循环，转去执行中断服务程序，中断服务程序执行完成后，再从刚才的断点处重新进入主循环。如下图。



2.6 与计算机的连接

CPU 模块上集成了 RS232 或者 RS485 串行通讯接口，用户可以使用这些通讯口与其它设备进行通讯。此处介绍如何建立计算机与 CPU 之间的通讯。

① 使用适当的编程电缆连接好计算机的串行通讯口和 CPU 的串行通讯口；然后启动 EuraProg，新建一个工程或者打开一个已经存在的工程。

若使用的是 RS232 通讯口，由于 RS232 的电气标准不支持带电插拔，因此用户在插拔电缆之前必须先断掉至少一方（CPU 或者计算机）的电，否则容易损坏通讯口。

② 设置计算机串行通讯口的通讯参数。这些参数必须与 CPU 通讯口的串行通讯参数完全一致才能正常地进行通讯。步骤如下：

在工程管理器中，双击【通讯设置】节点，或者在【通讯设置】节点上单击鼠标右键，

执行弹出菜单中的【打开…】命令，或者执行【工具】→【通讯设置…】菜单命令，均可进入“通讯设置”对话框，如下图所示。

图 2-12　通讯设置

在【**目标 PLC**】中选择将要连接的目标 PLC 站号；在【**COM 口**】中选择当前计算机所用的 COM 口；依据目标 PLC 的串行通讯参数来设置所选 COM 口的通讯参数（包括【**波特率**】、【**奇偶校验**】、【**数据位**】、【**停止位**】）。

设置好上述参数后，单击【**确定**】按钮即可。

若用户不知道所用 CPU 的通讯参数，那么该如何进行设置呢？此时有两种方法：

• 方法一

先选择将要使用的计算机【**COM 口**】，然后单击【**自动检测**】按钮，EuraProg 将自动检测当前所连 CPU 的通讯参数，检测所需要的时间可能是数秒钟到数分钟，若检测成功，那么 EuraProg 将依据检测到的 CPU 通讯参数自动对当前所用计算机 COM 口的参数进行设置。

• 方法二

先将所连 CPU 断电，将其运行开关拨至“STOP”位置，然后对 CPU 重新上电，此时 CPU 将使用默认的串行通讯参数：站号为 1，波特率 9600，无校验，8 位数据位，1 位停止位。用户依据此参数设置好计算机 COM 口的通讯参数即可进行各种通讯操作。注意：在 CPU 的通讯参数被修改之前不要改变运行开关的位置!

③　设置好计算机串口的通讯参数，即可对所连 PLC 进行编程、调试。

第三章　EuraProg 软件使用基础

这一章我们介绍软件的使用及其功能，包括如何建立一个工程、配置、编写、编译、调试及软件的其他功能，使用户了解开发一个工程的常见步骤。

3.1　新建工程

执行【文件】→【新建工程..】菜单命令；

单击工具栏上的图标会出现“新建工程”对话框（如图 3-1 所示）。通过该窗口可设置工程路径、PLC 类型、编程语言和工程描述。

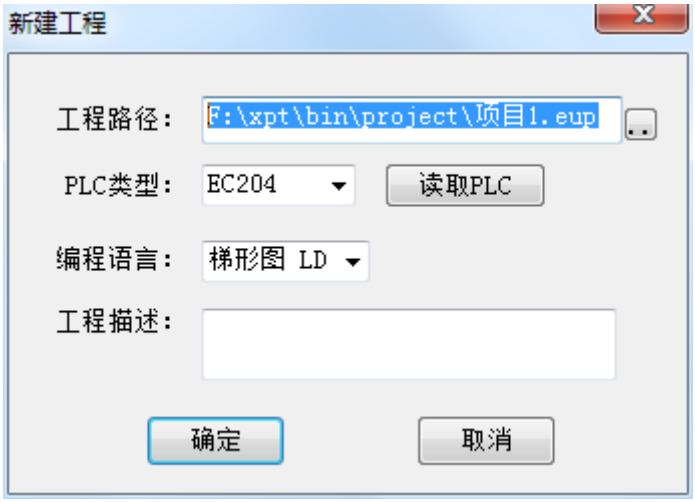


图 3-1　新建工程

工程路径：　程序会生成默认工程名称和存放路径，用户也可点击工程路径右侧的按钮，在弹出的工程路径设置窗口（如图 3-2 所示）中修改工程名称及存放路径；

PLC 类型：　用户可手动设置 PLC 类型也可通过点击【读取 PLC】按钮实现自动配置。自动配置功能需要保持 PC 与 PLC 通讯正常，程序将通过串口读取当前 PLC 的信息；

编程语言：　用户可根据需求选择“梯形图 LD”或“指令表 IL”编程模式；

工程描述：　用户可在此添加工程描述，限 30 字。

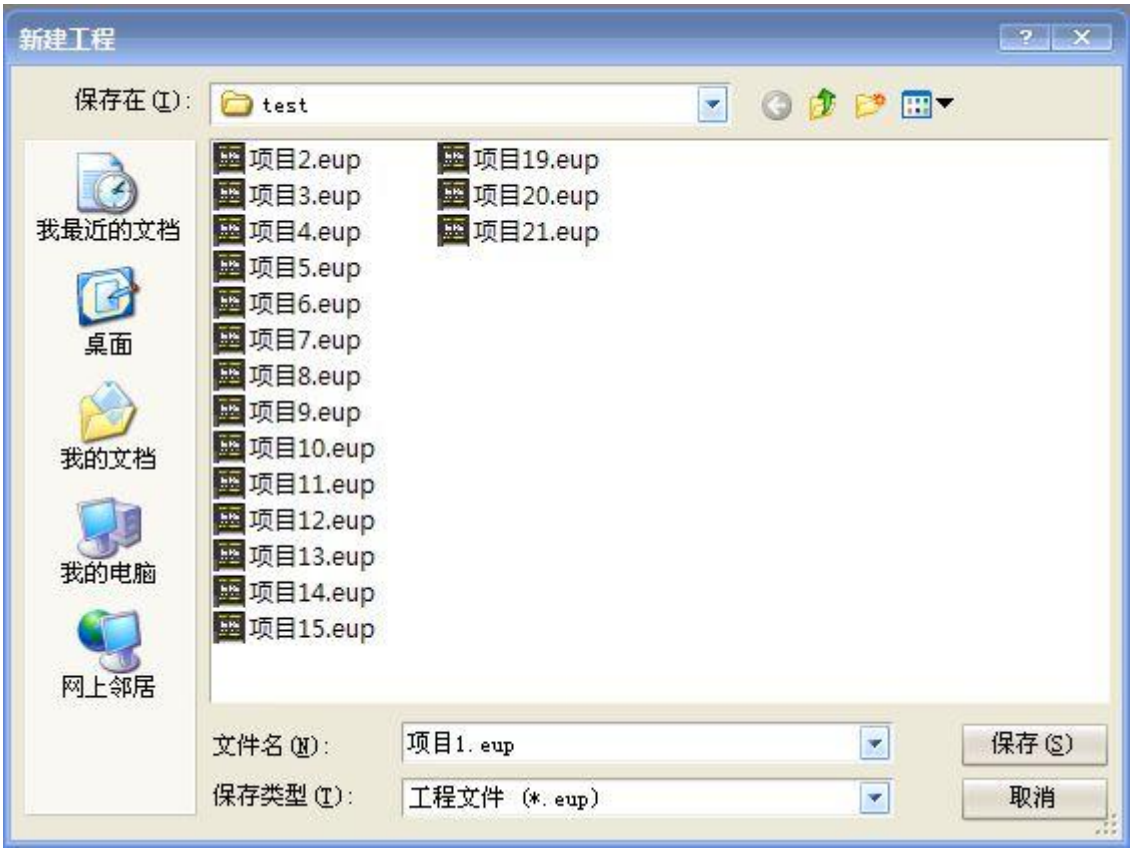


图 3-2　工程路径设置

3.2 PLC 配置

3.2.1　硬件配置窗口

- 用户可以在任何时候修改硬件配置参数，但由于硬件配置是一个工程中必需的信息，因此建议用户在工程中首先完成硬件配置。
- 用户连接好计算机和 PLC 之后就可以使用 EuraProg 来检查或者修改该 PLC 的通讯参数了。

  用户可以使用如下任一方法进入硬件配置窗口：

  • 双击工程管理器中**【配置】**组下的**【硬件配置】**节点；

  • 在**【硬件配置】**节点上单击鼠标右键，然后执行弹出的**【打开...】**菜单命令。

  硬件配置窗口分为三部分，CPU 模块、扩展模块和模块参数配置窗口。上面两部分显示的是模块列表信息，下面一部分显示的是模块参数配置信息。模块参数配置窗口显示的是 PLC 模块列表中选中模块的所有参数配置。鼠标左键单击选中模块列表中的模块，就可以切换到相应模块的参数配置窗口。如图 3-3 所示：

图 3-3　硬件配置窗口

在模块参数配置窗口中用户可以对通讯参数进行设置，如图 3-4 所示：

图 3-4　串行口通讯参数配置页面

3.2.2　添加和删除模块

完成模块的添加和删除操作，用户可以手动配置，也可使用自动配置功能。

♦　自动配置方法：

在【**硬件配置**】节点上单击鼠标右键，然后执行弹出的【**自动配置**】菜单命令实现。自动配置功能需要保持 PC 与 PLC 通讯正常，程序将通过串口读取当前 PLC 的模块信息，实现

自动配置。若读取失败，将无法实现自动配置功能。

- 手动配置方法：

1）添加模块

在硬件配置窗口的 CPU 模块列表中列出了工程中使用的 CPU 模块类型，左键单击 CPU 模块一列，即可在下拉列表中选择需要的 CPU 模块；扩展模块列表中列出了工程中使用的扩展模块的类型，左键单击扩展模块一列，即可在下拉列表中选择需要的扩展模块。硬件配置中模块列表的排列情况代表了实际模块的链接情况。

各个模块之间不允许有空行存在。若有空行，则软件不允许在其后添加模块，并且在保存、编译时会提示错误。

各种 CPU 所允许的最大 I/O 点数均有明确的规定，若已添加的模块的点数超出了 CPU 的限制，则软件将不允许继续添加模块，并且在保存、编译时会提示错误。

2）删除模块

用户可以使用如下任一方法删除一个模块：

- 鼠标单击模块列表中将要删除的模块，然后敲 **Delete** 键即可删除。
- 鼠标右键单击模块列表中将要删除的模块，执行弹出菜单的**【删除模块】**命令即可。

3.2.3 配置模块参数

EC100/EC200 的每种模块都提供了多种参数和选项设置以适应不同的具体应用，包括模块的 I/O 地址、模拟量模块各通道的信号类型等等，在 EuraProg 软件中允许用户自定义所有的这些参数和选项。

在模块列表中，鼠标单击任何一个模块并选中它，在下边就会出现该模块的参数配置窗口，用户可以在这个模块参数配置窗口中修改该模块的参数。

在模块参数配置窗口的右侧有两个公用的按钮：**【缺省值】**、**【取消】**。

【缺省值】：单击此按钮，将会取消当前页面用户的输入，软件为本模块自动分配参数。

【取　消】：单击此按钮，将会取消当前页面用户的输入，恢复本模块原有的配置。

注意：各模块在同一内存区域（I、Q、AI 或者 AQ）内的地址不允许重叠！

- **CPU 参数配置**

① **【I/O 设置】**页面

在此页面中可以完成 CPU 本体上 I/O 点的相关配置。如下图。

图 3-5 **【I/O 设置】**参数配置页面

输入区：用于配置 CPU 本体上 DI 点的参数。

- **起始地址**：DI 部分在 I 区中所占用地址的起始字节，固定为 0。
- **输入滤波**：定义 DI 信号采样时所需的滤波时间，单位 ms。每 4 个点被分为一组进行定义。

来自于现场的 DI 信号至少要保持所设定的滤波时间，然后 CPU 才会认为该输入有效并更新相应的 DI 映像区，否则 CPU 对该输入不响应。这样有助于滤除输入噪声，增强系统的抗干扰能力。

滤波时间越短，CPU 对该输入的响应就越快。在实际应用中，用户应根据现场的具体情况来设定滤波时间。所有的 DI 点默认是不滤波的。

输出区：用于配置 CPU 本体集成的 DO 点的参数。

- **起始地址**：DO 部分在 Q 区中所占用地址的起始字节，固定为 0。
- **输出保持**：设置当 CPU 处于 STOP 状态时各 DO 点的输出状态。

若某 DO 点对应的复选框被选中，则当 CPU 处于 STOP 时，该点输出为 1。

若某 DO 点对应的复选框没有被选中，则当 CPU 处于 STOP 时，该点输出为 0。

此项功能对于 CPU 停机后所需要的安全连锁非常有意义。

② **【通讯设置】**页面

用于配置 CPU 本体所带串行通讯口（Port0、Port1）的参数。如下图。

图 3-6 **【通讯设置】**参数配置页面

✓ Port0

• **站号** ：为 Port0 指定一个唯一的站号。该站号作为 Modbus RTU 从站时的站号。

• **波特率** ：设置串行通讯的波特率，其值有：2400、4800、9600、19200、38400、57600、115200。

• **奇偶校验**：设置串行通讯的奇偶校验方式，其值有：无校验、奇校验、偶校验。

• **数据位** ：设置串行通讯的数据位，其值有：8。

• **停止位** ：设置串行通讯的停止位，其值有：1。

✓ Port1

Port1 是 RS485 接口。

• **作为 Modbus 主站**：若选中此项，那么 Port1 就作为 Modbus RTU 的主站进行通讯。

• **超 时**：设置 Modbus 主站通讯的超时时间，单位 ms。范围 1-65535。

• **重 试**：当 Modbus 主站收到从站错误的应答后，继续重新尝试通讯的次数。范围 0-10。

其它参数请参见上面关于 Port0 的描述。

当主站发出一个命令后，在下述情况下会产生通讯错误：

1) 在定义的超时时间内没有收到从站的应答，则会产生一个通讯超时错误；

2) 主站收到了从站错误的应答，则会重新尝试通讯，最多重新发送“**重试”**次命令。若最后一次仍没有收到从站正确的应答，则主站继续等待超时时间后产生一个通讯超时错误。

③ **【数据保持】**页面

在此页面中配置数据保持区域。在 CPU 每个扫描周期的最后，数据保持区域中的数据将存储到特殊存储器件中，CPU 掉电再次上电时载入使用。数据保持页面如下图。

注意：“初始化数据表”、“数据保持”功能保持的内存区域要避免重叠。因为这些数据均在上电之后进入主循环之前进行恢复，其次序是：恢复“数据保持”中定义的内存数据、“初始化数据表”中定义的内存区域赋初始值。

图 3-7　【数据保持】参数配置页面

总共可以配置 4 个互相独立的数据保持区域。

• **数据区**：指定被保持的数据区所在的内存区，4 个数据保持区域分别依次为 V 区、M 区、C 区、T 区。

• **起始地址**：指定该保持区域的起始字节地址或者计数器与定时器的编号。

• **长度**：指定该保持区域的长度，单位：字节或者是计数器/定时器的个数。

对于定时器，要求只能设置保持性定时器 TONR。具体数据保持区可设置范围可参阅《附录 C 数据保持》或者通过 EuraProg 编程软件实际操作可知。以编程软件为准。

♦ DI 模块的参数配置

图 3-8　DI 模块参数配置

✓ 地址

• **起始地址**：指定该模块在 I 区中占用地址空间的起始字节地址。
该模块所有通道的地址都由这个“起始地址”决定。

• **长　　度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 DI 通道的数量。

如图 3-8，该模块具有 8 个 DI 通道，模块地址是%IB2，它的通道的地址是 %I2.0–%I2.7。

♦ DO 模块的参数配置

图 3-9　DO 模块参数配置

✓　地址

- **起始地址**：指定该模块在 Q 区中占用地址空间的起始字节地址。
- **长　　度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 DO 通道的数量。

如图 3-9，该模块具有 8 个 DO 通道，模块的起始地址是%QB2，它的通道的地址是 %Q2.0–%Q2.7。

✓　输出保持

设置当 CPU 处于 STOP 状态时该模块各 DO 点的输出状态。

若某 DO 点对应的复选框被选中，则当 CPU 处于 STOP 时，该点输出为 1；若某 DO 点对应的复选框没有被选中，则当 CPU 处于 STOP 时，该点输出为 0。输出点在 CPU 处于 STOP 时的缺省状态是输出为 0。

此项功能对于 CPU 停机后所需要的安全连锁非常有意义。

♦　DIO 模块的参数配置

图 3-10　DIO 模块参数配置

DIO 模块的输入区配置同 DI 模块的参数配置，DIO 模块的输出区配置同 DO 模块的参数配置。

♦　AI 模块的参数配置

图 3-11　AI 模块参数配置

✓ 地址

- **起始地址**：指定该模块在 AI 区中占用地址空间的起始字节地址（也就是第一个通道的地址）。每个 AI 点在 AI 区占用 2 个字节。因此，该地址必须为偶数。
- **长　　度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 AI 通道的数量。

如图 3-11，模块的起始地址被指定为%AIW0，该模块具有 4 个 AI 通道，因此它的 4 个通道的地址依次是%AIW0、%AIW2、%AIW4、%AIW6。

✓ 其它参数

- **信号形式**：各个通道可选择的输入信号有 4-20mA、0-20mA、1-5V、±10V。
  采样值将在 CPU 内自动进行线性转换，关于数据转换格式请参见硬件手册中的“测量范围和测量值表示格式”相关内容。
- **滤波方式**：为各个通道选择软件滤波器。
  对于变化较快的模拟量信号使用滤波器可以让测量值变得比较稳定。

注意：若系统需要对某 AI 信号快速响应，那么就不应该启用该通道的软件滤波器。

软件滤波器的输入采样均采用了滑动方式。软件滤波器有如下选项：

- 无　　--- 不启用软件滤波器。
- 算术平均 --- 对一定数量的信号采样值取算术平均值。
- 中值平均 --- 将一定数量的信号采样值去掉最大、最小值后，对剩下的数取平均值。

♦ AO 模块的参数配置

图 3-12　AO 模块参数配置

✓ 地址

- **起始地址**：指定该模块在 AQ 区中占用地址空间的起始字节地址（也就是第一个通道的地址）。每个 AO 点在 AQ 区占用 2 个字节。因此，地址必须为偶数。
- **长　　度**：该模块占用地址空间的长度。这是一个固定值，取决于模块上 AO 通道的数量。

如图 3-12，模块的起始地址被指定为%AQW4，该模块具有 2 个 AO 通道，它的 2 个通道的地址依次是%AQW4、%AQW6。

✓ 通道设置

- **信号形式**：　各个通道可选择的输出信号有 4-20mA、0-20mA、1-5V、±10V。
  关于各种信号的输出值表示格式的说明请参见硬件手册中“输出范围和输出值表示格式”相关内容。
- **停机保持**：　指定当 CPU 处于 STOP 状态时，该点是否保持原来的输出。若该点对应的复选框被选中，则该点采用停机保持方式。若该点对应的复选框没有被选中，则该点不采用停机保持方式。
- **停机输出值**：若该点设置为停机保持方式，则在此设置停机时该点的输出值。
  此处应当输入经过线性转换之后的数值而非实际的信号值。例如，若用户选择信号形式为 1-5V，停机时希望输出 1V，则应当在此处输入 1000。

♦ AIO 模块的参数配置

图 3-13　AIO 模块参数配置

AIO 模块的输入区配置同 AI 模块的参数配置，AIO 模块的输出区配置同 AO 模块的参数配置。

硬件配置信息必须下载到 CPU 以后才会生效。在每次冷启动（重新上电）时，CPU 会首先检测实际所连模块的信息，CPU 进入 RUN 状态后，将检测到的实际信息与用户工程设置的硬件配置信息进行比较，若有错误 ERROR 灯点亮，否则就进入正常运行状态。

3.2.4　初始化数据表

根据实际需要，用户可以随时填写初始化数据表。在初始化数据表中用户可以为 V 区中的 BYTE、WORD、DWORD、INT、DINT、REAL 类型的数据指定初始值。在 CPU 上电时进入主循环之前，初始化数据表将被处理一次，用户指定的初始值被赋给相应的地址。初始化数据表样式如下图：

初始化数据表

	起始地址	初始值	初始值	初始值	初始值
1	%VB100	B#33	B#34	B#35	
2	%VR200	0.99	5.889	6.03	
3	%VD300	DI#100005	DI#10006	DI#10007	
4	%VW400	66	88	99	
5					
6					

图 3-14　初始化数据表

注意：“初始化数据表”、硬件配置的“数据保持”要避免重叠。因为这些数据均在上电之后进入主循环之前进行恢复、赋值，其次序是：恢复“数据保持”中定义的内存数据、恢复“初始化数据表”中定义的内存区域初始值。

3.2.4.1　进入初始化数据表

有如下两种方式可以进入“初始化数据表”窗口：

- 双击工程管理器中的**【符号表】→【初始化数据表】**;
- 右键单击**【初始化数据表】**，执行打开菜单。

3.2.4.2　输入数据

鼠标左键单击表格单元可使其获得焦点选中状态，鼠标左键双击表格单元可让其进入编

辑状态。若某表格单元失去焦点，则其输入的数据将得到确认。

使用上、下、左、右箭头键可以改变具有焦点的表格单元。

使用 PageUp、PageDown 键可以翻页。

若输入的数据有错误，将会自动变成红色进行提示。错误的数据在保存时会被忽略掉。

3.2.4.3 定义初始化数据

初始化数据表中共有 5 列：1 个**【起始地址】**列和 4 个**【初始值】**列。

用户可以按如下步骤定义初始化数据：

• 在**【起始地址】**列中输入一个直接地址；

• 在**【初始值】**列中输入一个或者多个数值。若输入多个数值，则 EuraProg 将它们隐含地分配给从起始地址开始的连续多个同类型的直接地址变量。

图 3-14 中，第 1 行的含义是为%VB100、%VB101、%VB102 分别赋初始值 B#33、B#34、B#35；第 2 行的含义是为%VR200、%VR204、%VR208 分别赋初始值 0.99、5.889、6.03；第 3 行的含义是为%VD300、%VD304、%VD308 分别赋初始值 DI#10005、DI#10006、DI#10007；第 4 行的含义是为%VW400、%VW402、%VW404 分别赋初始值 66、88、99。

3.2.4.4 编辑初始化数据表

✓ 排序

单击**【起始地址】**列的列头即可让表格按照起始地址中字母的升序或者降序进行排序。

✓ 右键菜单

在表中任意一个单元单击鼠标右键，将会弹出如下菜单：

删除当前行：删除焦点所在的行。

插入一行（上）：在焦点所在行的上一行的位置插入一个新的空行。

插入一行（下）：在焦点所在行的下一行的位置插入一个新的空行。

另外，使用粘贴命令时请注意：不同类型的表格之间不允许粘贴，比如全局变量表与初始化数据表；不同的列之间不允许粘贴。

3.3 编写工程

用户编程可以使用两种语言：IL 或者 LD。用户可以随时执行**【查看】→【IL 编辑器】**或者**【查看】→【LD 编辑器】**菜单命令切换当前程序的语言。注意：任何 LD 程序都可以转

换为 IL 程序，但只有按照一定规则编写的 IL 程序才可以转换成 LD 程序。

1）主程序

当用户建立一个新工程时，软件将会自动创建一个主程序，其名字为“MAIN”。用户在 MAIN 中编写自己需要的主程序。

2）建立子程序

用户可以使用下面方法建立一个新的子程序：

- 工程管理器中执行**【程序】→【新建子程序】**菜单命令；
- 单击工具栏上的  图标。

建立的新子程序默认名字是“SBR_0”。在 SBR_0 中用户可以编写自己需要的子程序。

用户可以对子程序进行重命名：执行**【重命名】**命令，将名字改为任何想要使用的名字；执行**【属性**…**】**命令，在该程序的“属性”对话框中进行修改。

3）建立中断程序

用户可以使用下面方法建立一个新的中断程序：

- 工程管理器中执行**【程序】→【新建中断程序】**菜单命令；
- 单击工具栏上的  图标。

建立的新中断程序默认名字是“INT_0”。在 INT_0 中用户可以编写自己需要的中断程序。

用户可以对中断程序进行重命名：执行**【重命名】**命令，将名字改为任何想要使用的名字；执行**【属性**…**】**命令，在该程序的“属性”对话框中进行修改。

4）程序的导入、导出

用户可以在一个工程中将编好的主程序、子程序和中断程序导出，并可以将导出的程序导入到另一个工程中，导入、导出功能可操作如下：

- 导入：工程管理器中执行**【程序】→【导入程序】**菜单命令，弹出“导入程序…”对话框，可分别选择主程序、子程序和中断程序文件。
- 导出：在工程管理器中右击要导出的程序，执行【导出】命令，弹出“程序保存…”对话框，用于程序保存路径选择和文件名的设置。

3.4 编译工程

在编写完工程后，必须对工程进行编译。在编译之前，软件会自动保存本工程，以确保编译的是用户最新的输入。但是，仍然建议用户注意随时保存自己的工作，以免发生意外。

用户可以使用如下任一方法来启动编译过程：

- 执行**【PLC】→【全部编译】**菜单命令；
- 单击工具栏上的  图标；
- 使用快捷键 **F7**。

编译的结果将会在下方信息输出窗口中的“编译信息”页面中显示。如果工程中有错误，双击编译信息中的错误信息就会自动定位到该错误所在的位置。用户需要根据错误信息提示对工程进行修改，直至编译成功。

3.5 下载工程

编译成功后，就可以将当前工程下载至 PLC 中。若需要指定 PC 机串口的通讯参数，请参阅 2.6 与计算机的连接。

用户可以使用如下任一方法来打开“下载工程”对话框：

- 单击工具栏上的  图标；
- 使用快捷键 **F8**。

“下载工程”对话框如下图所示：

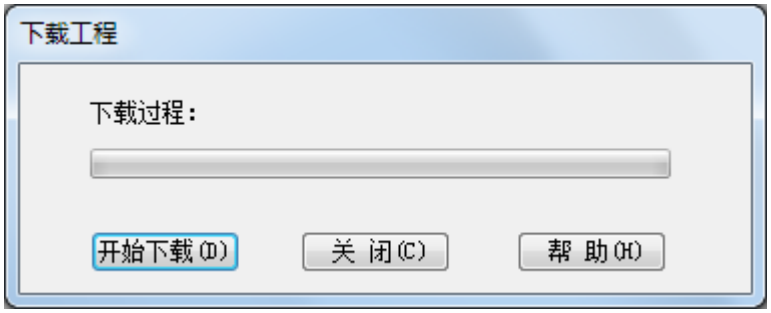


图 3-15 下载工程

单击【**开始下载**】按钮即可将工程下载至 PLC 中。我们可以执行【**工具**】→【**选项**】菜单，在【**软件设置**】页面下，对下载选项进行设置。如图 3-16 所示。

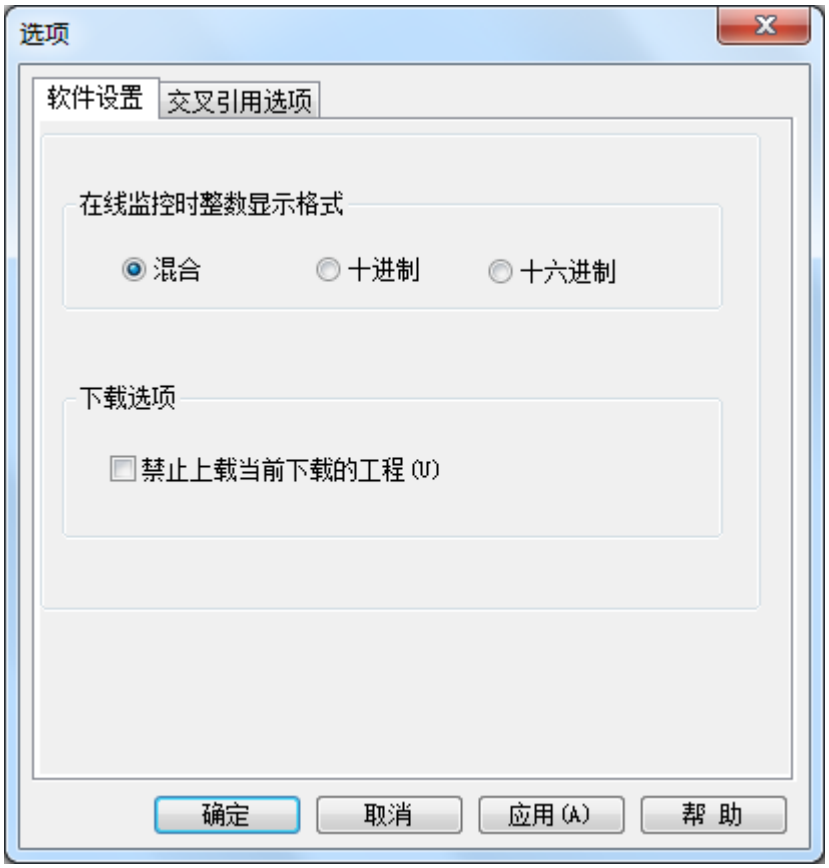


图 3-16 选项对话框

下载选项：

◆ **禁止上载当前下载的工程**

- 若选中此项，则在本次下载之后 CPU 将禁止任何人上载工程。
- 若不选中此项，则在本次下载之后 CPU 将允许用户按照原有的权限进行上载。

3.6 程序调试

用户可以使用 EC100/EC200 提供的在线监控、强制等功能对程序进行调试。

3.6.1 在线监控

在线监控有两种模式：

1）在状态表中进行监控。用户可以在表中输入想要监控的变量进行监控。

2）在程序中进行监控。用户可以观察到当前程序的运行情况。在线监控时不允许修改程序。

在线监控命令只有在打开状态表、LD 程序或者 IL 程序之后方能生效。注意在线监控命令是一种复选命令，若要脱离在线状态，再执行一次在线监控命令即可。

用户可以使用如下任一种方法进入在线监控状态：

- 执行【**调试**】→【**在线监控**】菜单命令；
- 单击工具栏上的  图标；
- 使用快捷键 **F6**。

在线监控状态之后可以对地址变量进行强制：

开关量：在触点、线圈上单击右键，执行弹出菜单中的【**强制为 TRUE**】、【**强制为 FALSE**】或者【**强制**…】命令；

非开关量：在变量名称上单击右键，执行弹出菜单中的【**强制**…】命令。

若要取消强制，执行相应的右键菜单命令即可。

3.6.2 状态表

用户可以利用状态表和示波器来对当前工程中用到的任意地址变量进行在线监控和强制。

3.6.2.1 监控变量的值

有如下三种方式可以进入状态表窗口：

- 双击工程管理器中的【**状态表**】；
- 右键单击【**状态表**】，执行打开菜单；
- 执行【**调试**】→【**状态表**】菜单命令。

用户可以按如下步骤来利用状态表监视用户程序中用到的变量的值：

- ✓ 在【**地址**】列中输入要监视的直接内存地址；
- ✓ 使用如下方法之一进入在线监控状态：

- 执行【**调试**】→【**在线监控**】菜单命令；
- 单击工具栏上的  图标；

- 使用快捷键 **F6**。
  - ✓ 单击工具栏上的图标；
  - ✓ 用户可以在任何时候改变监视值的【**显示格式**】。

状态表和示波器的样式如下图所示。

变量状态表

	地 址	全局变量名	显示格式	当前值	强制值	示波器
1	%VW0	plus	无符号整数	☐		不显示
2	%VW2	plus2	无符号整数	☐		不显示
3	%IO.0	input0	BOOL	☐		不显示
4	%IO.1	input1	BOOL	☐		不显示
5	%IO.2	input2	BOOL	☐		不显示
6	%IO.3	input3	BOOL	☐		不显示

图 3-17 状态表

表中共分如下六列：

【地　址】	输入将要被监测和强制的直接地址。
【全局变量名】	显示本行【地址】所对应的全局变量名。
【显示格式】	选择当前值和强制值的数据显示格式。 可选的格式有 BOOL、REAL、有符号整数、无符号整数、二进制、十六进制。
【当前值】	在线监控时将显示监测到的本行【地址】的值。 若复选框处于选中状态则表示本行的地址正处于强制状态。
【强制值】	在线监控时可以在此输入本行【地址】将要被强制为的值。
【示波器】	选择当前变量是否要在示波器中显示，示波器最多设置 10 通道，具体示波器功能使用详见 **3.6.2.5 示波器。**

注意：为了提高效率，EuraProg 只允许对当前工程中用到的变量进行监测和强制。若用户输入了未被用到的变量，EuraProg 虽然不提示错误，但当前值、强制值均不会起作用。

3.6.2.2 关于强制功能

用户可以使用强制功能来强制修改 PLC 中 I、Q、M、V、AI、AQ 区内的变量值，其中 I、Q、M 和 V 区中的变量可以按位、字节、字或者双字方式进行强制，AI、AQ 区中的变量可以按字方式进行强制。当 CPU 冷启动（重新上电）后，所有变量的强制状态都会被取消。

在扫描周期中的某一时刻，一个变量的取值会存在如下可能：外部的输入信号（I、AI）或者用户程序的执行结果（Q、AQ、M、V），用户指定的强制值。因此规定了如下变量取值的原则：

- 对于 M、V 区中的变量，强制值与程序执行结果处于同一优先级：若用户进行了强制，则在每个扫描周期的开始，强制值将会生效，之后程序执行结果将会生效。
- 对于 I、AI 区中的变量，强制值优先于外部信号输入值。若用户进行了强制，则在扫描过程中，强制值将会生效。

- 对于 Q、AQ 区中的变量，在扫描过程中以程序执行结果优先，但在扫描周期最后的输出任务中将会以强制值优先。

3.6.2.3 右键菜单

- **强制**：将【**强制值**】写入 PLC 内的直接内存【**地址**】中。
- **取消强制**：取消针对单击的行中 PLC 直接【**地址**】的强制状态。
- **全部强制**：将【**强制值**】列中的全部强制值都写入【**地址**】列中对应的 PLC 直接内存中。
- **全部取消强制**：取消针对【**地址**】列中所有 PLC 直接内存的强制状态。
- **读取全部强制**：读取所连 CPU 中所有被强制的地址及其强制值并显示于状态表中。

3.6.2.4 强制、取消强制

用户可以采用如下步骤来对一个内存变量进行强制或者取消强制：

✓ 在【**地址**】列中输入要强制的直接内存地址。

✓ 在【**显示格式**】列中可以选择、修改数值的显示格式。

✓ 在【**强制值**】列中输入想要的强制值。用户可以直接输入十进制的整数，EuraProg 会按照【**显示格式**】对数值的格式自动进行调整。

✓ 使用如下方法之一进行强制：

- 在本行单击鼠标右键，执行弹出菜单的【**强制**】命令；
- 鼠标单击本行任何一处，然后单击工具栏上的  图标；
- 鼠标单击本行任何一处，然后执行【**调试**】→【**强制**】菜单命令。

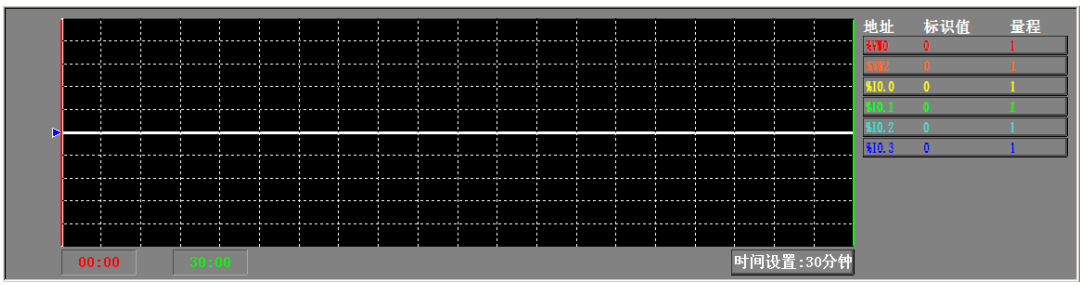
✓ 使用如下方法之一取消针对本行中 PLC 直接【地址】的强制状态：

- 在本行单击鼠标右键，执行弹出菜单的【**取消强制**】命令；
- 鼠标单击本行任何一处，然后单击工具栏上的  图标；
- 鼠标单击本行任何一处，然后执行【**调试**】→【**取消强制**】菜单命令。

✓ 状态表的编辑方式与初始化数据表一样，请参考初始化数据表中的相关内容。

3.6.2.5 示波器

示波器显示变量如图所示：



单击工具栏上的图标，即可启动示波器。

单击工具栏上的图标，即可停止示波器。

该示波器监控功能具体介绍如下：（以下功能均在停止示波器后操作）

- 左键双击右侧的变量，可设置显示变量的量程和波形颜色；
- 左键双击下方的时间设置按钮，可以设置示波器的时间量程，最长时间为 30 分钟；
- 拖动左侧的三角图标可以使波形上下移动；
- 左键单击红色竖线或者单击左下方红色时间显示方框即可选中红色竖线（绿线同样操作），左键单击波形显示区域可随意移动选中竖线，并会在右侧标识值列显示当前波形的值；
- 右键可以左右拖动波形；
- 示波器具有记忆功能，再次启动示波器后，之前采集的数据全部清零，重新计数。

3.7 软件其它功能

本节对 EuraProg 软件的其它功能、使用进行了详细的描述，用户在掌握基本概念的基础上，通过阅读本节可以快速认识并理解 EuraProg 的其它具体功能以及操作。

3.7.1 选项

用户在正式使用 EuraProg 之前需要对软件的一些默认值进行设置，设置结果将保存在当前计算机中，以后 EuraProg 在运行时将自动读取并使用这些设置值。

执行**【工具】→【选项】**菜单，可对**【工程属性默认值】**、**【在线监控时整数显示格式】**、**【下载选项】**进行设置。 **【选项】**对话框如下图：

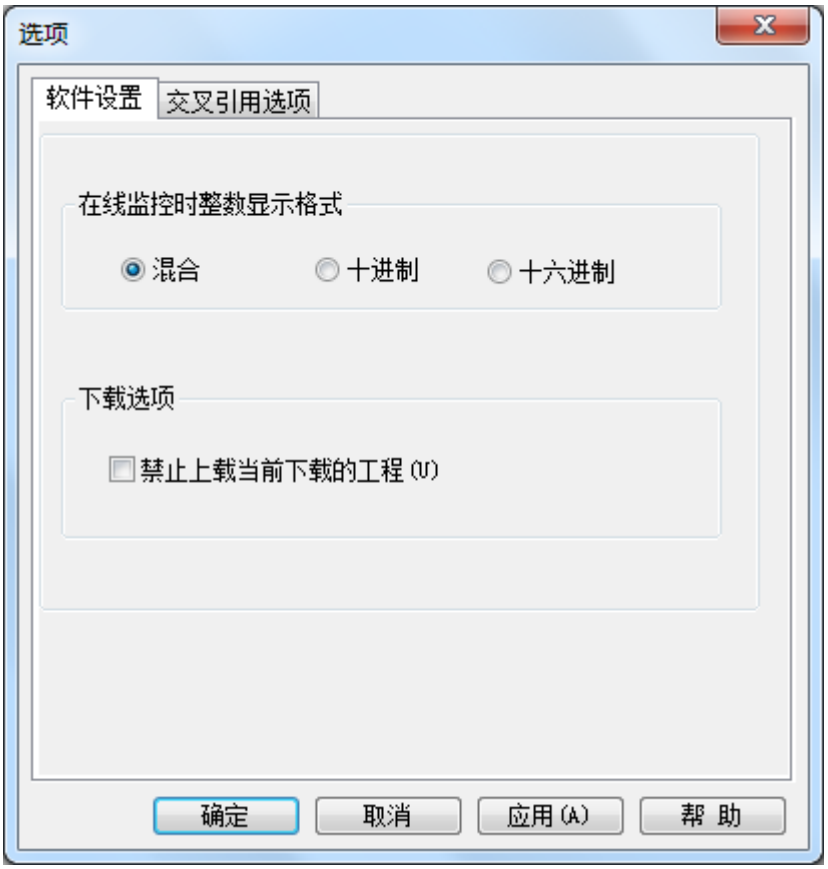


图 3-18　选项对话框

在线监控时整数显示格式

选择在线监控时编辑器中整数数据的显示格式。有如下三种选项：

- **混　　合**：INT、DINT 型数据以十进制格式，BYTE、WORD、DWORD 型数据以十六进制显示。
- **十 进 制**：所有整数都以十进制格式显示。
- **十六进制**：所有整数都以十六进制格式显示。

下载选项的说明见 3.5 下载工程

3.7.2　浮动窗口

工程管理器、指令集窗口、信息输出窗口均被设计成浮动窗口，它们可以停靠在 EuraProg 主窗口的任意一边。

单击浮动窗口右上角的  图标，即可让该窗口自动隐藏。

在自动隐藏状态下，窗口自动收缩成一个图标并停留在主窗口边缘。此时将鼠标指向该图标并停留片刻，窗口就会自动出现，然后若将鼠标置于窗口之外，它又将收缩于屏幕边缘。

在自动隐藏状态下，单击窗口右上角的  图标，窗口就会取消隐藏状态并重新停靠至上一次的停靠位置。

3.7.3 全局变量表

在全局变量表中用户可以方便地完成全局变量的定义，在 IEC 61131-3 中，全局变量的关键字是 VAR_GLOBAL。全局变量可以在所有的 POU 内读写。全局变量表窗口分为**【全局变量】**和**【功能块实例】**两个页面。

- **【全局变量】**页面

用于定义全局变量，也就是直接存取 PLC 内存地址的符号变量。

全局变量在程序中可以等效地代替对应的 PLC 直接内存地址使用，从而能够保证用户程序具有良好的可读性。每一个内存地址只允许赋予一个符号变量名，同样地，每一个符号变量名只允许有一个对应的内存地址。

符号变量的命名规则及全局变量的更多信息 4.2 数据及参数类型

全局变量表如下图 3-19 所示：

全局变量表

	全局变量名	地 址	数据类型	注 释
1	mstart	%I0.0	BOOL	电机启动
2	mstop	%I0.2	BOOL	电机停止
3	mrun	%I0.3	BOOL	电机运行
4				
5				
6				

全局变量 功能块实例

图 3-19 **【全局变量】**页面

- **【功能块实例】**页面

为了方便用户的使用，功能块实例的定义是由 EuraProg 自动完成的，因此在**【功能块实例】**页面中，所有的表格单元都是不可编辑的，其信息仅供用户进行参考。如图 3-20 所示：

图 3-20 **【功能块实例】**页面

3.7.3.1 进入全局变量表

有如下三种方式可以进入全局变量表窗口：

- 执行菜单**【工程】→【全局变量表】**；
- 双击工程管理器中的**【符号表】→【全局变量表】**；
- 右键单击**【全局变量表】**，执行打开菜单。

3.7.3.2 声明全局变量

全局变量表中共有 4 列：**【全局变量名】**、**【地址】**、**【数据类型】**和**【注释】**列。

用户可以按照如下步骤来声明一个全局变量：

- 进入全局变量表窗口并切换至**【全局变量】**页面；
- 在**【全局变量名】**列中输入一个变量名称并确认；
- 在**【地址】**列中输入一个直接内存地址并确认；
- 在**【数据类型】**列中为变量选择一个数据类型并确认；
- 在**【注释】**列中为这个全局变量输入注释。

如图 3-19 的第 1 行的含义是：为地址“%I0.0”定义一个符号名称为“mstart”，其数据类型为“BOOL”。在程序中，“%I0.0”与“mstart”是等价的。

全局变量表中的编辑方式与初始化数据表一样，请参考初始化数据表中的相关内容。

3.7.4 交叉引用表

交叉引用表用于分析当前工程中用到的所有地址变量并列表显示详细信息。使用交叉引用表便于用户对当前的工程进行统计、分析。

交叉引用表中的信息在第一次编译之后才能生成，并在以后的编译过程中自动刷新。

交叉引用表如下图所示：

交叉引用表

序号	地址	全局变量名	POU	位置	读写
0	%I0.0	input0	MAIN	Network 2	读
1	%I0.1	input1	MAIN	Network 3	读
2	%I0.1	input1	MAIN	Network 19	读
3	%I0.1	input1	MAIN	Network 20	读
4	%I0.1	input1	MAIN	Network 21	读
5	%I0.1	input1	MAIN	Network 22	读
6	%I0.1	input1	MAIN	Network 23	读
7	%I0.1	input1	MAIN	Network 24	读
8	%I0.1	input1	MAIN	Network 25	读
9	%I0.1	input1	MAIN	Network 26	读
10	%I0.1	input1	MAIN	Network 27	读
11	%I0.1	input1	MAIN	Network 28	读
12	%I0.1	input1	MAIN	Network 29	读
13	%I0.1	input1	MAIN	Network 30	读
14	%I0.1	input1	MAIN	Network 31	读
15	%I0.1	input1	MAIN	Network 32	读
16	%I0.1	input1	MAIN	Network 33	读
17	%I0.1	input1	MAIN	Network 34	读
18	%I0.1	input1	MAIN	Network 35	读
19	%I0.2	input2	MAIN	Network 4	读
20	%I0.3	input3	MAIN	Network 5	读
21	%I0.4		MAIN	Network 6	读
22	%I0.5		MAIN	Network 7	读
23	%I0.6		MAIN	Network 8	读
24	%I0.7		MAIN	Network 8	读
25	%I0.7		MAIN	Network 9	读
26	%I1.0		MAIN	Network 8	读
27	%I1.0		MAIN	Network 10	读
28	%I1.1		MAIN	Network 11	读
29	%I1.2		MAIN	Network 11	读
30	%I1.3		MAIN	Network 11	读
31	%I1.4		MAIN	Network 11	读
32	%I1.5		MAIN	Network 11	读

交叉引用 I区 Q区 M区 V区 SM区 T区 C区 HC区 AI区 AQ区 RS区 SR区

图 3-21　交叉引用表

- **【地　　址】**：显示了当前工程中用到的所有内存地址。
- **【全局变量名】**：显示本行**【地址】**所对应的全局变量名。
- **【P　O　U】**：显示本行**【地址】**所在的 POU 名称。
- **【位　　置】**：指明本行**【地址】**在【POU】中的具体位置。

  若 POU 用 IL 编写，则显示的是行号；若用 LD 编写，则显示的是网络号。
- **【读　/　写】**：指明本行**【地址】**在所处的**【位置】**上进行了读或者写操作。

如图 3-21，表格中第一行的意思是：在本工程 MAIN 程序的第 7 行用到了一次%I0.1，此处对%I0.1 进行的是读操作。

3.7.4.1　进入交叉引用表

有如下三种方式可以进入交叉引用表窗口：

- 执行**【工程】→【交叉引用】**菜单命令；
- 鼠标单击工具栏上的  图标；
- 双击工程管理器中的**【交叉引用】**。

3.7.4.2 在交叉引用表中进行操作

✓ 变量定位

鼠标双击某一行，即可打开本行的【**POU**】并定位到【**地址**】所在的具体【**位置**】上。

✓ 右键菜单

在表格任何一行上单击鼠标右键，将弹出如下右键菜单：

- **刷新**：刷新显示交叉引用数据。
- **变量定位**：打开本行的【**POU**】并定位到【**地址**】所在的具体【**位置**】上。
- **过滤**：菜单会弹出【**交叉引用选项**】如下图 3-22 所示，对交叉引用表中显示的内容进行过滤。“仅显示已用”选型可以在各个区的显示类型进行切换，√选中后，在显示的各个区的表中只是显示已经用过的资源，√去掉后，在表中显示已用和未用资源。

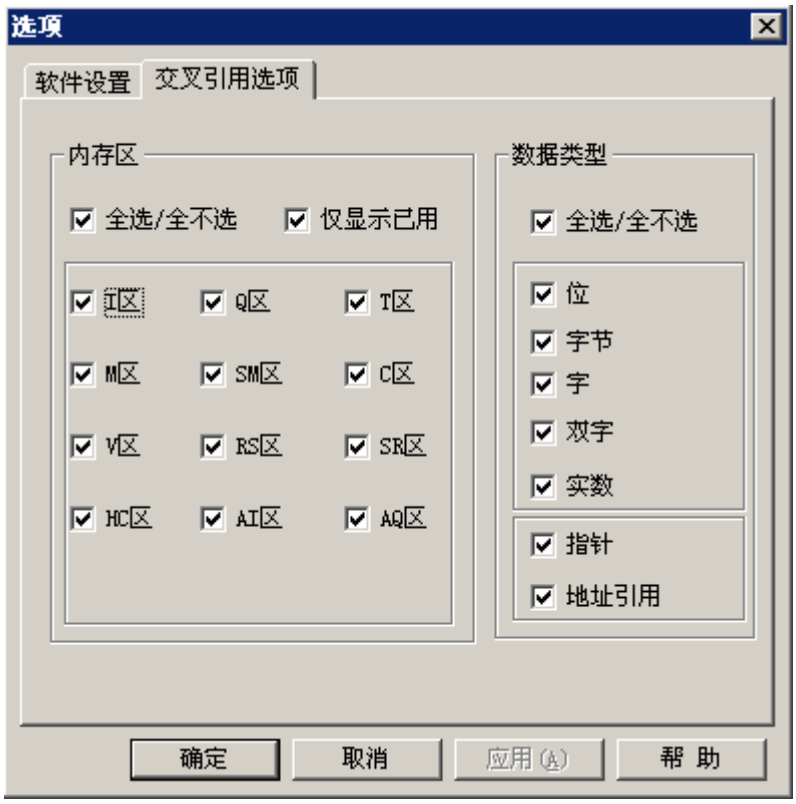


图 3-22 交叉引用选项

3.7.5 密码保护

EC100/EC200 的 CPU 提供了密码保护功能，允许用户通过对 CPU 进行加密来限制特定功能的使用。若 CPU 被加密，则用户在使用受限功能之前会被要求输入密码。此时若用户输入的密码正确，则 CPU 允许该操作；若用户输入的密码有误，则 CPU 将禁止该操作。输入

的密码只对当次操作有效，以后再使用受限功能时仍然需要输入密码。

CPU 允许的密码长度不超过 8 位，密码可以由数字、字母、下划线组成。

3.7.5.1 保护等级

CPU 的密码保护提供了如下 3 个等级：

等级 1：无保护。对 CPU 功能的使用没有限制。这是缺省的等级。

等级 2：部分保护。用户在执行下载功能时需要提供密码。

等级 3：最高保护。用户在执行上载、下载功能时需要提供密码。

3.7.5.2 修改密码和保护等级

执行【**PLC**】→【**修改密码**…】菜单命令，就可以进入“CPU 密码保护”窗口中进行修改。

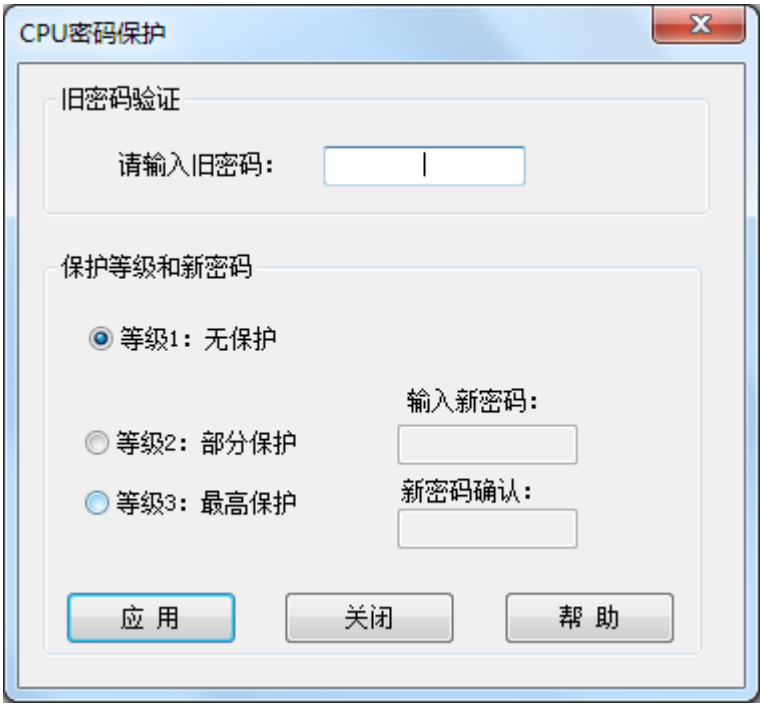


图 3-23 “CPU 密码保护”窗口

✓ 旧密码验证

若所连 CPU 原来已经被设置了密码保护，则必须在此正确输入原有的旧密码以供验证。若原来没有设置过密码保护，则输入框保持为空即可。

✓ 新保护等级和新密码：在此可以为所连 CPU 设置新的保护等级和密码。

- **保护等级**：可以从等级 1、等级 2、等级 3 中任选一种。
- **新密码**：在此输入新的密码。密码长度不得超过 8 位，可以由数字、字母、下划线组成。该输入框当保护等级选择为等级 2 或者等级 3 时生效。
- **新密码确认**：在此用户需要再次输入新的密码。

用户全部完成上述设置后，单击【**应用**】按钮，新的设置就将被写入所连 CPU 中，EuraProg

将会有对话框弹出来提示修改的结果。

3.7.5.3 忘记密码后的措施

如果用户忘记了 CPU 中的密码，那么所有受限的功能都将不能使用。

此时用户可以执行【**PLC**】→【**清除**…】菜单命令来清除 CPU 存储器中的所有内容，然后就可以继续使用这个 PLC 了。执行完【**清除**…】命令后，用户程序、配置数据、密码等等将全部被清除，CPU 将恢复成出厂的初始状态，但是时钟仍然保持清除之前的设置不变。此时，通讯参数是：PLC 站号为 1，波特率 9600，无校验，数据位 8 位，停止位 1 位。

3.7.6 比较

EuraProg 为用户提供比较功能，该功能是将上位机中的工程文件与 PLC 中的工程文件相比较， 以确定 PLC 内工程是否为所需要工程。

此时用户可以执行【**PLC**】→【**比较**…】菜单命令来比较，如果上位机中的工程文件与 PLC 中的工程文件一致，则出现如下提示：

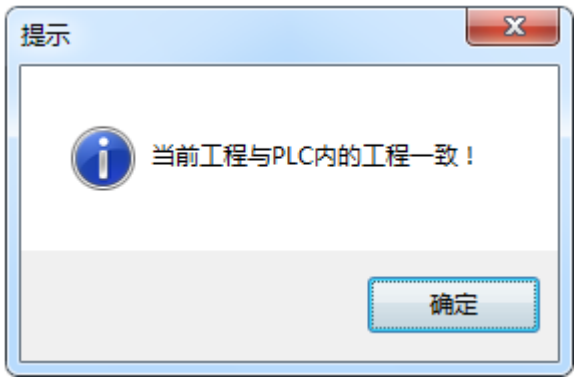


图 3-23 “比较一致”窗口

如果上位机中的工程文件与 PLC 中的工程文件不一致，则出现如下提示：

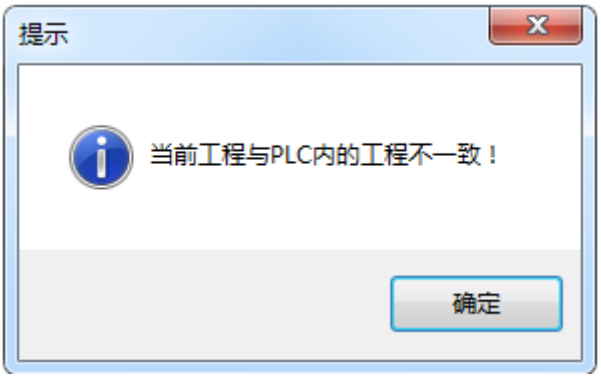


图 3-24 “比较不一致”窗口

3.7.7 清除

如果用户想要清除 CPU 内所有用户信息，此时用户可以执行 **【PLC】→【清除…】**菜单命令来清除 CPU 存储器中的所有内容。执行完 **【清除…】**命令后，用户程序、配置数据、密码等等将全部被清除，CPU 将恢复成出厂的初始状态，但是时钟仍然保持清除之前的设置不变。此时，通讯参数是：PLC 站号为 1，波特率 9600，无校验，数据位 8 位，停止位 1 位。

执行 **【PLC】→【清除…】**菜单命令，就可以进入 “清除…” 窗口中执行是否清除 CPU 内用户信息。

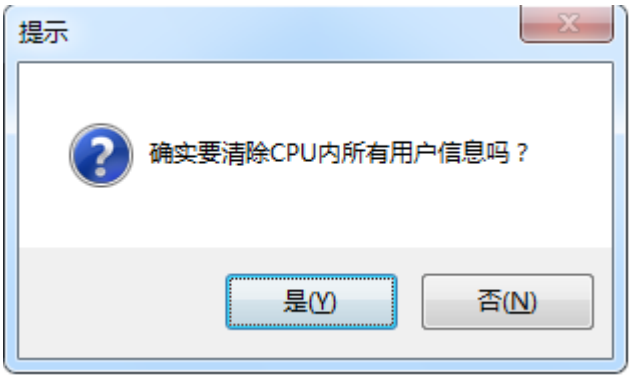


图 3-25 “清除…”窗口

3.7.8 信息

如果用户想要了解使用 PLC 的信息，执行**【PLC】→【信息…】**菜单命令，就可以进入“ PLC 信息…”窗口 ，了解 PLC 的 CPU 信息和模块的信息，其中模块信息为硬件配置中所设置的。CPU 信息包括“类型”、“当前状态”、“固件版本”、“工程版本”等关于 CPU 的信息。
模块信息包括“位置”、“模块型号”、“故障”、“故障描述”等有关模块的信息。

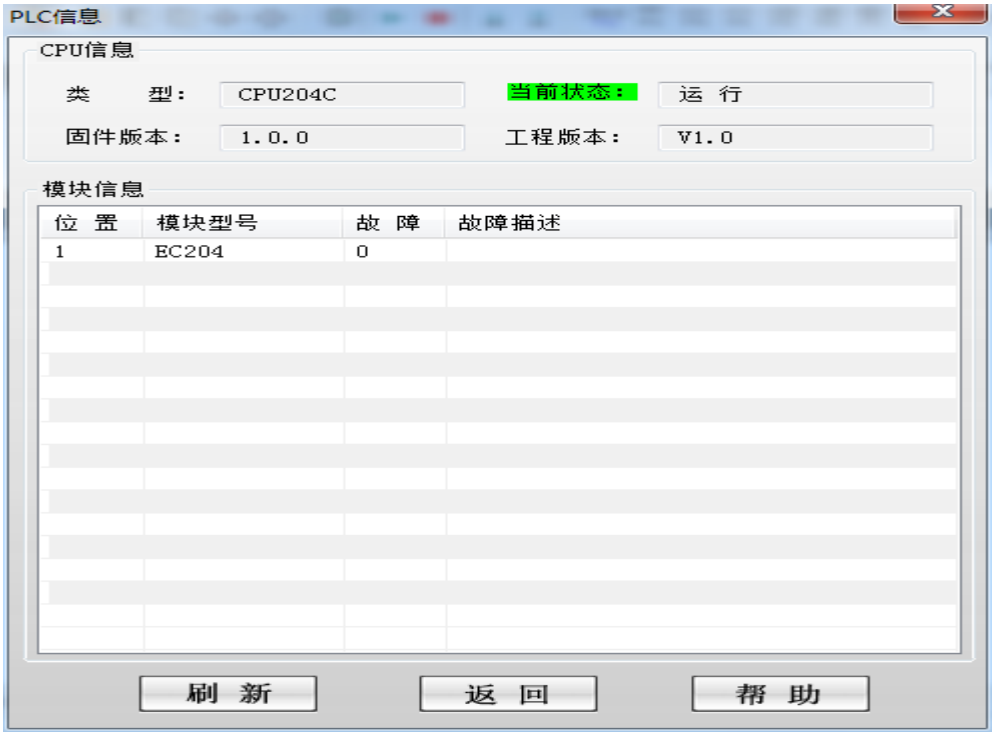


图 3-26 “ PLC 信息”窗口

第四章 PLC 编程基础

本章对编程的基础知识做了详细、系统的介绍，同时也介绍了 IEC 61131-3 标准中的一些基本概念，这些概念对于用户使用任何一种 IEC 61131-3 软件都是非常有用的。另外也对两种编辑器的功能、使用作了详细阐述。

4.1 程序组织单元

IEC 61131-3 引入 POU 的概念。POU 包含有程序代码，是独立的、最小的软件单元，它是程序和工程的基本单元。传统的 PLC 制造商在编程方面为各自的 PLC 定义了各种类型的块，IEC 61131-3 将这些块统一为 3 种类型的 POU。

用户程序由一个或者多个 POU 组成，这些 POU 可以是用户自己创建的。POU 之间可以相互调用，但是不允许递归调用，在 IEC 61131-3 中明确规定了 POU 禁止直接或者间接调用自身。

下面描述了标准的 POU 类型。

- 程序（Programme）

  关键字：**PROGRAMME**。

  这种类型的 POU 代表了“主程序”，用于执行一定的任务。

  它可以有输入和输出参数，但是无返回值。

- 功能（Function）

  关键字：**FUNCTION** 。

  这种类型的 POU 可以有输入参数，只有一个返回值，其返回值是通过功能名带回的。当以相同的输入参数调用功能时，其返回值总是相同的。

  它主要用于代码重用，可被其它 POU 调用。

- 功能块（Function Block）

  关键字：**FUNCTION_BLOCK**。

  这种类型的 POU 简称 FB。它可以有输入、输出参数，并具有静态变量（即能够记忆 FB 以前的状态）。FB 的输出值通过其输出参数传递。FB 的输出不仅取决于其输入值，而且也取决于在其静态变量中储存的状态值。

  FB 也主要用于代码重用，可被其它 POU 调用。

4.2 数据及参数类型

4.2.1 数据类型

数据类型定义了数据的位长度、取值范围及其初始化值。

在 IEC 61131-3 中定义了一组最常用的基本数据类型，因此在 PLC 领域内这些数据类型的含义以及使用方式是开放、统一的。目前 EC100/EC200 系列 PLC 支持的基本数据类型如

下表所示。

表 4-1　EC100/EC200 支持的基本数据类型

数据类型	描述	长度（位）	取值范围	缺省初始值
BOOL	布尔型	1	true, false	false
BYTE	8 位位串	8	0 ~ (2^8-1)	0
WORD	16 位位串	16	0 ~ (2^{16}-1)	0
DWORD	32 位位串	32	0 ~ (2^{32}-1)	0
INT	整型，有符号	16	-2^{15} ~（2^{15}-1）	0
DINT	双整型，有符号	32	-2^{31} ~（2^{31}-1）	0
REAL	实型	32	采用 ANSI/IEEE754-1985 标准；约 1.18*10^{-38} ~ 3.40*10^{38}，-3.40*10^{38} ~ -1.18*10^{-38}	0.0

4.2.2　标识符

标识符是由数字、字母、下划线字符组成的字符串，它必须以一个字母或者下划线开始。编程人员可以使用标识符来为变量、程序等定义名称。

4.2.2.1　标识符的定义

标识符的定义和使用必须依据如下原则：

- 必须以一个字母或者一个单一的下划线字符开始，随后是一定数量的数字、字母或者下划线。
- 标识符是大小写无关的。比如，abc、ABC、aBC 是同一个标识符。
- 标识符的长度仅受各编程系统的限制。在 EuraProg 中，标识符的最大长度是 16 个字符。
- 用户自定义的标识符不允许使用关键字。关键字是标准的标识符，其拼写形式和使用目的均由 IEC 61131-3 明确规定。

4.2.2.2　标识符的使用

下面列出了在 EuraProg 中可使用标识符的语言元素：

- 程序、功能、功能块名称
- 变量
- 语句标号等

4.2.3　常量

在程序运行的过程中，其值不能改变的量称为常量。常量可以分为数字常量、字符串常量和时间常量。常量的特性由它的值和数据类型来描述。根据数据类型的不同，常量的书写格式各不相同。下表列出了 EC100/EC200 支持的各种类型常量的定义及示例。

表 4-2　常量的定义

数据类型	格 式[1]	取值范围	示例
BOOL	true、false	true 代表真，false 代表假	false
BYTE	**B#**十进制数字	B#0 ~ B#255	B#129
	B#2#二进制数字		B#2#10010110
	B#8#八进制数字		B#8#173
	B#16#十六进制数字		B#16#3E
WORD	**W#**十进制数字	W#0~ W#65535	W#39675
	2#二进制数字		2#100110011
	W#2#二进制数字		W#2#110011
	8#八进制数字		8#7432
	W#8#八进制数字		W#8#174732
	16#十六进制数字		16#6A7D
	W#16#十六进制数字		W#16#9BFE
DWORD	**DW#**十进制数字	DW#0 ~ DW#4294967295	DW#547321
	DW#2#二进制数字		DW#2#10111
	DW#8#八进制数字		DW#8#76543
	DW#16#十六进制数字		DW#16#FF7D
INT	十进制数字	-32768~32767	12345
	I#十进制数字		I#-2345
	I#2#二进制数字[2]		I#2#1111110
	I#8#八进制数字[2]		I#8#16732
	I#16#十六进制数字[2]		I#16#7FFF
DINT	**DI#**十进制数字	DI#-2147483648~DI#2147483647	DI#8976540
	DI#2#二进制数字[2]		DI#2#101111
	DI#8#八进制数字[2]		DI#8#126732
	DI#16#十六进制数字[2]		DI#16#2A7FF
REAL	带小数点的十进制数字	$1.18*10^{-38}$ ~ $3.40*10^{38}$， $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	1.0，-243.456
	指数形式：x**E**y x：带小数点的十进制数字 y：整数。		-2.3E-23

提示：

(1) 在 IEC 61131-3 中，标识符的使用是大小写无关的。因此在程序中将格式字符写为大写或者小写均合法，比如 W#234, dw#12345 均为合法常量。

具体请参见关于标识符的定义部分。

(2) INT、DINT 型常量的二进制、八进制、十六进制表示方法均采用了通用计算机中标准的补码表示法，其最高有效位（MSB）是符号位：MSB 为 1 则代表负数，MSB 为 0 则代表

正数。比如 I#16#FFFF＝-1，I#7FFF＝32767，I#8000＝-32768 等。

4.2.4 变量

在程序的运行过程中，其值可以改变的量称为变量。

一个变量必须有一个名字，占据一定的存储单元，在该存储单元中存放着变量的值。在 IEC 61131-3 中，变量的存储位置可以由用户自行指定一个有效的 PLC 内存地址，也可以由编程系统自行分配。

4.2.4.1 变量声明

变量的使用必须遵循“先声明，后使用”的原则。变量的命名请参阅 4.2.2.1 标识符的定义。在声明变量时，必须为它指定一个确定的数据类型，同时也必须指定它的变量类型。

在 IEC 61131-3 中定义了各种变量类型。比如，变量可以被定义为一个 POU 的形式参数；也可以在一个 POU 内定义，仅作为本 POU 的局部变量；也可以在 POU 外定义，在整个工程范围内作为全局变量使用。下表描述了 EC100/EC200 支持的标准变量类型。

表 4-3　EC100/EC200 支持的变量类型

变量类型	存储权限		描述
	外部	内部	
VAR	---	读写	局部变量。 只能在定义它的 POU 内使用，在此 POU 外部是不可见的。
VAR_INPUT	写	读	输入变量。在定义它的 POU 内作为 POU 的输入参数仅可读，在调用此 POU 时此变量仅可写。
VAR_OUTPUT	读	读写	输出变量。在定义它的 POU 内作为 POU 的输出参数可读写，在调用此 POU 时此变量仅可读。
VAR_IN_OUT	读写	读写	输入/输出变量，是 VAR_INPUT 和 VAR_OUTPUT 的组合类型。在定义它的 POU 内以及在调用此 POU 时此变量均可读写。
VAR_GLOBAL	读写	读写	全局变量。可被所有的 POU 直接读写。

4.2.4.2 在 EuraProg 中声明变量

在 EuraProg 中，各种变量的声明均在相应的表格中进行，这样既避免了让用户进行繁琐的输入，同时软件还能够对用户的输入进行严格的语法检查。

全局变量的声明在全局变量表中完成。POU 的局部变量、形式参数在该 POU 编辑界面的变量声明表格中完成。若全局变量与局部变量的名称相同，则在程序中局部变量的使用优先。

具体的使用方法请参见本手册中关于界面的详细介绍部分。

4.2.4.3 变量的检验

在编辑、编译程序时，EuraProg 可以自动对变量的使用情况进行检验，即判别该变量是否按照其变量类型、数据类型进行处理。这也是 IEC 61131-3 的重要优点之一。比如，在程序中为一个 WORD 类型的变量赋一个 BOOL 类型的值，或者对一个 VAR_INPUT 型的变量进行赋值操作，EuraProg 就会向用户进行错误警告并提示修改。

由于变量的特性取决于其变量类型和数据类型，因此这种检验能够在很大程度上避免由于变量使用而引起的错误。

4.3 EC100/EC200 的内存区域及寻址方式

内存区域中的每个单元都有明确、唯一的编号，这个编号就是内存单元的物理地址。物理地址是一个抽象的数值，不便于在程序中使用。所以为了方便用户，在 EC100/EC200 中内存被进一步划分为不同类型的几个区域并对各区域重新按字节进行编址，为每个内存单元都分配了一个直接地址，例如%I0.0、%VW20 等。在这种情况下，直接地址就如同一个变量一样。实际上在本书后面的描述中，也经常用到诸如“直接地址变量”、“变量%VW100”这样的说法。

4.3.1 内存区域类型及其特性

表 4-4 EC100/EC200 的内存区域分配

I	
描述	DI（开关量输入）映像区 在每个扫描周期的开始，CPU 读取所有物理 DI 通道的状态并将这些状态写入 I 区中以供用户程序使用。
访问方式	可按位、字节、字、双字访问
存储权限	只读
其它	允许强制，不能掉电保持
Q	
描述	DO（开关量输出）映像区 在每个扫描周期的结束，CPU 将 Q 区中的值全部输出至物理 DO 通道。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	允许强制，不能掉电保持
AI	
描述	AI（模拟量输入）映像区 AI 扩展模块对模拟量输入信号进行采样并转换为整型数值。在每个扫描周期的开始，CPU 从所有 AI 扩展模块读取测量值并写入 AI 区中以供用户程序使用。
访问方式	可按字（INT 型）访问
存储权限	可读

其它	允许强制，不能掉电保持
AQ	
描述	AO（模拟量输出）映像区 在每个扫描周期的结束，CPU 将 AQ 区中的所有数值全部输出至物理 AO 通道。
访问方式	可按字（INT 型）访问
存储权限	可读、可写
其它	允许强制，不能掉电保持
HC	
描述	高速计数器区。用于存放各高速计数器的当前计数值。
访问方式	可按双字（DINT 型）访问
存储权限	可读
其它	不允许强制，不能掉电保持
V	
描述	变量存储区。该区域比较大，可用于存储大量的数据。
访问方式	可按字节、字、双字访问；部分可按位访问，具体见表 4-5。
存储权限	可读、可写
其它	允许强制，允许掉电保持
M	
描述	内部存储区。用于一些中间状态或者其它数据。 与 V 区相比，M 区的访问速度更快，更有利于位操作。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	允许强制，允许掉电保持
SM	
描述	系统存储区。 用于存储数据。用户程序可以访问 SM 区中的一些地址来获取系统当前的状态信息，也可以利用 SM 区的一些地址来选择和控制 CPU 的某些特殊功能。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	不允许强制，不能掉电保持
L	
描述	局部变量区。 所有 POU 的局部变量、输入/输出参数均在 L 区内自动分配地址。 **不建议用户直接操作 L 区**。
访问方式	可按位、字节、字、双字访问
存储权限	可读、可写
其它	不允许强制，不能掉电保持

4.3.2 内存区域的直接寻址

用户可以使用直接地址来存取其中的数据，这种方式称为直接寻址。请务必注意“直接地址”和“直接地址中的数据”这两个概念的区别。

4.3.2.1 直接地址表示格式

IEC 61131-3 规定，所有的直接地址都必须以“%”开始。

各内存区域的直接寻址方式如下述列表。表中的 x、y 均代表十进制数字。

- **I 区**

位寻址	格式	**%I**x.y
	描述	x：字节地址，表示在 I 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%I0.0　%I0.7　%I5.6
字节寻址	格式	**%IB**x
	描述	x：字节地址，即在 I 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%IB0　%IB1　%IB10
字寻址	格式	**%IW**x
	描述	x：字节地址，即在 I 区中该内存单元首字节的地址。 由于 WORD 类型的数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%IW0　%IW2　%IW12
双字寻址	格式	**%ID**x
	描述	x：字节地址，即在 I 区中该内存单元首字节的地址。 由于 DWORD 类型的数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%ID0　%ID4　%ID12

- **Q 区**

位寻址	格式	**%Q**x.y
	描述	x：字节地址，即在 Q 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%Q0.0　%Q0.7　%Q5.6
字节寻址	格式	**%QB**x
	描述	x：字节地址，即在 Q 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%QB0　%QB1　%QB10
字寻址	格式	**%QW**x
	描述	x：字节地址，即在 Q 区中该内存单元首字节的地址。 由于 WORD 类型的数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%QW0　%QW2　%QW12
双字寻址	格式	**%QD**x
	描述	x：字节地址，即在 Q 区中该内存单元首字节的地址。 由于 DWORD 类型的数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%QD0　%QD4　%QD12

- **M 区**

位寻址	格式	**%M**x.y
	描述	x：字节地址，即在 M 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%M0.0　%M0.7　%M5.6
字节寻址	格式	**%MB**x
	描述	x：字节地址，即在 M 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%MB0　%MB1　%MB10
字寻址	格式	**%MW**x
	描述	x：字节地址，即在 M 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%MW0　%MW2　%MW12
双字寻址	格式	**%MD**x
	描述	x：字节地址，即在 M 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%MD0　%MD4　%MD12

• **V 区**

位寻址	格式	**%V**x.y
	描述	x：字节地址，即在 V 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%V0.0　%V0.7　%V5.6
字节寻址	格式	**%VB**x
	描述	x：字节地址，即在 V 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%VB0　%VB1　%VB10
字寻址	格式	**%VW**x
	描述	x：字节地址，即在 V 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%VW0　%VW2　%VW12
双字寻址	格式	**%VD**x 或者 **%VR**x
	描述	x：字节地址，即在 V 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT（**%VD**x）；REAL（**%VR**x）
	示例	%VD0、%VD12；REAL 型：%VR0、%VR4

- **SM 区**

位寻址	格式	**%SM**x.y
	描述	x：字节地址，即在 SM 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%SM0.0　%SM0.7　%SM5.6
字节寻址	格式	**%SMB**x
	描述	x：字节地址，即在 SM 区中该内存单元所在字节的编号（地址）。
	数据类型	BYTE
	示例	%SMB0　%SMB1　%SMB10
字寻址	格式	**%SMW**x
	描述	x：字节地址，即在 SM 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%SMW0　%SMW2　%SMW12
双字寻址	格式	**%SMD**x
	描述	x：字节地址，即在 SM 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT
	示例	%SMD0　%SMD4　%SMD12

L 区（注意：不建议用户使用直接地址来访问 L 区）

位寻址	格式	**%L**x.y
	描述	x：字节地址，即在 L 区中该内存单元所在字节的编号（地址）。 y：位地址，表示位于字节 x 的第几位。范围为 0~7。
	数据类型	BOOL
	示例	%L0.0　%L0.7　%L5.6
字节寻址	格式	**%LB**x
	描述	x：字节地址，即在 L 区中该内存单元所在的字节的编号（地址）。
	数据类型	BYTE
	示例	%LB0　%LB1　%LB10
字寻址	格式	**%LW**x
	描述	x：字节地址，即在 L 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	WORD、INT
	示例	%LW0　%LW2　%LW12
双字寻址	格式	**%LD**x
	描述	x：字节地址，即在 L 区中该内存单元首字节的地址。 由于双字数据长度为 4 字节，因此 x 必须为偶数。
	数据类型	DWORD、DINT、REAL
	示例	%LD0　%LD4　%LD12

• AI 区

字寻址	格式	**%AIW**x
	描述	x：字节地址，即在 AI 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	INT
	示例	%AIW0　%AIW2　%AIW12

• AQ 区

字寻址	格式	**%AQW**x
	描述	x：字节地址，即在 AQ 区中该内存单元首字节的地址。 由于字数据长度为 2 字节，因此 x 必须为偶数。
	数据类型	INT
	示例	%AQW0　%AQW2　%AQW12

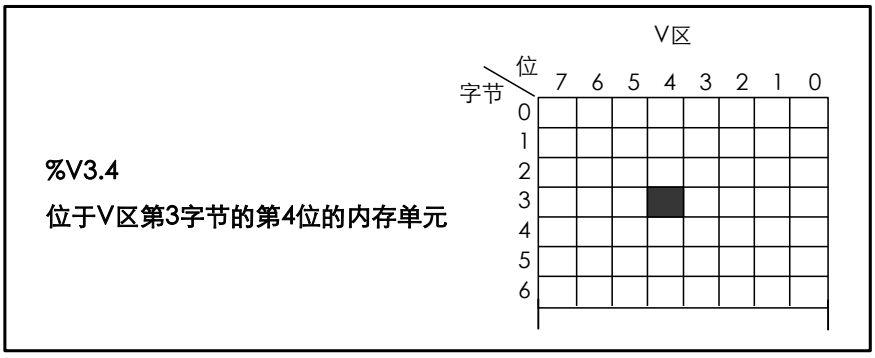
• **HC 区**

字寻址	格式	**%HC**x
	描述	x：高速计数器的编号。
	数据类型	DINT
	示例	%HC0　%HC1

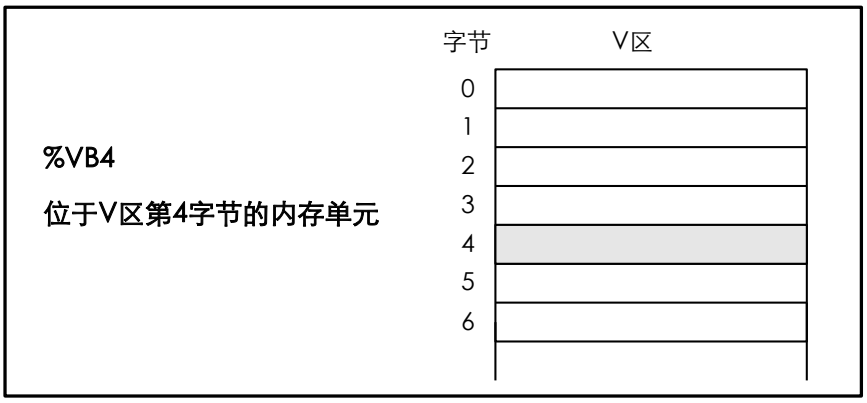
4.3.2.2　直接地址与内存单元之间的映射

每一个合法的直接地址都对应于 CPU 中的一个内存单元。在程序中对直接地址的操作就是对其对应的内存单元进行操作。下面将以 V 区为例图解直接地址与内存单元之间的映射关系。

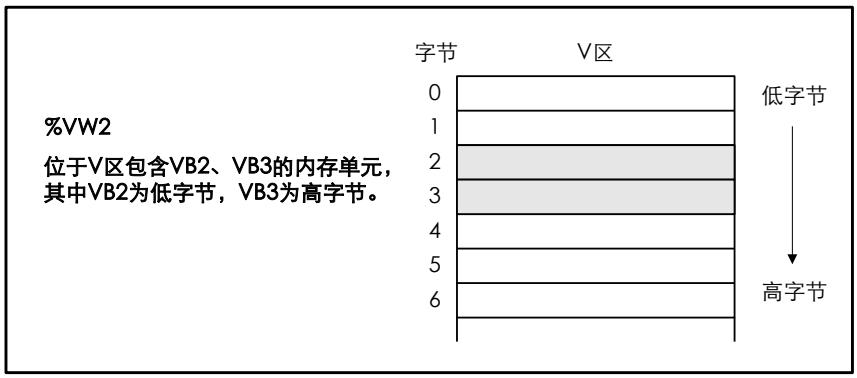
• **位地址：**



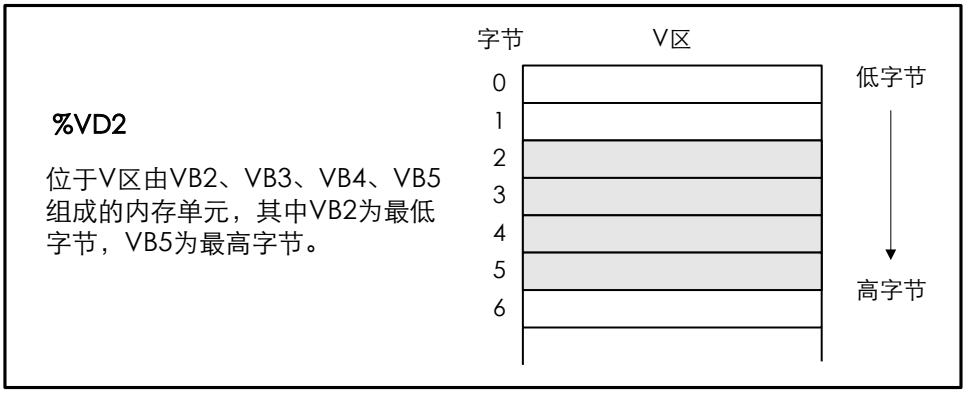
• **字节地址：**



• **字地址：**



• **双字地址：**



4.3.3 内存区域的间接寻址

指针是一个 32 位长度的变量，用于存放一个内存单元的物理地址。用户可以利用指针来存取对应内存单元中的数据，这种方式称为间接寻址。

在 EC100/EC200 中，只允许使用 V 区中的直接地址变量（双字长度）来作为指针。另外，只允许使用指针对 V 区进行间接寻址，但是不支持位变量的间接寻址。

4.3.3.1 建立指针

为了实现对某个内存单元的间接寻址，必须先建立起它的指针。这时候需要用到取地址运算符“&”，例如，&VB100 表示 VB100 的物理地址。

建立指针的方法为：使用取地址运算符获取某个内存单元的物理地址，并将其使用 MOVE 指令写入另一个直接地址变量来作为指针。例如：

```
MOVE    &VB0, %VD200      (* 建立指针 VD200，其中存放着 VB0 的物理地址 *)
MOVE    &VW2, %VD204      (* 建立指针 VD204，其中存放着 VW2 的物理地址 *)
```

4.3.3.2 使用指针存取数据

在指令中的操作数前加上“*”就表示该操作数是一个指针。“*”是指针运算符，在一个指针之前加“*”就代表了该指针所指向的直接地址变量。在使用指针作为指令的操作数时，需要注意指针所指向变量的数据类型。例如：

```
LD          %SM0.0
MOVE    &VB1, %VD400        (*  建立指针 VD400，其中存放着 VB1 的物理地址  *)
MOVE    *VD400, %VB20       (*  该指令将 VB1 的值赋给 VB20。因为指针 VD400 指向
                                直接地址变量 VB1，所以*VD400 代表了 VB1。*)
```

4.3.3.3 修改指针的值

由于指针是一个 32 位的变量，因此可以使用指令修改它的值，比如加法、减法等指令。需要注意的是，指针的值每改变 1，那么它所指向的直接地址就相应改变了 1 个字节。在修改指针的值时，需要注意它所指向变量的数据类型：

- 若指向字节长度（BYTE 型）的变量，则指针值的改变量可以是任意的双整数。
- 若指向字长度（INT 或者 WORD 型）的变量，则指针值的改变量须是 2 的倍数。
- 若指向双节长度（DINT、DWORD 或者 REAL 型）的变量，则指针值的改变量须是 4 的倍数。

4.3.3.4 使用指针的注意事项

- 指针的有效性是由用户程序自己保证的。指针比较灵活，因此用户使用时必须小心，如果在程序中让指针指向了一个非法的变量地址，那么就会导致不可预期的结果。
- EC100/EC200 只支持一重的指针与地址，多重的应用是非法的。比如下面的指令就是非法的：

```
MOVE    &VB6, *VD40
```

4.3.3.5 间接寻址示例

```
(* Network 0 *)
LD          %SM0.0
MOVE    &VW10, %VD400     (*  建立指针 VD400，其中存放着 VW10 的物理地址  *)
MOVE    *VD400, %VW60     (*  该指令将 VW10 的值赋给 VW60。因为指针 VD400 指向
                              直接地址变量 VW10，所以*VD400 代表了 VW10。*)
ADD     DI#2,   %VD400    (*  指针 VD400 的值增加 2，所以它指向的直接地址
                              增加了 2 字节，也就是说指针 VD400 指向了 VW12 *)
MOVE    *VD400, %VW62     (*  该指令将 VW12 的值赋给 VW62 *)
```

4.3.4 内存区域的地址范围

EC100/EC200 有几种不同型号的 CPU。各型 CPU 中内存区域的地址范围有所不同，超

出允许范围的地址是非法的。下表对此进行了详细说明。

表 4-5　EC100/EC200 内存区域的范围

		EC102	EC104	EC106	EC202、EC204、EC206
I	长度（字节）	1	2	3	32
	位地址	%I0.0 --- %I0.7	%I0.0 --- %I1.7	%I0.0 --- %I2.7	%I0.0 --- %I31.7
	字节地址	%IB0	%IB0 --- %IB1	%IB0 --- %IB2	%IB0 --- %IB31
	字地址	———	%IW0	%IW0	%IW0 --- % IW30
	双字地址	———	———	———	%ID0 --- %ID28
Q	长度（字节）	1	2	2	32
	位地址	%Q0.0---%Q0.7	%Q0.0---%Q1.7	%Q0.0--- %Q1.7	%Q0.0 --- %Q31.7
	字节地址	%QB0	%QB0、%QB1	%QB0、%QB1	%QB0 --- %QB31
	字地址	———	%QW0	%QW0	%QW0 --- %QW30
	双字地址	———	———	———	%QD0 --- %QD28
AI	长度（字节）	0	0	0	64
	字地址	———	———	———	AIW0 --- AIW62
AQ	长度（字节）	0	0	0	64
	字地址	———	———	———	AQW0 --- AQW62
HC	长度（字节）	16			
	双字地址	%HC0 --- %HC3			
V	长度（字节）	6144			10240
	位地址	%V0.0 ---%V2047.7（V 区可位操作存储区为 2K 字节）			
	字节地址	%VB0 --- %VB6143			%VB0---%VB10239
	字地址	%VW0 --- %VW6142			%VW0 --- %VW10238
	双字地址	%VD0 --- %VD6140　%VR0 --- %VR6140			%VD0 --- %VD10236 %VR0 --- %VR10236
M	长度（字节）	64			128
	位地址	%M0.0 --- %M63.7			%M0.0 --- %M127.7
	字节地址	%MB0 --- %MB63			%MB0 --- %MB127
	字地址	%MW0 --- %MW62			%MW0 --- %MW126
	双字地址	%MD0 --- %MD60			%MD0 --- %MD124
SM	长度（字节）	300			
	位地址	%SM0.0 --- %SM299.7			
	字节地址	%SMB0 --- %SMB299			
	字地址	%SMW0 --- %SMW298			
	双字地址	%SMD0 --- %SMD296			

L	长度(字节)	272
	位地址	%L0.0 --- %L271.7
	字节地址	%LB0 --- %LB271
	字地址	%LW0 --- %LW270
	双字地址	%LD0 --- %LD268

4.3.5 关于功能块及功能块实例

4.3.5.1 IEC 61131-3 中定义的标准功能块

在 IEC 61131-3 中定义了如下标准的功能块：

- 定时器：TON --- 接通延时定时器；TONR --- 保持型接通延时定时器；TOF --- 断开延时定时器；TP --- 脉冲定时器。
- 计数器：CTU --- 加计数器；CTD --- 减计数器；CTUD --- 加/减计数器。
- 双稳态触发器：SR --- SR 触发器；RS --- RS 触发器。
- 边沿检测： R_TRIG --- 上升沿检测； F_TRIG --- 下降沿检测。

4.3.5.2 FB 的实例化

在 IEC 61131-3 中，“FB 实例化”的概念特别重要。

所谓实例化，就是用户在变量声明部分通过指定变量名称及其数据类型来建立一个变量。

FB 也需要如同变量那样首先进行实例化。在程序中不允许直接调用 FB，而只能调用 FB 的实例。形象地讲，在程序中不能读写一个数据类型（比如 INT 型），而只能读写声明为该数据类型（比如 INT 型）的具体的变量。如下图，在程序中只能调用、访问 T1。

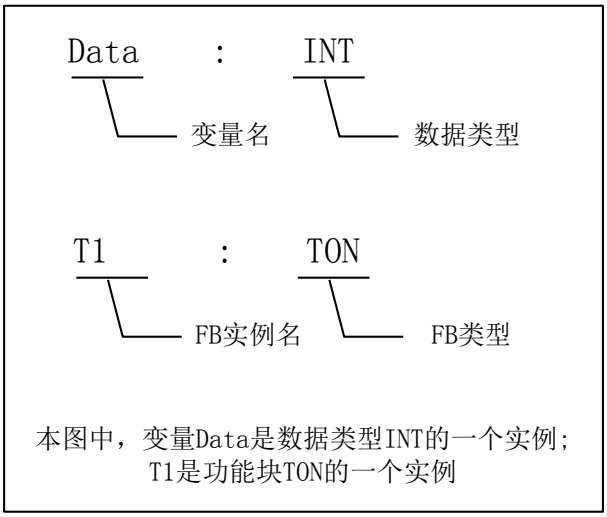


图 4-1 实例化举例

4.3.5.3 FB 实例存储区

EC100/EC200 在内存中为每一种 FB 类型都分配了一个实例存储区域。当定义了一个 FB

实例时，EuraProg 会自动在对应的存储区域内为该实例分配一个独立的存储单元。

下表描述了 EC100/EC200 的 FB 实例存储区域。

表 4-6　FB 实例的存储区

T	
描述	定时器区，可在该区域内分配 TON、TONR、TOF、TP 的实例。 用于存储所有定时器实例的状态值和当前计时值。
访问方式	直接访问定时器的状态值、当前计时值
存储权限	可读
其它	允许掉电保持，不允许强制
C	
描述	计数器区，可在该区域内分配 CTU、CTD、CTUD 的实例。 用于存储所有计数器实例的状态值和当前计数值。
访问方式	直接访问计数器的状态值、计数值
存储权限	可读
其它	允许掉电保持，不允许强制
RS	
描述	RS 触发器区，可在该区域内分配 RS 的实例。 用于存储所有 RS 实例的状态值。
访问方式	直接访问 RS 状态值
存储权限	可读
其它	不允许掉电保持，不允许强制
SR	
描述	SR 触发器区，可在该区域内分配 SR 的实例。 用于存储所有 SR 实例的状态值。
访问方式	直接访问 SR 状态值
存储权限	可读
其它	不允许掉电保持，不允许强制

4.3.6　FB 实例的命名与使用

FB 的实例遵循“先声明，后使用”的原则。

为了方便用户，在 EuraProg 中特意作了如下处理：FB 实例的命名遵循传统 PLC 的常用方式，比如 T0、C3 等。用户不需要手工输入 FB 实例的声明语句，只需在程序中调用合法的功能块实例即可，软件将会在全局变量表中为用户调用的实例自动生成声明语句。

FB 实例存储区的直接寻址方式如下述列表。

• T

直接寻址	格式	**T**x
	描述	x：定时器编号，十进制数字。
	数据类型	BOOL --- 定时器的状态值 INT --- 定时器的当前计时值。 **T**x 兼具以上两种含义，但用户只需在程序中使用实例名即可，其含义将由软件自动识别。
	示例	T0、T5、T20

• C

直接寻址	格式	**C**x
	描述	x：计数器编号，十进制数字。
	数据类型	BOOL --- 计数器的状态值 INT --- 计数器的当前计时值。 **C**x 兼具以上两种含义，但用户只需在程序中使用实例名即可，其含义将由软件自动识别。
	示例	C0、C5、C20

• RS

直接寻址	格式	**RS**x
	描述	x：RS 触发器编号，十进制数字。
	数据类型	BOOL --- RS 触发器的状态值
	示例	RS0、RS5、RS10

• SR

直接寻址	格式	**SR**x
	描述	x：SR 触发器编号，十进制数字。
	数据类型	BOOL --- SR 触发器的状态值
	示例	SR0、SR5、SR10

4.3.7 FB 实例存储区的范围

系统能够为各种 FB 类型分配的存储区域的大小受到硬件本身资源的限制，因此，EC100/EC200 系列各型号的 CPU 为各 FB 实例存储区分配的范围有所不同，如下表所示：

表 4-7　FB 实例存储区的分配

		EC102、EC104、EC106、EC202、EC204、EC206
T	允许数量	256
	范围	T0 --- T255
	时基	T0 --- T3：1ms T4 --- T19：10ms T20 --- T255：100ms
	最大定时时间	32767*时基
C	允许数量	256
	范围	C0 --- C255
	最大计数值	32767
RS	允许数量	32
	范围	RS0 --- RS31
SR	允许数量	16
	范围	SR0 --- SR31

4.4 IL 编程

针对 PLC 应用程序的编写，IEC 61131-3 中规定了 3 种文本化语言和 3 种图形化语言。文本化语言包括：指令表（IL）、结构化文本（ST）、顺序功能图（SFC，文本化版本）；图形化语言包括：梯形图（LD）、功能块图（FBD）、顺序功能图（SFC，图形化版本）。

EuraProg 目前支持 IL 语言和 LD 语言编程。在一个工程中，IL 和 LD 语言可以混用，但是在一个 POU 中只允许使用一种语言。用户可以随时执行**【查看】→【IL 编辑器】**或者**【查看】→【LD 编辑器】**菜单命令切换当前程序的语言。注意：任何 LD 程序都可以转换为 IL 程序，但只有按照一定规则编写的 IL 程序才可以转换成 LD 程序。

本节对 EuraProg 中 IL 编辑器的功能、使用进行了详细的描述，同时也阐述了 IEC 61131-3 标准中关于 IL 语言的相关语法、规定。通过阅读本节，用户可以轻松地使用 EuraProg 编写出符合 IEC 标准的应用程序。

4.4.1 IL 的背景

IL 语言是一种低级语言，与汇编语言非常相似，是在借鉴、吸收世界上著名 PLC 厂商指令表语言的基础上形成的一种标准语言。

IL 接近于机器码，因此使用 IL 编写的程序效率更高、更加紧凑。IL 很适合经验丰富的

编程人员使用，有时候使用 IL 语言可以解决 LD 等图形化语言难以解决的问题。

4.4.2 IL 的语法规定

4.4.2.1 IL 语法格式

IL 程序面向行，每行语句只能有一条指令或者一个标号。IL 程序中允许有空白行存在。

IL 程序中一个语句的基本格式如下图：

```
标号：
操作符/功能/功能块    操作数（表）  (* 注释 *)
```

- **标号**

  可选；

  使用标号的目的在于实现程序的跳转。标号的命名规则与变量的命名规则相同。

- **操作符/功能/功能块**
- **操作数表**

  请参见下一章中对于各操作符、功能、功能块的详细说明，其中包含相应操作数的描述。

  在操作符、操作数之间至少有一个空格。

- **注释**

  可选；

  在所有的语言中，注释的格式都是相同的，由 (* *) 进行界定；

  每行只允许有一个注释，注释也可以单独占用一行；

  注释不允许嵌套，否则将被编译器认为是错误。

  IL 语句举例如下：

```
(* NETWORK 0 *)
AAA:                    (* 标号，供跳转时使用 *)
LD      %I0.1
TP      T2, 168     (* 若 I0.1 为 true，启动定时器 T2，T2 被声明为 TP 型 *)
```

4.4.2.2 关于 CR

IL 中提供了一个叫做“Current Result（CR）”的通用累加器，在 CR 中存储了用户程序的当前执行结果。用户程序中的每一行语句执行后，CR 都会被刷新。依据后续语句的不同，CR 值可能作为下一条语句的执行条件，也可能作为下一条语句的操作数之一。

操作符不同，执行之后对 CR 值的影响也不同。下表根据对 CR 值不同的影响将 EuraProg 中的操作符进行了初步的分组。更详细的描述请参见下一章对于各指令的介绍。

表 4-8　各操作符执行之后对于 CR 的影响

操作符	分组缩写	对 CR 值的影响
LD，LDN	C	CR 值重新建立
逻辑指令，比较指令等	P	CR 值被更新为操作结果

ST，R，S，JMP，JMPCN，JMPC 等	U	CR 值保持不变

注意：在 IEC61131－3 中并没有完善地定义各种操作符对于 CR 的影响，因此在不同的编程系统中这些定义可能有所不同。

4.4.2.3 网络

在 IL 程序中也存在着网络（Network）的概念，以网络作为基本的段落，一个 POU 的代码部分就是由若干个网络组成。

一个典型的网络由网络标号和网络中的语句组成。在 EuraProg 中，关于网络的格式有如下规定：

在一个网络中可以只有一个语句标号。例如：

```
(* NETWORK 0 *)
AAA:            (*  可以只有一个标号  *)
```

在一个网络中可以只有程序语句。

在 4.4.2.2 关于 CR 中，我们将所有指令分为了三组：“C”、“P”、“U”。

网络中的程序必须以“C”组中的指令开始，以“P”、“U”组中的指令结束。

举例如下：

```
(* NETWORK 0 *)
LD   %M3.3      (*  以 LD 指令开始  *)
ST   %Q1.3      (*  以允许的指令结束  *)
```

在一个网络中可以有语句标号和程序语句。

网络中的程序必须以语句标号或者“C”组中的指令开始，以“P”、“U”组中的指令结束。

举例如下：

```
(* NETWORK 0 *)
AAA:            (*  以语句标号开始  *)
LD %M3.3
ST %Q1.3        (*  以允许的指令结束  *)
```

4.4.3 EuraProg 中的 IL 编辑器

当使用 IL 语言新建一个新程序时，就将进入 IL 编辑器；若打开一个用 IL 编写的程序，也将进入 IL 编辑器。IL 编辑器的外观如下图。

图 4-2 IL 编辑器

一个 POU 由两部分组成：参数、变量声明部分；代码部分。因此，编辑器相应地也分为两部分区域：

- 变量声明表格：用于声明该 POU 的输入/输入参数和局部变量。支持右键菜单。
- 程序编辑区：这个区域类似于一个文本编辑器，用户可以在此直接编辑自己的应用程序。

4.4.3.1 IL 程序编辑

♦ **添加一个网络**

可以使用如下任何一种方法来加入一个新网络：

- 使用 **Ctrl+Q** 快捷键；
- 在程序编辑区内单击鼠标右键，执行弹出菜单命令**【加入新网络】**。

♦ **输入 IL 语句**

程序编辑区类似于一个文本编辑器，用户可以在此直接输入语句，编辑自己的应用程序。这里支持通常的键盘操作，比如删除（**Delete**），退格（**BackSpace**），使用上、下、左、右方向键移动光标等。

IL 编辑器能够自动地对用户输入的语句进行格式化，并将语句中的关键字用蓝色来显示，注释采用绿色来显示。

在编辑语句时，如果光标切换到其它行，IL 编辑器就会对刚离开的行进行语法检查，如果有语法错误，则会在该行的开头显示一个红色的问号“？”。只有检查正确的语句才会格式化显示。

♦ **选中/删除/复制/剪切/粘贴**

在 IL 编辑器中可以使用**【编辑】**菜单中的所有编辑命令，包括全选、复制、剪切、粘贴等。另外，在程序编辑区中单击鼠标右键将会弹出右键菜单，用户也可以使用右键菜单命令进行编辑。

执行【**全选**】命令将会选中编辑区内所有的文本内容。使用鼠标在编辑区中拖动，或者使用上、下、左、右方向键移动光标同时按下 SHIFT 键，将会选中所经过的文本内容。选中的区域将采用黑色作为底色，文字呈高亮状态显示。

使用 **Delete** 键或者执行【**删除**】命令，可以删除选中的内容。

使用快捷键 **Ctrl+C** 或者执行【**复制**】命令，可以将选中的内容复制到 Windows 的剪贴板中。被复制的内容可以在任何文本编辑器中粘贴，比如 Windows 记事本、Word 等。

剪切的操作相当于复制并删除选中的内容。使用快捷键 **Ctrl+X** 或者执行【**剪切**】命令，可以将选中的内容剪切到 Windows 剪贴板中。

复制或者剪切完毕后，将光标定位到想要粘贴的位置，然后使用快捷键 **Ctrl+V** 或者执行【**粘贴**】命令，即可将剪贴板中的内容粘贴到光标所在的位置。

♦ **查找/替换**

IL 编辑器提供了标准的查找和替换命令。

- 查找

使用快捷键 **Ctrl+F** 或者执行【**编辑**】→【**查找...**】菜单命令，将会弹出如下的查找窗口：

图 4-3　查找对话框

在【**查找内容**】输入框中输入要查找的字符串，然后单击【**查找下一个**】按钮即可开始查找，找到的内容会被选中并高亮显示。

其它选项均采用了 Windows 标准的查找窗口中的含义，此处不再赘述。

- 替换

使用快捷键 **Ctrl+R** 或者执行【**编辑**】→【**替换...**】菜单命令，将会弹出如下的替换窗口：

图 4-4　替换对话框

在【**查找内容**】输入框中输入要查找的字符串，在【**替换为**】输入框中输入要替换的字符串，然后单击【**替换**】按钮，IL 编辑器将自动查找下一个符合查找内容的字符串并将该字符串替换。单击【**全部替换**】按钮，IL 编辑器将自动查找当前程序中所有符合查找内容的字符串并全部替换为指定的字符串。

其它选项均采用了 Windows 标准的查找窗口中的含义，此处不再赘述。

4.4.4　IL 程序转换为 LD 程序

执行【**查看**】→【**LD 编辑器**】菜单命令可以将当前 POU 的语言切换为 LD。

并非所有的 IL 程序都可以转换为 LD 格式，IL 程序要转换成 LD 程序须满足如下条件：

① IL 程序本身没有任何错误；

② IL 程序中的各个网络必须严格符合如下规则，包括：

✓ 若网络中包含有语句标号，则只允许有一个语句标号，除此之外不能有任何其它指令。

✓ 若网络中不包含语句标号，则需满足如下条件：

- 网络必须以“C”组指令（LD 类指令）开始；
- 作为网络开始行的“C”组指令（LD 类指令）在一个网络中只能出现一次；
- 网络中的程序必须以“P”、“U”组指令结束。

4.4.5　调试和监控程序

4.4.5.1　在线监控 IL

若当前窗口是 IL 编辑器，则可以使用如下任一方法进入在线监控状态。

- 执行【**调试**】→【**在线监控**】菜单命令。

- 单击工具栏上的  图标。
- 使用 F6 快捷键。

在线监控状态下，在原程序编辑区中将分为两栏，右边显示程序，左边显示相应的变量值，中间以一条竖线分割开。将光标置于竖线上，待其形状变为  后，即可拖动该线改变左、右区域的大小。

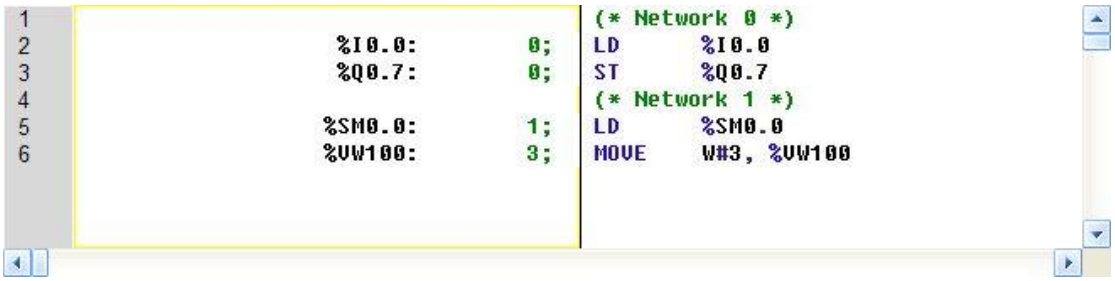
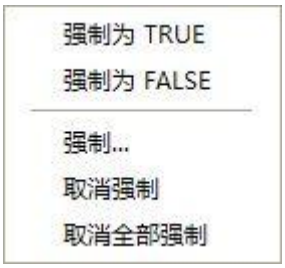


图 4-5 IL 编辑器的在线监控

注意：在线监控状态下不允许对程序进行编辑。

4.4.5.2 强制指定变量

在线监控 IL 程序时，用户可以在 IL 编辑器中直接对指定变量进行强制、取消强制等操作：在程序中某个变量上单击鼠标右键，将弹出如下右键菜单（注：若右键单击的是非开关量变量，则菜单中的**【强制为 TRUE】**和**【强制为 FALSE】**命令将是无效的状态）。



- **强制为 TRUE**：将该变量（开关量）的值强制为 1（TRUE）。
- **强制为 FALSE**：将该变量（开关量）的值强制为 0（FALSE）。
- **强制...**：执行该命令后，将弹出如下对话框：

图 4-6 强制变量对话框

在**【强制值】**输入框中输入要强制的值（相应数据类型的常量），然后单击**【强制】**按钮

即可。

关于常量的表示格式请参阅 4.2.3 常量 。

- **取消强制**：取消对该变量的强制。
- **全部取消强制**：取消对所连 CPU 中当前所有变量的强制。

4.5 LD 编辑器

本节对 EuraProg 中 LD 编辑器的功能、使用进行了详细的描述，同时也阐述了 IEC 61131-3 标准中关于 LD 语言的相关语法、规定。通过阅读本节，用户可以轻松地使用 EuraProg 编写出符合 IEC 标准的应用程序。

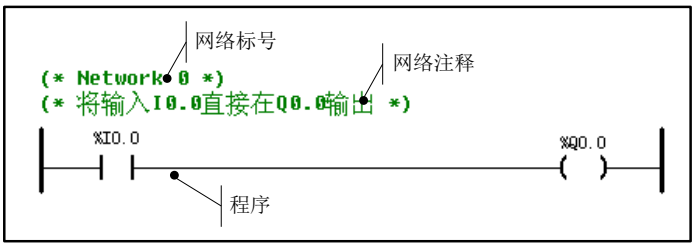
4.5.1 LD 的背景

LD（梯形图）语言是 IEC 61131-3 标准中规定的五种编程语言之一，是 PLC 编程中被最广泛使用的一种图形化语言。

LD 语言源于机电一体化领域中图形表示的继电器逻辑，用它编写的程序能够与实际的电气操作原理图相对应，非常直观，易于学习和掌握。LD 语言的长处在于布尔逻辑的处理。

4.5.2 网络

在 LD 中也使用“网络（Network）”的概念，以网络作为基本的段落。一个典型的网络由网络标号、注释和网络中的程序这三部分组成。如下图。



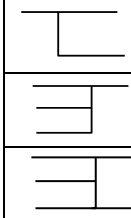
一个 LD 网络的边界是位于左、右两侧的电源线（Power Rails）。左侧电源线的状态一直为“1”，右侧电源线的状态没有定义。我们可以对一个 LD 网络作如下形象的描述：电能（能量流）自左侧的电源线沿着连接线流经线上所有的元件（包括触点、线圈、功能、功能块等），这些元件依据自身的逻辑状态，或是中断能量流，或是将电能传输到后继的元件并最终到达右侧的电源线。

4.5.3 标准化图形对象

- **连接**

LD 中使用水平连接线和垂直连接线，分别对应着串联和并联的关系。下表提到的连接线的值或者连接线某侧的值也可以理解为是否有能量流存在。

表 4-9 LD 中的连接

图形对象	名 称	描 述
———	水平连接	可以将其理解为一根理想的导线。 水平连接将线上任一点左侧的值传递到右侧。
	垂直连接（含有水平连接）	垂直连接将它左侧所有水平连接的值首先进行逻辑“或”运算，然后再将运算结果传递到它右侧所有的水平连接上。

- **触点**

触点有两种类型：常开触点和常闭触点。

每个触点都必须对应一个有效的布尔型变量，变量名称被写在图形元素的上面，该变量的值决定了触点的状态（闭合或者断开），并进而决定了触点所在线路的通或者断。

表 4-10 LD 中的接点

图形对象	名 称	描 述
变量名 —┤ ├—	常开接点	若变量值为 TRUE，则触点闭合，它左侧连接的值被传递到右侧连接；否则触点断开，它右侧连接的值为 FALSE。
变量名 —┤/├—	常闭接点	若变量的值为 TRUE，则触点断开，它右侧连接的值为 FALSE； 否则触点闭合，它左侧连接的值被传递到右侧连接。

- **线圈**

线圈是用于给布尔型变量赋值的图形元素。

表 4-11 LD 中的线圈

图形对象	名 称	描 述
变量名 —( )—	线圈	将左侧连接的值传递给变量。
变量名 —(/)—	取反线圈	将左侧连接的值取反然后传递给变量。
变量名 —(S)—	置位线圈	若左侧连接的值为 TRUE，则变量值被置为 TRUE； 若左侧连接的值为 FALSE，则变量值保持不变。
变量名 —(R)—	复位线圈	若左侧连接的值为 TRUE，则变量值被置为 FALSE； 若左侧连接的值为 FALSE，则变量值保持不变。

♦ **调用功能和功能块**

LD 支持对功能和功能块的调用。被调用的功能和功能块以矩形框来表示，其形参显示在矩形框内，实参显示在矩形框外部的连接线上。功能或者功能块的参数中至少有一个 BOOL 型的输入参数和一个 BOOL 型的输出参数，用于将它接至连接线上。

功能必须有一个名为 EN 的 BOOL 型输入和一个名为 ENO 的 BOOL 型输出，用于控制该功能的执行。若 EN 的值为 1，则该功能被执行且 ENO 也将被置为 1；若 EN 的值为 0，则不执行该功能且 ENO 也将被置为 0。

4.5.4 EuraProg 中的 LD 编辑器

当使用 LD 语言新建一个新程序时，就将进入 LD 编辑器；若打开一个用 LD 编写的程序，也将进入 LD 编辑器。LD 编辑器的外观如下图。

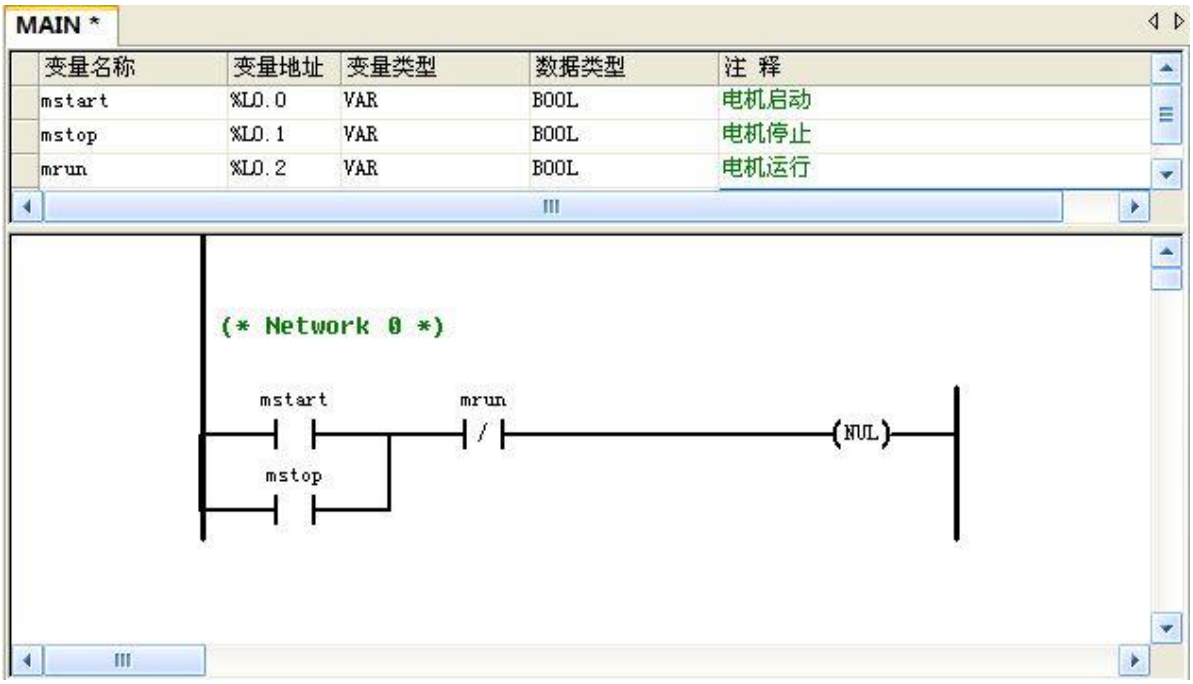


图 4-7 LD 编辑器

一个 POU 由两部分组成：参数、变量声明部分；代码部分。因此，编辑器相应地也分为两部分区域：

- 变量声明表格：用于声明该 POU 的输入/输入参数和局部变量，支持右键菜单。
- 程序编辑区：这个区域类似于一块画布，用户可以在这里输入各种 LD 图形元件。

4.5.4.1 LD 程序的限制

程序编辑区被划分为许多个单元格，用户可以通过鼠标单击或使用键盘的方向键来选择单元格，选中的单元格会被虚线矩形框包围以提示当前位置。各种 LD 元件根据自身大小不同，分别占用一个或多个单元格。由于画布的大小限制，因此在每一个 LD 程序中也有如下限制：

- 最多允许 200 个网络；

- 每条连接线路径上串联的元件数量限制：对于线圈、触点，则不超过 35 个触点加 1 个线圈；若只有功能/功能快，则建议不超过 12 个块加 1 个线圈加 1 个触点；
- 一个并联关系不允许超过 16 个分支。

4.5.4.2 LD 程序编辑

♦ **标记元件**

鼠标左键单击某个元件可以选中并标记该元件。被标记的元件会被虚线矩形框所包围，表示该元件获得了当前的焦点。另外，也可以使用键盘上的方向键来移动焦点，从而改变标记的元件。

被标记的元件将作为基准，用户添加新元件需要根据这个基准位置来进行。

另外，用户可以对被标记的元件进行各种编辑操作，比如修改属性、删除、复制、剪切等。

♦ **添加一个网络**

✓ 在期望添加的位置上标记一个元件作为基准。

若标记的是网络注释，那么将在标记元件所在网络的上方加入新网络；若标记的是其它元件，那么将在标记元件所在网络的下方加入新网络。

若当前是一个没有任何元件的空程序，那么无需标记元件，将在编辑区的最上部加入新网络。

✓ 使用如下任一方法在程序中加入一个新网络：

- 单击工具栏上的  图标；
- 使用 **F2** 快捷键；
- 鼠标右键单击任何一个元件，执行右键菜单中的**【新网络】**命令。

新网络加入后如同下图的形式：

♦ **输入注释**

双击网络标号部分，即可弹出“注释”对话框，如下图所示。

图 4-8　注释

用户可以在左侧的输入框中输入该网络的注释，然后单击【**确认**】按钮即可。

- **添加一个串联触点**
  - ✓ 在期望添加的位置上标记一个元件作为基准。
  - ✓ 使用如下任一方法在程序中加入一个串联触点：
    - 单击工具栏上的  图标（左触点）或者  图标（右触点）；
    - 使用 **Ctrl+L**（左触点）或者 **Ctrl+E**（右触点）快捷键；
    - 鼠标右键单击基准元件，执行右键菜单中的【**左触点**】或者【**右触点**】命令；
    - 鼠标双击窗口左侧“**指令集**”中所需要的触点指令。
- **添加一个并联触点**
  - ✓ 在期望添加的位置上标记一个触点元件作为基准。
  - ✓ 使用如下任一方法在程序中加入一个并联触点：
    - 单击工具栏上的  图标；
    - 使用 **F3** 快捷键；
    - 鼠标右键单击基准元件，执行右键菜单中的【**并联触点**】命令。
- **添加一个并联线圈**
  - ✓ 在期望添加的位置上标记一个线圈元件作为基准。
  - ✓ 使用如下任一方法在程序中加入一个并联线圈：
    - 单击工具栏上的  图标；
    - 使用 **F12** 快捷键；
    - 鼠标右键单击基准元件，执行右键菜单中的【**并联线圈**】命令；
    - 鼠标双击窗口左侧“**指令集**”中所需要的线圈指令。
- **添加一个串联的功能/功能块**
  - ✓ 在期望添加的位置上标记一个元件作为基准。
  - ✓ 使用如下任一方法在程序中加入一个串联的功能/功能块：
    - 单击工具栏上的  图标；
    - 使用 **F11** 快捷键；
    - 鼠标右键单击基准元件，执行右键菜单中的【**功能/功能块**】命令。

然后都会弹出如下的“功能/功能块/子程序”窗口：

图 4-9　建立功能块

用户选好所需的【**功能/功能块**】或者【**子程序**】，然后单击【**确认**】按钮即可。

- 鼠标双击窗口左侧“**指令集**”中所需要的功能、功能块指令或者子程序。

♦ **建立一个并联支路**

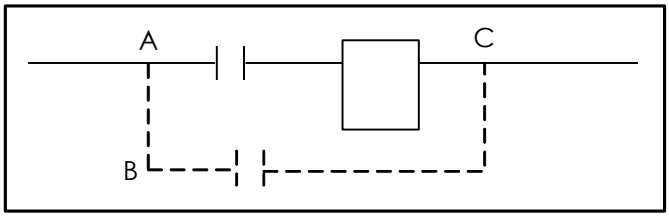
LD 编辑器提供了并联支路编辑模式，从而让用户能够快速创建一个复杂的并联支路。进入并联支路编辑模式后，鼠标指针形状将变为，另外，用户可以随时使用 ESC 键来退出该模式。

✓用户建立一个并联支路的步骤如下：

- 单击工具栏上的图标；
- 使用 **Ctrl+H** 快捷键；
- 鼠标右键单击一个元件，执行右键菜单中的【**并联支路**】命令。

然后都会进入并联支路编辑模式。

✓使用鼠标继续按如下步骤来画一个并联支路：



- 在期望的并联起始位置处（如上图中 A）单击；
- 向上或者向下移动鼠标至任意位置（如 B）并单击；
- 移动鼠标至期望的并联结束位置处（如 C）并单击，完成一个并联支路的建立。

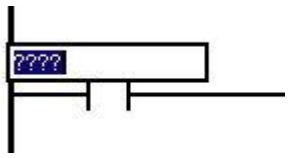
注：上面提到的三个位置并不需要太精确，用户在“大约”的位置处单击即可。

♦ **修改元件的参数**

加入一个元件后，它的实际参数以红色的问号（????）来表示，这些红色问号必须被修改为适当的常数、直接地址、全局变量或者局部变量名称。

用户可以采用如下任一方法来修改元件的参数：

✓鼠标单击将要修改的元件参数所在的位置，将会在此参数区域出现一个输入框，如下图。



用户直接在此输入框内输入期望的常数、直接地址、全局变量或者局部变量名称，然后按 **Enter** 键确认即可。LD 编辑器会自动对用户输入的直接地址进行格式化，因此用户在输入时可以不输入百分号（%）。在输入时，也可以按 **ESC** 键取消当前的修改。

✓标记要修改的元件，然后按 **Enter** 键，就会在被标记元件的第一个参数的区域出现一个输入框，然后按照上面任一方法进行输入即可。

✓鼠标双击将要修改的元件，将会弹出该元件的属性对话框

- 触点的属性对话框

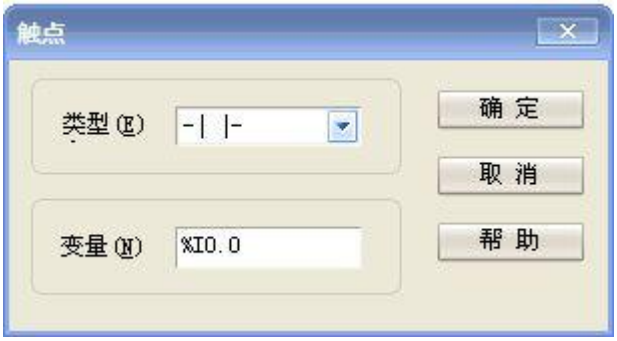


图 4-10　触点属性

用户可以修改该元件的【**类型**】、连接的【**变量**】，然后单击【**确定**】按钮即可。

- 线圈的属性对话框

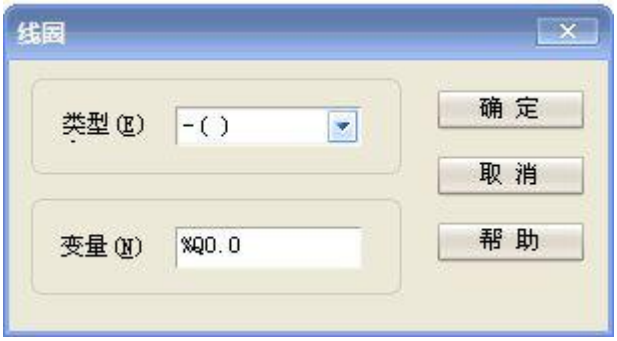


图 4-11　线圈属性

用户可以修改该元件的【**类型**】、连接的【**变量**】，然后单击【**确定**】按钮即可。

- 功能/功能块的属性对话框

图 4-12　功能块属性

双击【**参数**】列表中的某项参数，就会在该参数的【**变量连接**】处出现一个输入框，用户直接在此框内输入期望的常数、直接地址、全局变量或者局部变量名称，然后按 **Enter** 键确认即可。

用户也可以鼠标单击或者使用上、下方向键选择某项参数，然后按 **Enter** 键进入输入框修改该变量。

EuraProg 将对用户的输入进行严格的语法检查，不接受错误的输入，包括非法的变量、非法的数据类型、非法的内存区域类型等。

所有输入完成后，单击【**确定**】按钮即可。

- **删除一个元件**

✓标记期望删除的元件。

✓使用如下任一方法删除标记的元件：

- 单击工具栏上的  图标；
- 单击鼠标右键，执行右键菜单中的【**删除元件**】命令；
- 执行【**编辑**】→【**删除**】菜单命令；
- 使用 **Delete** 键。

- **删除一个网络**

✓在期望删除的网络中标记任何一个元件。

✓使用如下任一方法删除标记元件所在的网络：

- 单击工具栏上的  图标；
- 单击鼠标右键，执行右键菜单中的【**删除网络**】命令；
- 使用 **Ctrl+Del** 快捷键。

♦ **选中/删除/复制/剪切/粘贴**

在 LD 编辑器中可以使用【**编辑**】菜单中的编辑命令，包括全选、复制、剪切、粘贴等。另外，在程序编辑区中单击鼠标右键将会弹出右键菜单，用户也可以使用右键菜单命令进行编辑。

✓**选中** 执行【**全选**】命令将会选中编辑区内所有的内容。使用鼠标在编辑区中拖动，或者使用上、下、左、右方向键移动光标同时按下 SHIFT 键，将会选中所经过的内容。选中的区域将采用黑色作为底色，图形元件和文字呈高亮状态显示。

✓**删除** 使用 **Delete** 键或者执行【**删除**】命令，可以删除选中的内容。

✓**复制** 使用快捷键 **Ctrl+C** 或者执行【**复制**】命令，可以复制选中的内容。被复制的内容可以在 LD 编辑器中进行粘贴。

✓**剪切** 剪切的操作相当于复制并删除选中的内容。使用快捷键 **Ctrl+X** 或者执行【**剪切**】命令，可以剪切选中的内容。

✓**粘贴** 复制或者剪切完毕后，在要粘贴的位置标记一个元件，然后使用快捷键 **Ctrl+V** 或者执行【**粘贴**】命令，即可将复制的内容粘贴到相应的位置：若标记的是网络注释，那么将粘贴至标记元件所在网络的上方；若标记的是其它元件，那么将粘贴至标记元件所在网络的下方。

4.5.5 监控和调试程序

4.5.5.1 在线监控 LD 程序

注意：在线监控状态下不允许对程序进行编辑。

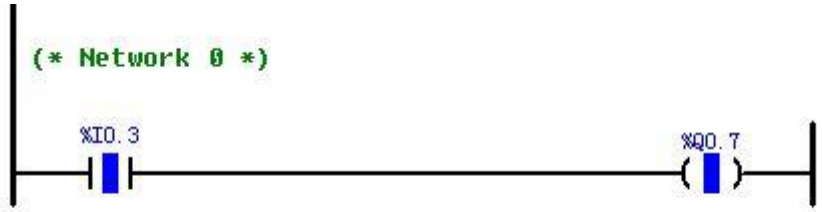
若当前窗口是 LD 编辑器，则可以使用如下任一方法进入在线监控状态：

- 执行【**调试**】→【**在线监控**】菜单命令；
- 单击工具栏上的  图标；
- 使用 **F6** 快捷键。

在线监控状态下，程序中的变量状态显示方式为：

- %I0.0 ┤ ├ 表示该触点断开，%I0.0 ┤▮├ 表示该触点闭合；
- %M1.0 ( ) 表示该线圈断开，%M1.0 (▮) 表示该线圈闭合；
- 程序中非开关量变量的值将直接在该变量的右侧显示出来。

下图是一个在线监控的程序示例：



4.5.5.2 强制指定变量

在线监控 LD 程序进行强制变量命令时， 与前面 4.4.5.2 强制指定变量类似。

第五章　PLC 指令集

EC100/EC200 支持 IEC 61131-3 标准的基本指令及其大部分功能/功能块，编程风格符合 IEC 61131-3 标准要求，同时，根据实际的需要，对标准指令作了适当的扩充，可以满足不同用户、多应用领域工程实际的需要。

5.1　综述

本章对指令集中的所有指令进行了详细的说明，同时对大多数指令也提供了具体的使用实例。对于指令的说明包括 LD、IL 两种格式。

在 LD 格式中，下文的描述没有具体提及能量流，能量流的含义是固定的，一个网络上各指令的状态控制着能量流在本网络上的流动。我们可以认为它是部分指令（无 EN、ENO 操作数的指令）的虚拟输入，并且其类型是 BOOL 型。为了便于描述，在下文中我们将能量流是否到达某点简称为在该点能量流的值为 1 或者 0。

另外，在 LD 格式中，下文对 EN、ENO 操作数和其数据类型也没有说明，因为它们均为 BOOL 型，且含义也是固定的：EN 的意思为“使能”，若某功能/功能块的 EN 值为 1，则该功能/功能块才被执行，能量流继续向前流动，在 ENO 端输出为 1。若 EN 值为 0，则该功能/功能块不被执行，在 ENO 端输出为 0。

在 4.4.2.2 关于 CR 中提到，在 IL 程序中，每一条指令执行完之后都会对 CR 值产生相应的影响。本章对每条 IL 指令对 CR 值的影响都作了说明，在说明时采用了在 4.4.2.2 关于 CR 中规定的“分组缩写”符号。

5.2　位指令

5.2.1　标准触点

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	常开触点	*bit* ⊣ ⊢	
	常闭触点	*bit* ⊣/⊢	
IL	LD	LD *bit*	C
	AND	AND *bit*	P
	OR	OR *bit*	
	LDN	LDN *bit*	C
	ANDN	ANDN *bit*	P
	ORN	ORN *bit*	

操作数	输入/输出	数据类型	允许的内存区
bit	输入	BOOL	I、Q、V、M、SM、L、T、C、RS、SR、常量

- **LD**

常开触点的作用是：若 *bit* 值为 1，则触点闭合，能量流继续向后传递；若 *bit* 值为 0，则触点断开，能量流也被切断。

常闭触点的作用是：若 *bit* 值为 0，则触点闭合，能量流继续向后传递；若 *bit* 值为 1，则触点断开，能量流也被切断。

- **IL**

常开触点由 *LD*、*AND*、*OR* 指令提供。

LD 指令将 *bit* 值装载于 CR 中并作为新的 CR 值；

AND 指令将 CR 值和 *bit* 值进行“与”运算，并将运算结果作为新的 CR 值；

OR 指令将 CR 值和 *bit* 值进行“或”运算，并将运算结果作为新的 CR 值。

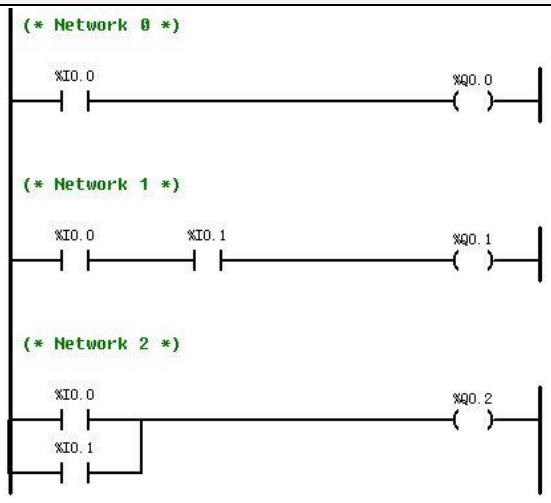
常闭触点由 *LDN*、*ANDN*、*ORN* 指令提供。

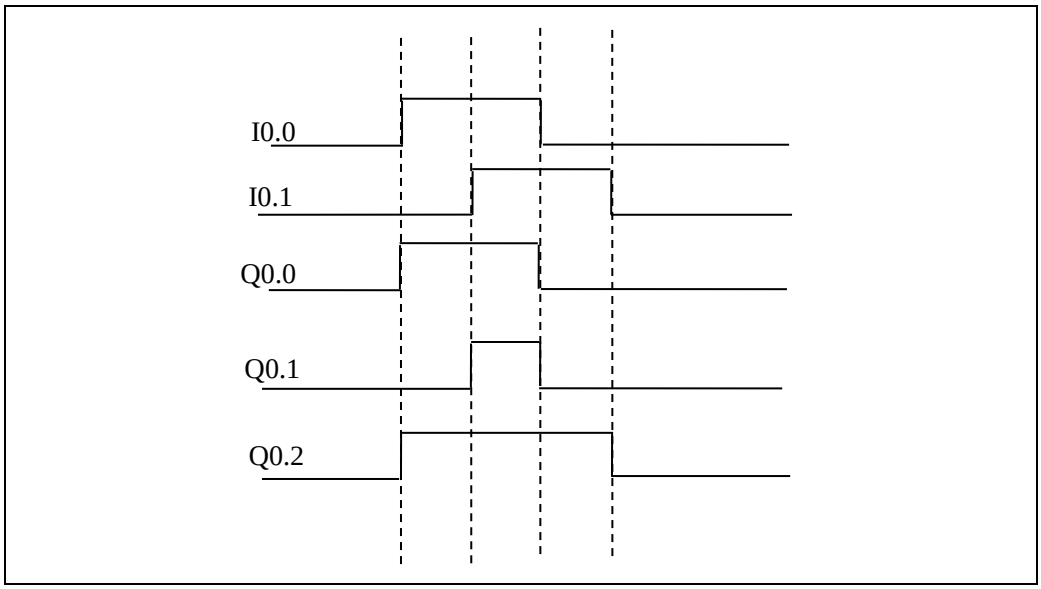
LDN 指令将 *bit* 值取反，然后将结果装载于 CR 中并作为新的 CR 值；

ANDN 指令将 *bit* 值取反，然后将结果和 CR 值进行“与”运算，并将运算结果作为新的 CR 值；

ORN 指令将 *bit* 值取反，然后将结果和 CR 值进行“或”运算，并将运算结果作为新的 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.0 ST %Q0.0 (* Network 1 *) LD %I0.0 AND %I0.1 ST %Q0.1 (* Network 2 *) LD %I0.0 OR %I0.1 ST %Q0.2
时序图	



5.2.2 立即触点

- 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	立即常开触点	*bit* I	
	立即常闭触点	*bit* /I	
IL	LDI	LDI *bit*	C
	ANDI	ANDI *bit*	P
	ORI	ORI *bit*	
	LDNI	LDNI *bit*	C
	ANDNI	ANDNI *bit*	P
	ORNI	ORNI *bit*	

操作数	输入/输出	数据类型	允许的内存区
bit	输入	BOOL	I（CPU 本体）

在指令执行时，立即触点直接读取 *bit* 的物理输入通道的实际状态，但是不更新输入映像区。

立即触点指令只能用于 CPU 本体的 DI 点。在用户工程的**【硬件配置】**中，CPU 模块的**【I/O 设置】→【输入滤波】**的设置对立即触点指令没有影响。

与普通触点指令相比，立即触点指令读取的不是输入映像区的数据，因此不会受到 CPU 扫描周期的影响，能够快速对输入信号做出响应。

- **LD**

立即常开触点的作用是：若 *bit* 的物理输入通道的实际状态值为 1，则触点闭合，能量流继续向后传递，否则触点断开，能量流也被切断。

立即常闭触点的作用是：若 *bit* 的物理输入通道的实际状态值为 0，则触点闭合，能量流继续向后传递，否则触点断开，能量流也被切断。

- **IL**

立即常开触点由 *LDI*、*ANDI*、*ORI* 指令提供。

LDI 指令将 *bit* 的物理输入通道的实际状态值装载于 CR 中并作为新的 CR 值；

ANDI 指令将 CR 值和 *bit* 的物理输入通道的实际状态值进行“与”运算，并将运算结果作为新的 CR 值；

ORI 指令将 CR 值和 *bit* 的物理输入通道的实际状态值进行“或”运算，并将运算结果作为新的 CR 值。

立即常闭触点由 *LDNI*、*ANDNI*、*ORNI* 指令提供。

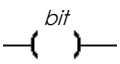
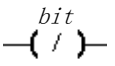
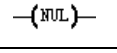
LDNI 指令将 *bit* 的物理输入通道的实际状态值取反，然后将结果装载于 CR 中并作为新的 CR 值；

ANDNI 指令将 *bit* 的物理输入通道的实际状态值取反，然后将结果和 CR 值进行“与”运算，并将运算结果作为新的 CR 值；

ORNI 指令将 *bit* 的物理输入通道的实际状态值取反，然后将结果和 CR 值进行“或”运算，并将运算结果作为新的 CR 值。

5.2.3 普通输出

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	线圈		
	取反线圈		
	空线圈		
IL	ST	ST *bit*	U
	STN	STN *bit*	

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q、V、M、SM、L

- **LD**

线圈的作用是：将线圈左侧能量流的值赋给 *bit* 。

取反线圈的作用是：将线圈左侧能量流的值取反并赋给 *bit* 。

空线圈的作用是：在 LD 程序中指示一个网络的结束。该指令仅为方便用户的编程，并不执行具体的操作。

- **IL**

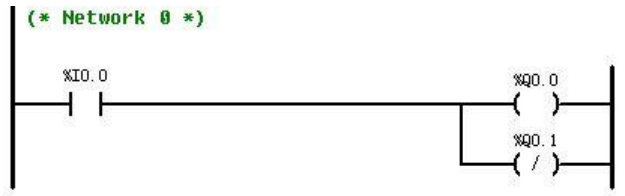
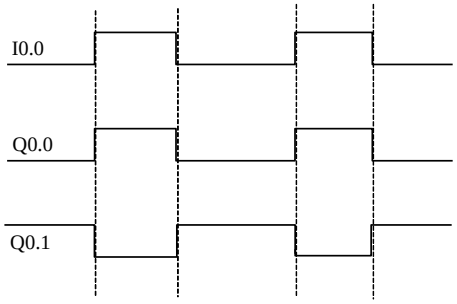
  线圈由 *ST* 指令提供，取反线圈由 STN 指令提供。

  ST 指令用于将 CR 值赋给 *bit* 。

  STN 指令用于将 CR 值取反并赋给 *bit* 。

  ST、*STN* 指令的执行对 CR 值无影响。

➢ 指令使用举例

LD	**IL**
	(* Network 0 *) LD %I0.0 ST %Q0.0 STN %Q0.1
时序图	
	

5.2.4 立即输出

➢ 指令及其操作数说明

	名 称	**指令格式**	**影响 CR 值**
LD	立即线圈		
IL	STI	STI *bit*	U

操作数	**输入/输出**	**数据类型**	**允许的内存区**
bit	输出	BOOL	Q（CPU 本体）

立即输出指令只能用于 CPU 本体的 DO 点。

- **LD**

在指令执行时，立即线圈左侧能量流的值被直接写入 *bit* 的物理输出点进行输出，同时更新相应的输出映像区。

- **IL**

立即输出由 *STI* 指令提供。

STI 指令将 CR 值直接写入 *bit* 的物理输出点进行输出，同时更新相应的输出映像区。*STI* 指令的执行对 CR 值无影响。

5.2.5 置位与复位

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	复位	*bit* —(R)—	
	置位	*bit* —(S)—	
IL	复位	R *bit*	U
	置位	S *bit*	

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q、V、M、SM、L

- **LD**

复位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 值被置为 0，否则 *bit* 值保持不变。

置位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 值被置为 1，否则 *bit* 值保持不变。

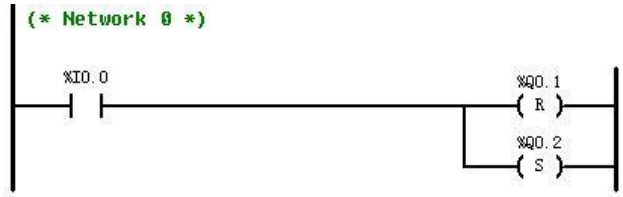
- **IL**

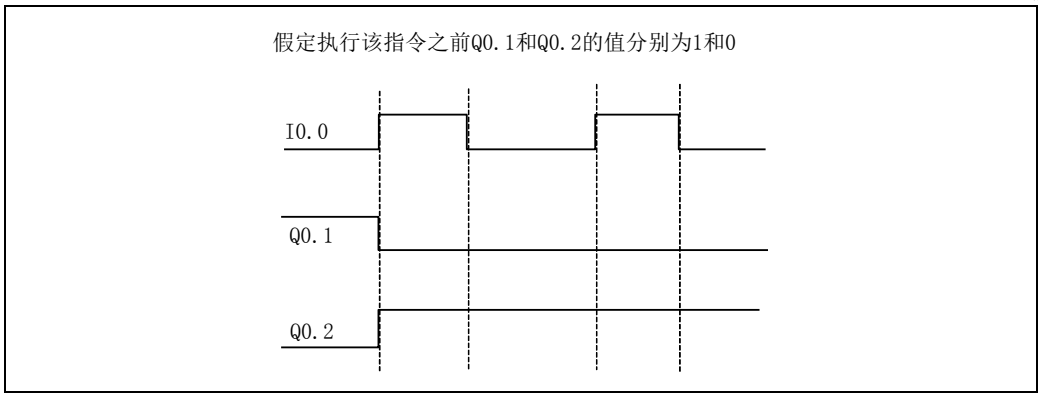
复位指令的作用是：若 CR 值为 1，则 *bit* 值被置为 0，否则 *bit* 值保持不变。

置位指令的作用是：若 CR 值为 1，则 *bit* 值被置为 1，否则 *bit* 值保持不变。

R、S 指令的执行不影响 CR 值。

指令使用举例：

LD	**IL**
	(* Network 0 *) LD %I0.0 R %Q0.1 S %Q0.2
时序图	



5.2.6 立即置位与立即复位

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	立即复位	bit —(RI)—	
	立即置位	bit —(SI)—	
IL	立即复位	RI *bit*	U
	立即置位	SI *bit*	

操作数	输入/输出	数据类型	允许的内存区
bit	输出	BOOL	Q（CPU 本体）

立即置位与立即复位指令只能用于 CPU 本体的 DO 点。

- **LD**

立即复位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 对应的输出映像寄存器和物理输出点均被置为 0，否则这两者的值保持不变。

立即置位线圈的作用是：若线圈左侧能量流的值为 1，则 *bit* 对应的输出映像寄存器和物理输出点均被置为 1，否则这两者的值保持不变。

- **IL**

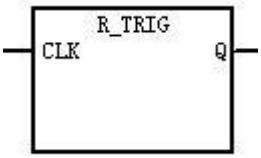
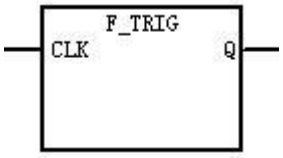
立即复位指令的作用是：若 CR 值为 1，则 *bit* 对应的输出映像寄存器和物理输出点均被置为 0，否则这两者的值保持不变。

立即置位指令的作用是：若 CR 值为 1，则 bit 对应的输出映像寄存器和物理输出点均被置为 1，否则这两者的值保持不变。

RI、SI 指令的执行不影响 CR 值。

5.2.7 边沿检测

➢ 指令及其参数说明

	名 称	指令格式	影响 CR 值
LD	上升沿检测		
	下降沿检测		
IL	上升沿检测	R_TRIG	P
	下降沿检测	F_TRIG	

参数	输入/输出	数据类型	允许的内存区
CLK（LD）	输入	BOOL	能量流
Q（LD）	输出	BOOL	能量流

- **LD**

R_TRIG 用于检测 *CLK* 输入的上升沿跳变：如果 *CLK* 值产生了由 0 到 1 的跳变，则 *Q* 输出为 1 并保持一个扫描周期，然后 *Q* 将输出为 0。

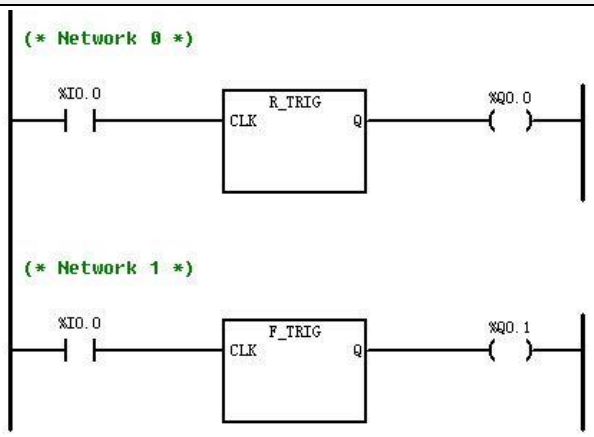
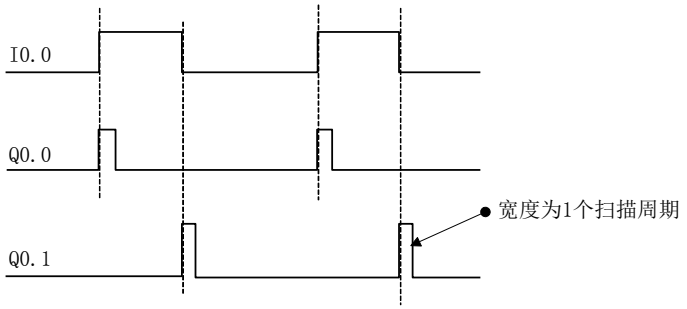
F_TRIG 用于检测 *CLK* 输入的下降沿跳变：如果 *CLK* 值产生了由 1 到 0 的跳变，则 *Q* 输出为 1 并保持一个扫描周期，然后 *Q* 将输出为 0。

- **IL**

R_TRIG 用于检测当前点 CR 值的上升沿跳变：如果当前点的 CR 值产生了由 0 到 1 的跳变，则立即将 CR 值设置为 1，否则将 CR 值设置为 0。

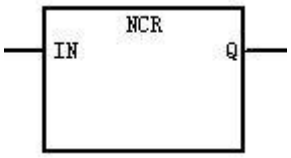
F_TRIG 用于检测当前点 CR 值的下降沿跳变：如果当前点的 CR 值产生了由 1 到 0 的跳变，则立即将 CR 值设置为 1，否则将 CR 值设置为 0。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.0 R_TRIG ST %Q0.0 (* Network 1 *) LD %I0.0 F_TRIG ST %Q0.1
时序图	
	

5.2.8 NCR（取反）

➢ 指令及其参数说明

	名 称	指令格式	影响 CR 值
LD	取反		
IL	取反	NCR	P

参数	输入/输出	数据类型	允许使用的内存区
IN	输入	BOOL	能量流
Q	输出	BOOL	能量流

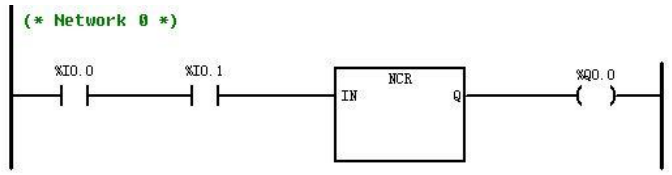
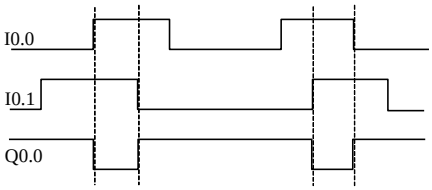
- **LD**

该指令用于改变能量流的状态：将输入端能量流的值取反并输出。

- **IL**

该指令用于改变 CR 值：将 CR 值取反，并将结果作为新的 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.0 AND %I0.1 NCR ST %Q0.0
时序图	
	

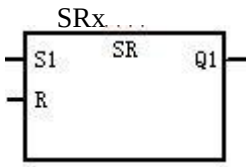
5.2.9 双稳态触发器

双稳态触发器是 IEC 61131-3 标准中定义的功能块之一，共有 SR 触发器（置位优先触发器）和 RS 触发器（复位优先触发器）两种。

关于功能块及其实例的使用请参阅 4.3.5 关于功能块以及功能块实例中的描述。

5.2.9.1 SR（置位优先触发器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	置位优先触发器		
IL	置位优先触发器	LD *S1* SR *SRx, R*	P

参数	输入/输出	数据类型	允许使用的内存区
SRx	-	SR 触发器实例	SR
S1	输入	BOOL	能量流
R	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
OUT	输出	BOOL	能量流

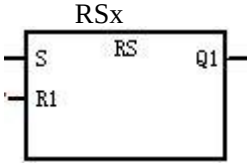
SR 触发器是置位端输入（*S1*）优先的触发器：若置位端输入（*S1*）和复位端输入（*R*）均为 1 时，则实例 *SRx* 的输出（*OUT*）及其状态值均为 1。

SR 触发器的真值表如下：

S1	R	OUT 及 SRx 状态值
0	0	保持前一状态
0	1	0
1	0	1
1	1	1

5.2.9.2 RS（复位优先触发器）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	复位优先触发器	RSx 	
IL	复位优先触发器	LD *S* RS *RSx, R1*	P

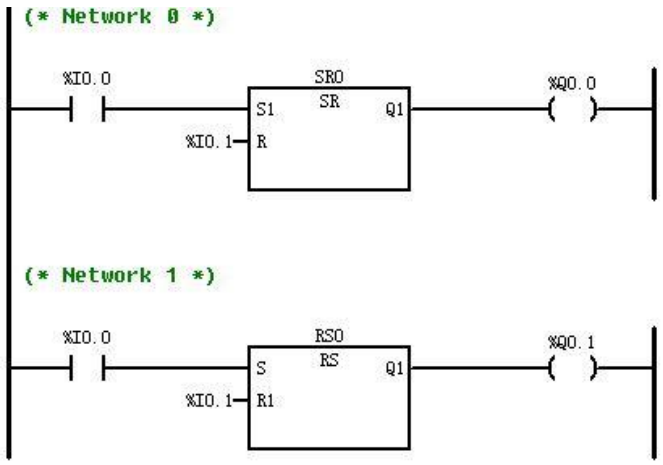
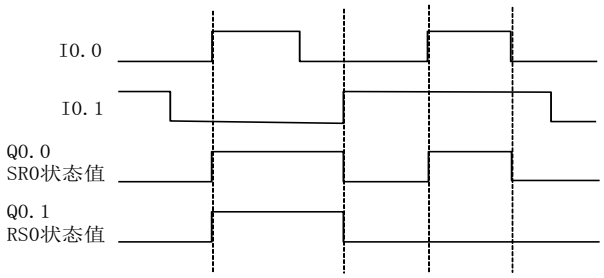
参数	输入/输出	数据类型	允许使用的内存区
RSx	-	RS 触发器实例	RS
S	输入	BOOL	能量流
R1	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
OUT	输出	BOOL	能量流

RS 触发器是复位端输入（*R1*）优先的触发器：若置位端输入（*S*）和复位端输入（*R1*）均为 1 时，则实例 *RSx* 的输出（*OUT*）及其状态值均为 0。

RS 触发器的真值表如下：

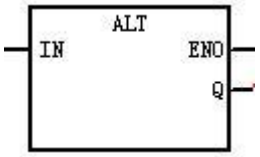
R1	S	OUT 及 RSx 状态值
0	0	保持前一状态
0	1	1
1	0	0
1	1	0

5.2.9.3 RS、SR 使用举例

LD	IL
	(* Network 0 *) LD %I0.0 SR SR0, %I0.1 ST %Q0.0 (* Network 1 *) LD %I0.0 RS RS0, %I0.1 ST %Q0.1
时序图	
	

5.2.10 ALT（反转）

➢ 指令及其参数说明

	名 称	指令格式	影响 CR 值
LD	取反		
IL	取反	ALT *Q*	U

参数	输入/输出	数据类型	允许的内存区
IN（LD）	输入	BOOL	能量流
Q	输出	BOOL	Q、V、M、SM、L

- **LD**

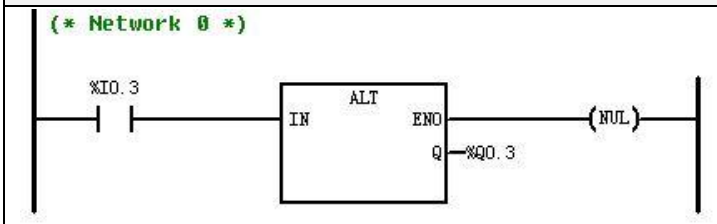
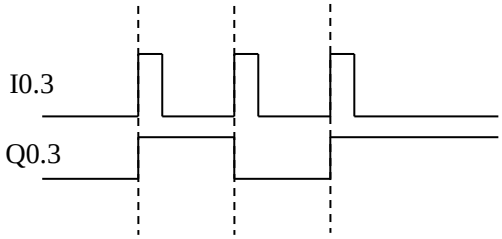
  如果输入 *IN* 产生了上升沿跳变，则 *Q* 立即被取反，否则 *Q* 保持不变。

- **IL**

  如果当前点的 CR 值产生了上升沿跳变，则 *Q* 立即被取反，否则 *Q* 保持不变。

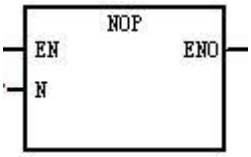
  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.3 ALT %Q0.3
时序图	
	

5.2.11 NOP（空操作）

➢ 指令及其参数说明

	名 称	指令格式	影响 CR 值
LD	空操作		
IL	空操作	NOP *N*	U

操作数	输入/输出	数据类型	允许的内存区
N	输入	INT	常数（正数）

NOP 指令仅仅让 CPU 产生 *N* 次空操作，不会影响用户程序的执行结果。参数 *N* 指定了执行空操作的次数，它必须是一个大于 0 的 INT 型常数。

5.2.12 括号修饰符

- 指令及其参数说明

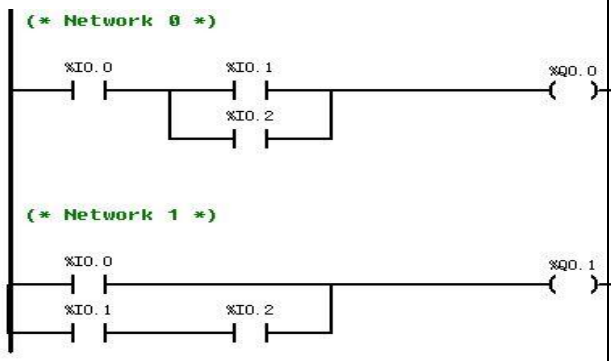
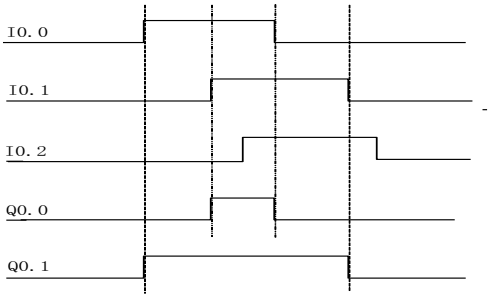
	名 称	指令格式	影响 CR 值
IL	AND(	AND(	U
	OR(	OR(	
	)	)	P

括号修饰符只有在 IL 中提供。在 IL 语言中只有简单的表达式，而不可能象在 LD、ST 等语言中那样采用复杂的表达式作为操作数，因此 IEC 61131-3 标准中定义了括号来处理一些复杂的表达式。“AND(”或者“OR(”都是和“)”配对使用的。

在 IL 程序中，执行“AND(”和“)”之间的指令之前，先将 CR 值暂存，再执行括号中的指令，执行完之后将结果与刚才暂存的 CR 值进行“与”运算，并将运算结果作为新的 CR 值。

类似地，执行“OR(”和“)”之间的指令之前，首先将 CR 值暂存，再执行括号中的指令，执行完之后将结果与刚才暂存的 CR 值进行“或”运算，并将运算结果作为新的 CR 值。

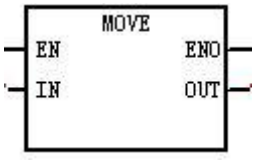
指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.0 AND(LD %I0.1 OR %I0.2) ST %Q0.0 (* Network 1 *) LD %I0.0 OR(LD %I0.1 AND %I0.2) ST %Q0.1
时序图	
	

5.3 赋值指令

5.3.1 MOVE（赋值）

- 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	MOVE		
IL	MOVE	MOVE *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L、SM、AQ、指针

该指令执行赋值操作，其参数 *IN* 与 *OUT* 的数据类型必须一致。

- **LD**

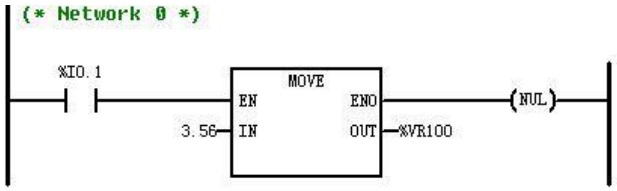
  如果 *EN* 为 1，则该指令将输入变量 *IN* 的值赋给输出变量 *OUT*。

- **IL**

  如果 CR 值为 1，则该指令将输入变量 *IN* 的值赋给输出变量 *OUT*。

  该指令的执行对 CR 值无影响。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.1 (* 建立 CR，其值为 I0.1 的值 *) MOVE 3.56, %VR100 (* 若 CR 为 1：VR100 被赋赋为 3.56；若 CR 为 0：不执行 MOVE 指令，VR100 的值不变 *)

5.3.2 BLKMOVE（块传送）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	BLKMOVE		
IL	BLKMOVE	BLKMOVE *IN, OUT,N*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC
N	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L、SM、AQ

参数 *IN*、*OUT* 的数据类型必须一致。该指令用于将从地址 *IN* 开始的连续 *N* 个变量的值传送给从地址 *OUT* 开始的连续 *N* 个变量，这些变量的数据类型与 *IN*、*OUT* 一致。

- **LD**

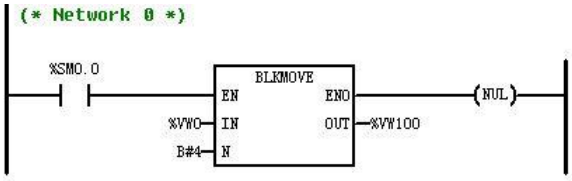
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。该指令的执行对 CR 值无影响。

指令使用举例：

LD



IL

```
(* Network 0 *)
LD          %SM0.0  (*  建立 CR，其值为 1  *)
BLKMOVE %VW0, %VW100, B#4  (*  VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中  *)
```

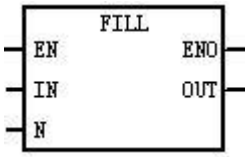
若 *BLKMOVE* 被执行，则结果如下：

VW0	VW2	VW4	VW6
0	10	20	30

VW100	VW102	VW104	VW106
0	10	20	30

5.3.3 FILL（块赋值）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	FILL		
IL	FILL	FILL *IN*, *OUT*, *N*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	常量
N	输入	BYTE	常量
OUT	输出	BYTE	M、V、L

该指令用于将从地址 *OUT* 开始的连续 *N* 个字节变量赋值为常数 *IN*。

- **LD**

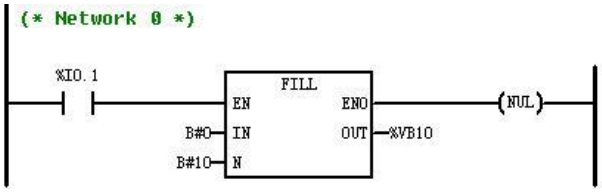
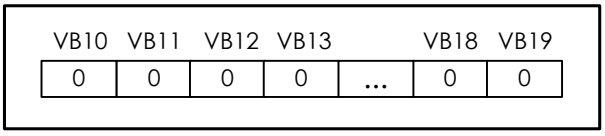
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

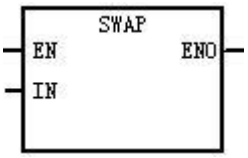
  该指令的执行对 CR 值无影响。

指令使用举例：

LD	**IL**
	(* Network 0 *) LD %I0.1 (* 建立 CR，其值为 I0.1 的值 *) FILL B#0, %VB10, B#10 (* 若 CR 为 1：VB10 至 VB19 总计 10 个变量均被赋值为 B#0；若 CR 为 0：不执行 FILL 指令*)
若 *FILL* 被执行，则结果如下：	
	

5.3.4 SWAP（交换）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	SWAP		
IL	SWAP	SWAP *IN*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入/输出	WORD、DWORD	Q、M、V、L、SM

若参数 *IN* 是 WORD 型，则该指令用于交换 *IN* 的高字节与低字节；若参数 *IN* 是 DWORD 型，则该指令用于交换 *IN* 的高字与低字。

- **LD**

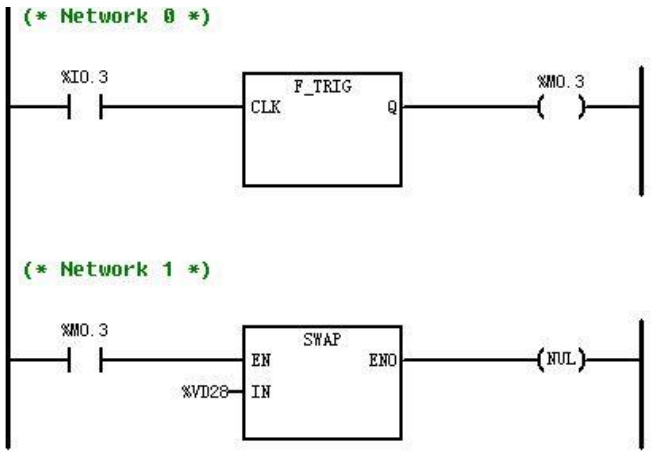
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

  该指令的执行对 CR 值无影响。

指令使用举例：

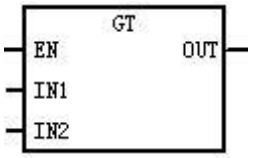
LD	IL
	(* Network 0 *) LD %I0.3 F_TRIG ST %M0.3 (* 若 I0.3 产生下降沿跳变 *) (* Network 1 *) LD %M0.3 SWAP %VD28 (* 交换 VD28 的高、低字节 *)

5.4 比较指令

注意：对于所有的比较指令，BYTE 类型的比较是无符号比较，其余的比较均为有符号比较。

5.4.1 GT（大于）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	GT		
IL	GT	GT *IN1, IN2*	P

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- **LD**

  如果 *EN* 为 1，该指令被执行：若 *IN1* 大于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；

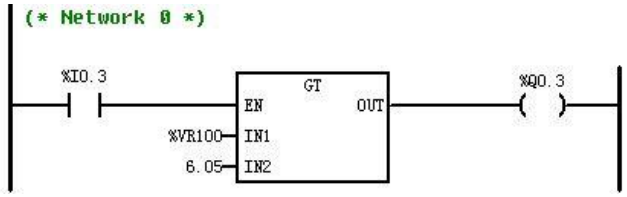
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- **IL**

  如果 CR 值为 1，该指令被执行：若 *IN1* 大于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

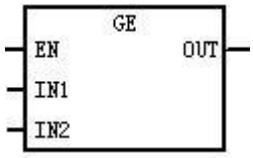
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.3(* 建立 CR，其值为 I0.3 的值 *) GT %VR100, 6.05 ST %Q0.3（若 CR 为 1：若 VR100 大于 6.05，则 Q0.3 被置为 1，否则 Q0.3 被置为 0 ；若 CR 为 0：不进行 GT 比较，Q0.3 保持为 0 *)

5.4.2 GE（大于等于）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	GE		
IL	GE	GE *IN1, IN2*	P

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- **LD**

  如果 *EN* 为 1，该指令被执行：若 *IN1* 大于等于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；

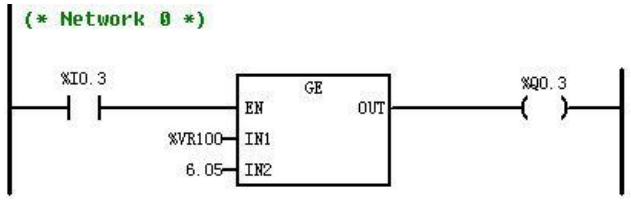
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- **IL**

如果 CR 值为 1，该指令被执行：若 *IN1* 大于等于 *IN2*，则将 CR 值置为 1，否则将 CR 值置为 0。

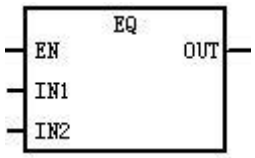
如果 CR 值为 0，该指令不执行，CR 值保持为 0。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.3(* 建立 CR，其值为 I0.3 的值 *) GE %VR100, 6.05 ST %Q0.3（若 CR 为 1：若 VR100 大于等于 6.05，则 Q0.3 被置为 1，否则 Q0.3 被置为 0 ；若 CR 为 0：不进行 GT 比较，Q0.3 保持为 0 *)

5.4.3 EQ（等于）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	EQ		
IL	EQ	EQ *IN1, IN2*	P

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- **LD**

  如果 *EN* 为 1，该指令被执行：若 *IN1* 等于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；

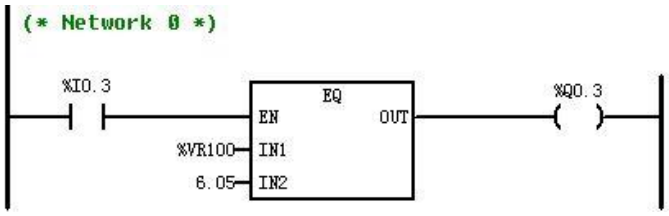
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- **IL**

  如果 CR 值为 1，该指令被执行：若 *IN1* 等于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.3(* 建立 CR，其值为 I0.3 的值 *) EQ %VR100, 6.05 ST %Q0.3(若 CR 为 1：若 VR100 等于 6.05，则 Q0.3 被置为 1，否则 Q0.3 被置为 0 ；若 CR 为 0：不进行 GT 比较，Q0.3 保持为 0 *)

5.4.4 NE（不等于）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	NE		
IL	NE	NE *IN1, IN2*	P

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- **LD**

  若 *EN* 为 1，该指令被执行：若 *IN1* 不等于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；

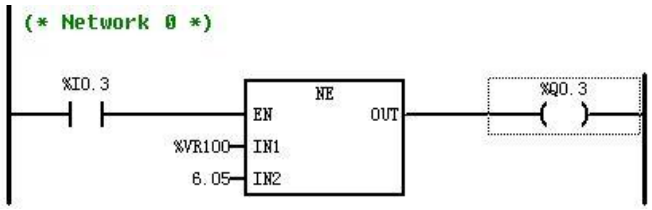
  若 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- **IL**

  若 CR 值为 1，该指令被执行：若 *IN1* 不等于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

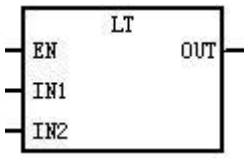
  若 CR 值为 0，该指令不执行，CR 值保持为 0。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.3(* 建立 CR，其值为 I0.3 的值 *) NE %VR100, 6.05 ST %Q0.3（若 CR 为 1：若 VR100 不等于 6.05，则 Q0.3 被置为 1，否则 Q0.3 被置为 0 ；若 CR 为 0：不进行 GT 比较，Q0.3 保持为 0 *)

5.4.5 LT（小于）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	LT		
IL	LT	LT *IN1, IN2*	P

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- **LD**

  如果 *EN* 为 1，该指令被执行：若 *IN1* 小于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；

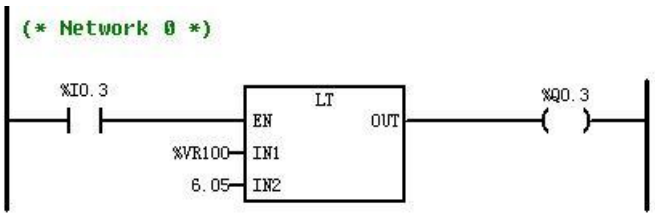
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- **IL**

  如果 CR 值为 1，该指令被执行：若 *IN1* 小于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；

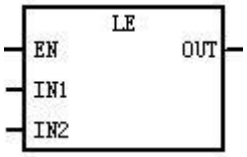
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.3(* 建立 CR，其值为 I0.3 的值 *) LT %VR100, 6.05 ST %Q0.3（若 CR 为 1：若 VR100 小于 6.05，则 Q0.3 被置为 1，否则 Q0.3 被置为 0 ；若 CR 为 0：不进行 GT 比较，Q0.3 保持为 0 *)

5.4.6 LE（小于等于）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	LE		
IL	LE	LE *IN1, IN2*	P

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
IN2	输入	BYTE、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT（LD）	输出	BOOL	能量流

参数 *IN1*、*IN2* 的数据类型必须一致。

- **LD**

  如果 *EN* 为 1，该指令被执行：若 *IN1* 小于等于 *IN2*，则在 *OUT* 输出为 1，否则输出为 0；

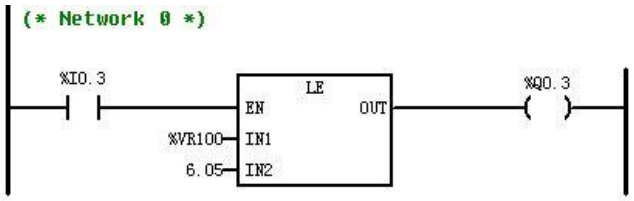
  如果 *EN* 为 0，该指令不执行，在 *OUT* 端输出为 0。

- **IL**

  如果 CR 值为 1，该指令被执行：若 *IN1* 小于等于 *IN2*，则将 CR 值置为 1，否则将 CR 值置为 0；

如果 CR 值为 0，该指令不执行，CR 值保持为 0。

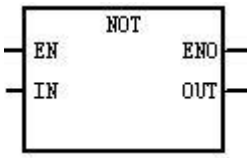
指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.3(* 建立 CR，其值为 I0.3 的值 *) GE %VR100, 6.05 ST %Q0.3（若 CR 为 1：若 VR100 小于等于 6.05，则 Q0.3 被置为 1，否则 Q0.3 被置为 0；若 CR 为 0：不进行 GT 比较，Q0.3 保持为 0 *)

5.5 逻辑运算

5.5.1 NOT（按位取反）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	NOT		
IL	NOT	NOT *OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

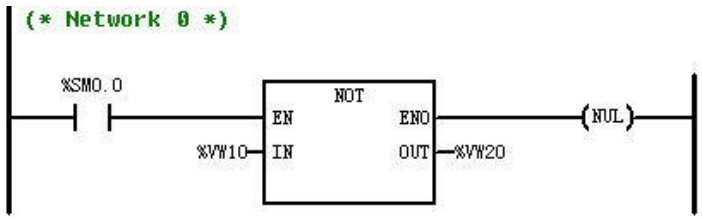
  参数 *IN*、*OUT* 的数据类型必须一致。

  如果 *EN* 为 1，则该指令被执行：将 *IN* 的每一个二进制位都取反，然后将结果赋给 *OUT*。

- **IL**

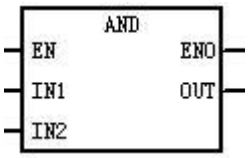
  如果 CR 值为 1，则该指令被执行：将 *OUT* 的每一个二进制位都取反，结果仍旧赋给 *OUT*。该指令的执行不影响 CR 值。

  指令使用举例：

LD	IL
	(* Network 0 *) LD %SM0.0 (* 建立 CR，其值为 1 *) MOVE %VW10, %VW20 NOT %VW20 (* 将 VW10 按位取反，结果仍放于 VW20 中 *)
若 *NOT* 被执行，则结果如下：	
VW10 W#16#FFFF　　VW20 W#16#0000	

5.5.2 AND（按位与）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	AND		
IL	AND	AND　　*IN1, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

  若 *EN* 为 1，则该指令被执行：将 *IN1* 和 *IN2* 按二进制位进行“与”运算后，将结果赋给 *OUT*。

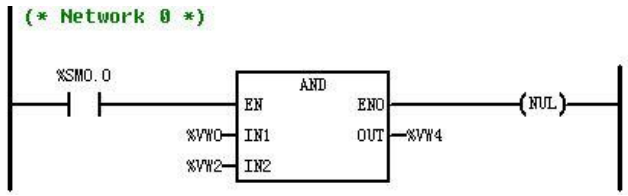
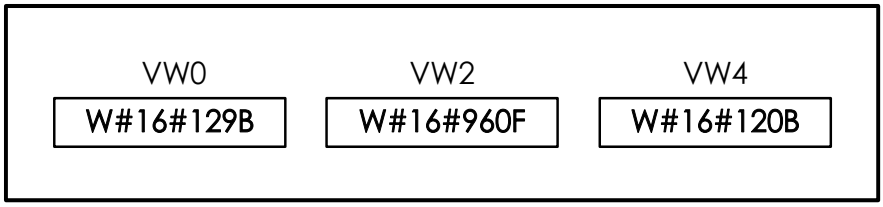
- **IL**

  参数 *IN1*、*OUT* 的数据类型必须一致。

若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“与”运算后，将结果赋给 *OUT*。

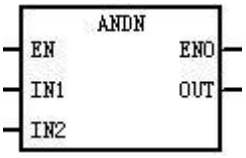
该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %SM0.0(* 建立 CR，其值为 1 *) MOVE %VW0, %VW4 AND %VW2, %VW4 (*将 VW0 和 VW2 的值按位相“与”，结果仍放于 VW4 中 *)
若 *AND* 被执行，则结果如下：	
	

5.5.3 ANDN（按位与非）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ANDN		
IL	ANDN	ANDN *IN1, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

  若 *EN* 为 1，则该指令被执行：将 *IN1* 和 *IN2* 按二进制位进行“与”运算并取反，然后将结果

赋给 *OUT*。

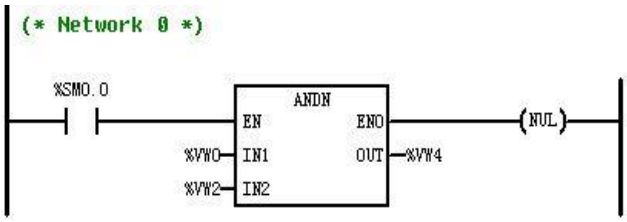
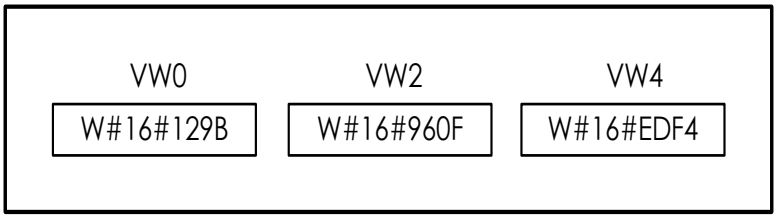
- **IL**

参数 *IN1*、*OUT* 的数据类型必须一致。

若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“与”运算并取反，然后将结果赋给 *OUT*。

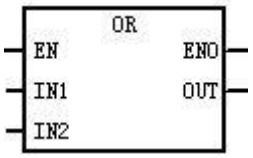
该指令的执行不影响 CR 值。

指令使用举例：

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* Network 0 *)
LD %SM0.0(* 建立 CR，其值为 1 *)
MOVE %VW0, %VW4
ANDN %VW2, %VW4
(*将 VW0 和 VW2 的值按位相“与” 并取反，结果仍放于 VW4 中 *)</td></tr>
<tr><td colspan="2">若 ANDN 被执行，则结果如下：</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

5.5.4 OR（按位或）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	OR		
IL	OR	OR *IN1, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

  若 *EN* 为 1，则该指令被执行：将 *IN1* 和 *IN2* 按二进制位进行“或”运算后，将结果赋给 *OUT*。

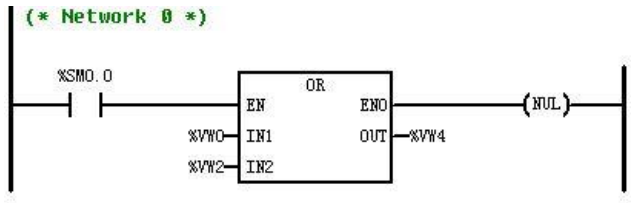
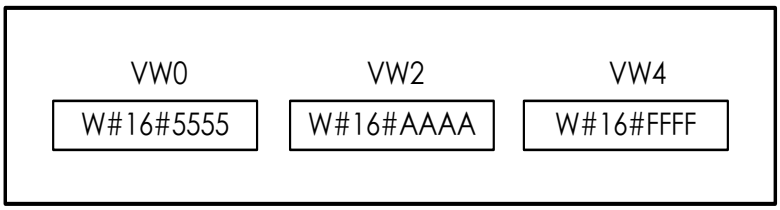
- **IL**

  参数 *IN1*、*OUT* 的数据类型必须一致。

  若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“或”运算后，将结果赋给 *OUT*。

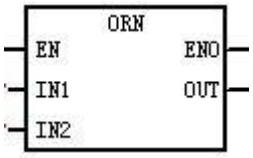
  该指令的执行不影响 CR 值。

指令使用举例：

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* Network 0 *)
LD %SM0.0(* 建立 CR，其值为 1 *)
MOVE %VW0, %VW4
OR %VW2, %VW4
(*将 VW0 和 VW2 的值按位相“与” 并取反，结果仍放于 VW4 中 *)</td></tr>
<tr><td colspan="2">若 OR 被执行，则结果如下：</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

5.5.5 ORN（按位或非）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	ORN		
IL	ORN	ORN I*N1, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

若 *EN* 为 1，则该指令被执行：将 *IN1 和 IN2* 按二进制位进行“或”运算并取反，然后将结果赋给 *OUT*。

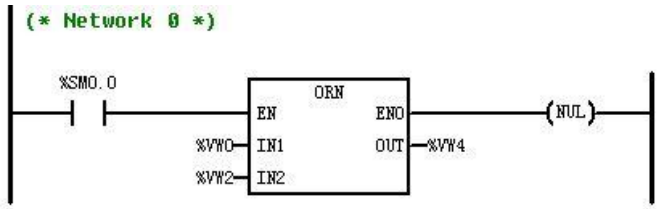
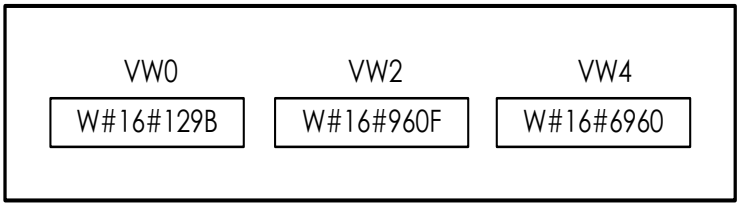
- **IL**

参数 *IN*、*OUT* 的数据类型必须一致。

若 CR 值为 1，则该指令被执行：将 *IN1 和 OUT* 按二进制位进行“或”运算并取反，然后将结果赋给 *OUT*。

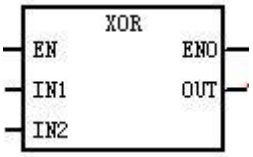
该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %SM0.0(* 建立 CR，其值为 1 *) MOVE %VW0, %VW4 ORN %VW2, %VW4 (*将 VW0 和 VW2 的值按位相“与” 并取反，结果仍放于 VW4 中 *)
若 *ORN* 被执行，则结果如下：	
	

5.5.6 XOR（按位异或）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	XOR		
IL	XOR	XOR *IN1, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN1	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
IN2	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

  若 *EN* 为 1，则该指令执行：将 *IN1* 和 *IN2* 按二进制位进行“异或”运算后，将结果赋给 *OUT*。

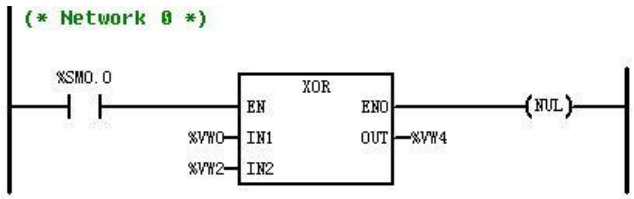
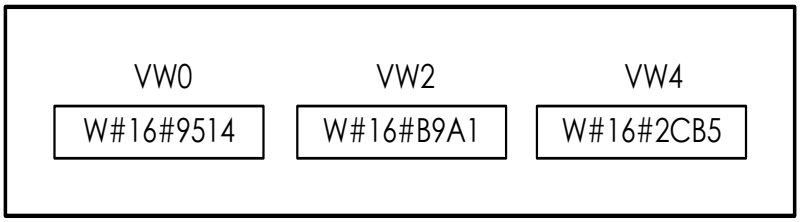
- **IL**

参数 *IN1*、*OUT* 的数据类型必须一致。

若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“异或”运算后，将结果赋给 *OUT*。

该指令的执行不影响 CR 值。

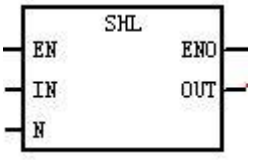
指令使用举例：

LD	IL
	(* Network 0 *) LD %SM0.0(* 建立 CR，其值为 1 *) MOVE %VW0, %VW4 ORN %VW2, %VW4 (*将 VW0 和 VW2 的值按位相“与” 并取反，结果仍放于 VW4 中 *)
若 *XOR* 被执行，则结果如下：	
	

5.6 移位指令

5.6.1 SHL（左移）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	SHL		
IL	SHL	SHL *OUT, N*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

参数 *IN*、*OUT* 的数据类型必须一致。

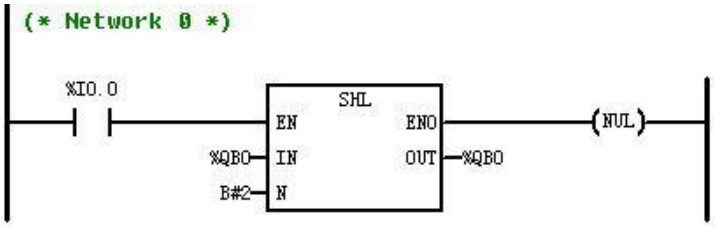
若 *EN* 为 1，则该指令被执行：将 *IN* 的全部二进制位向左移动 *N* 位，移出的高位被舍弃并且低位补 0，最终的结果赋给 *OUT*。

- **IL**

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向左移动 *N* 位，移出的高位被舍弃并且低位补 0，最终的结果仍旧被赋给 *OUT*。

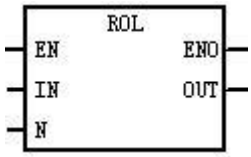
该指令的执行不影响 CR 值。

指令使用举例：

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* Network 0 *)
LD %I0.0(* 建立 CR，其值为 I0.0 的值 *)
SHL %QB0, B#2(* 若 CR 为 1：将 QB0 值的全部二进制位每次左移 2 位，结果仍放于 QB0 中；若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td colspan="2">若 SHL 被执行，则结果如下：</td></tr>
<tr><td colspan="2">QB0 初值：B#2#10111110
执行一次 SHL 指令后 QB0 的 B#2#11111000</td></tr>
</table>

5.6.2 ROL（循环左移）

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值
LD	ROL		
IL	ROL	ROL *OUT, N*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

参数 *IN*、*OUT* 的数据类型必须一致。

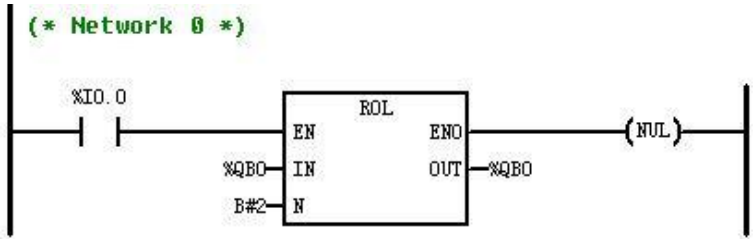
若 *EN* 为 1，则该指令被执行：将 *IN* 的全部二进制位向左移动 *N* 位，移出的高位被依次移进低位位置，最终的结果赋给 *OUT*。

- **IL**

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向左移动 *N* 位，移出的高位被依次移进低位位置，最终的结果仍旧被赋给 *OUT*。

该指令的执行不影响 CR 值。

指令使用举例：

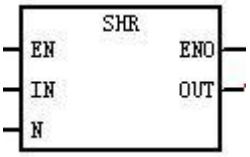
LD	IL
	(* Network 0 *) LD %I0.0(* 建立 CR，其值为 I0.0 的值 *) ROL %QB0, B#2(* 若 CR 为 1：将 QB0 值的全部二进制位每次循环左移 2 位，结果仍放于 QB0 中 ； 若 CR 为 0：不执行 SHL 指令 *)
若 *ROL* 被执行，则结果如下：	

QB0 初值：　B#2#10111110

执行一次 ROL 指令后 QB0 的　B#2#11111010

5.6.3 SHR（右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	SHR		
IL	SHR	SHR　*OUT, N*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

参数 *IN*、*OUT* 的数据类型必须一致。

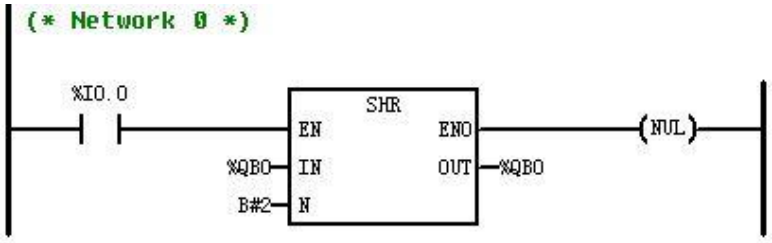
若 *EN* 为 1，则该指令被执行：将 *IN* 的全部二进制位向右移动 *N* 位，移出的低位被舍弃并且高位补 0，最终的结果赋给 *OUT*。

- **IL**

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向右移动 *N* 位，移出的低位被舍弃并且高位补 0，最终的结果仍旧被赋给 *OUT*。

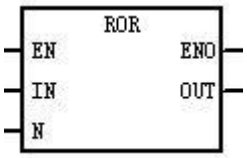
该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.0(* 建立 CR，其值为 I0.0 的值 *) SHR %QB0, B#2(* 若 CR 为 1：将 QB0 值的全部二进制位每次右移 2 位，结果仍放于 QB0 中 ； 若 CR 为 0：不执行 SHL 指令 *)
若 *SHR* 被执行，则结果如下：	
QB0 初值： B#2#10111110 执行一次 SHR 指令后 QB0 的 B#2#00101111	

5.6.4 ROR（循环右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ROR		
IL	ROR	ROR *OUT, N*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD	I、Q、M、V、L、SM、常量、指针
N	输入	BYTE	I、Q、M、V、L、SM、常量、指针
OUT	输出	BYTE、WORD、DWORD	Q、M、V、L、SM、指针

- **LD**

参数 *IN*、*OUT* 的数据类型必须一致。

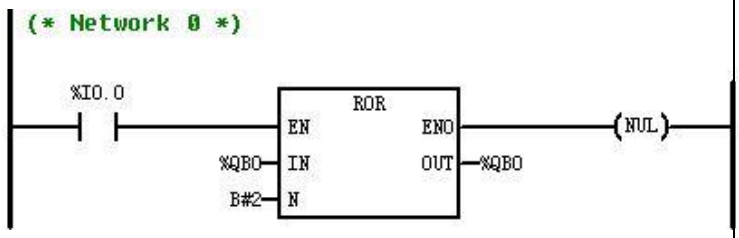
若 *EN* 为 1，则该指令被执行：将 *IN* 的全部二进制位向右移动 *N* 位，移出的低位被依次移进高位位置，最终的结果赋给 *OUT*。

- **IL**

若 CR 值为 1，则该指令被执行：将 *OUT* 的全部二进制位向右移动 *N* 位，移出的低位被依次移进高位位置，最终的结果仍旧被赋给 *OUT*。

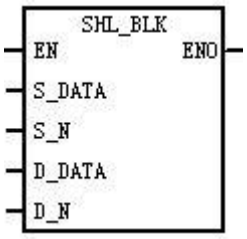
该指令的执行不影响 CR 值。

指令使用举例：

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* Network 0 *)
LD %I0.0
(* 建立CR，其值为I0.0的值 *)
ROR %QB0, B#2
(* 若 CR 为 1：将 QB0 值的全部二进制位每次循环右移 2 位，结果仍放于 QB0 中 ； 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td colspan="2">若 ROR 被执行，则结果如下：</td></tr>
<tr><td colspan="2">QB0 初值： B#2#10111110
执行一次 ROR 指令后 QB0 的 B#2#10101111</td></tr>
</table>

5.6.5 SHL_BLK（位串左移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	SHL_BLK		
IL	SHL_BLK	SHL_BLK S_DATA, S_N, D_DATA, D_N	U

参数	输入/输出	数据类型	允许的内存区
S_DATA	输入	BOOL	I、Q、M、V、L
S_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针
D_DATA	输入/输出	BOOL	Q、M、V、L
D_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针

该指令执行时，将从 *D_DATA* 开始的连续 *D_N* 个二进制位向左移动 *S_N* 位，移出的高位被丢弃，同时将从 *S_DATA* 开始的连续 *S_N* 个二进制位补入 *D_DATA* 的最右端。

- **LD**

若 *EN* 为 1，则该指令被执行。

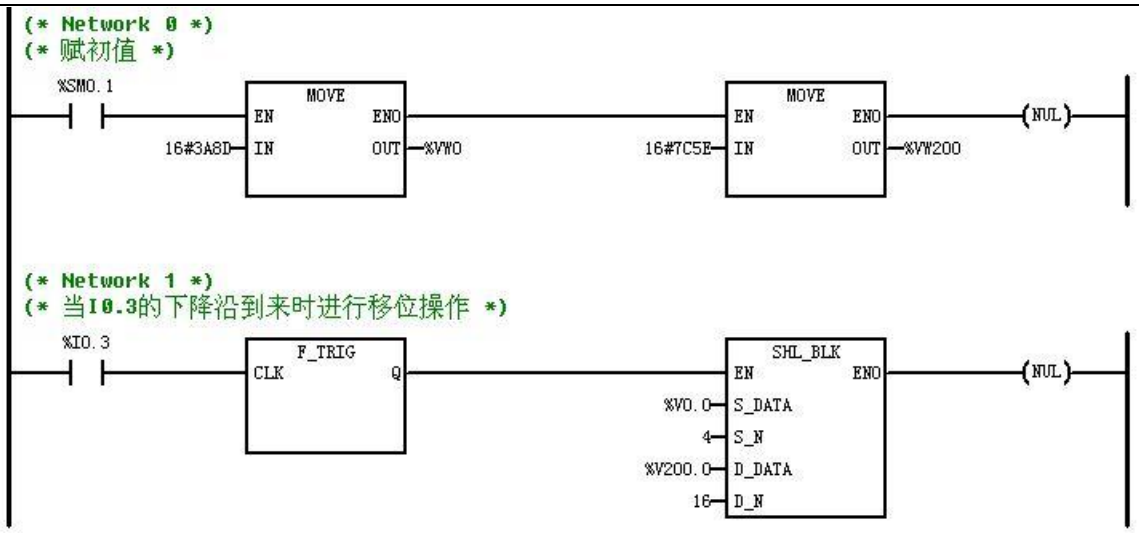
- **IL**

若 CR 值为 1，则该指令被执行。

该指令的执行不影响 CR 值。

指令使用举例：

LD



IL

```
(* Network 0 *)
(*赋初值*)
LD        %SM0.1
MOVE      16#3A8D, %VW0
MOVE      16#7C5E, %VW200
(* Network 1 *)
(*当 I0.3 的下降沿到来时进行移位操作*)
LD        %I0.3
F_TRIG
SHL_BLK %V0.0, 4, %V200.0, 16
```

结果

V0.0

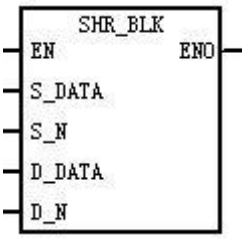
VW0 的值	0011	1010	1000	**1101**

V200.0

VW200 初始值	0111	1100	0101	1110
执行一次 SHR_BLK 后 VW200 的值	1100	0101	1110	**1101**

5.6.6 SHR_BLK（位串右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	SHR_BLK		
IL	SHR_BLK	SHR_BLK S_DATA, S_N, D_DATA, D_N	U

参数	输入/输出	数据类型	允许的内存区
S_DATA	输入	BOOL	I、Q、M、V、L
S_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针
D_DATA	输入/输出	BOOL	Q、M、V、L
D_N	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量、指针

该指令执行时，将从 *D_DATA* 开始的连续 *D_N* 个二进制位向右移动 *S_N* 位，移出的低位被丢弃，同时将从 *S_DATA* 开始的连续 *S_N* 个二进制位补入 *D_DATA* 的最左端。

- **LD**

  若 *EN* 为 1，则该指令被执行。

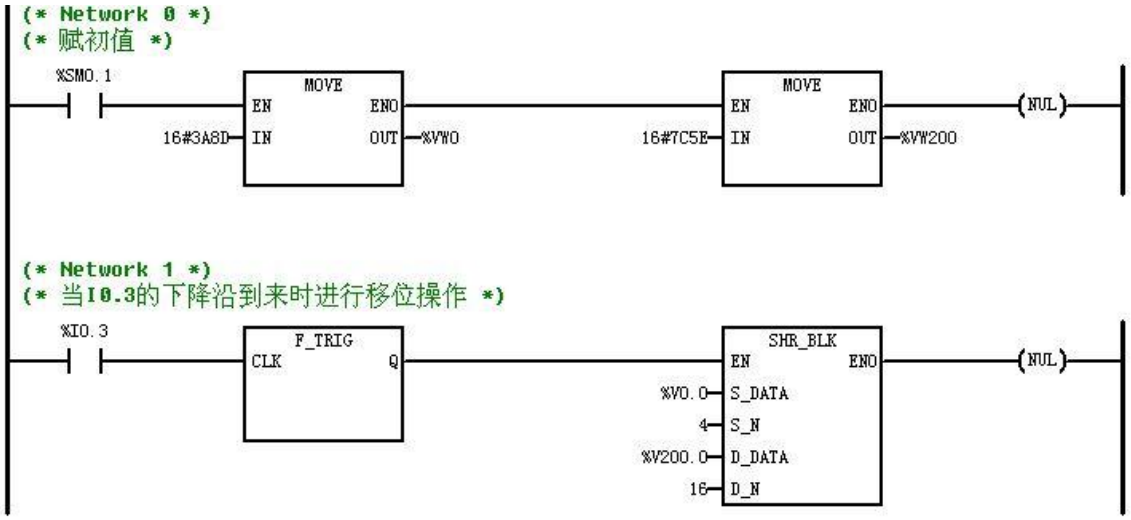
- **IL**

  若 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

指令使用举例：

LD



IL

```
(* Network 0 *)
(*赋初值*)
LD          %SM0.1
MOVE        16#3A8D, %VW0
MOVE        16#7C5E, %VW200
(* Network 1 *)
(*当 I0.3 的下降沿到来时进行移位操作*)
LD          %I0.3
F_TRIG
SHR_BLK %V0.0, 4, %V200.0, 16
```

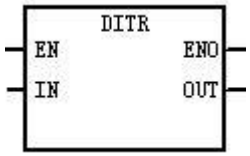
结果

	V0.0			
VW0 的值	0011	1010	1000	**1101**
	V200.0			
VW200 初始值	0111	1100	0101	1110
执行一次 SHR_BLK 后 VW200 的值	**1101**	0111	1100	0101

5.7 类型转换

5.7.1 DITR（双整型转实型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	DITR		
IL	DITR	DITR *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
OUT	输出	REAL	V、L

该指令将输入的双整数 *IN* 转换为实数并赋给 *OUT*。

- **LD**

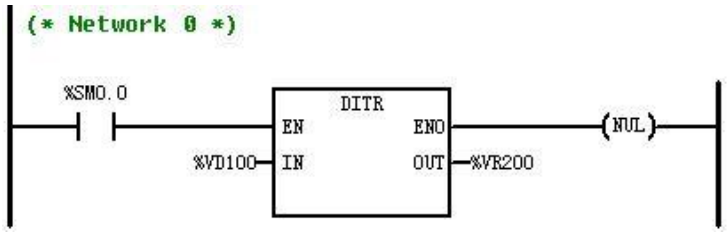
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

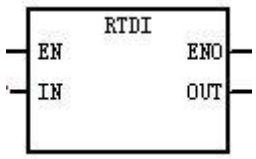
  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %SM0.0 (* 建立 CR，其值为 1 *) DITR %VD100, %VR200 (* 将 VD100 转换为实数并赋给 VR200 *)
举例结果如下：	
VD100 值： 356 执行一次 DITR 指令后 VR200 356．000000	

5.7.2 RTDI（实型转双整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	RTDI		
IL	RTDI	RTDI *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L 、常量
OUT	输出	DINT	M、V、L、SM

该指令将输入的实数 *IN* 转换为双整数（小数部分四舍五入）并赋给 *OUT*。

- **LD**

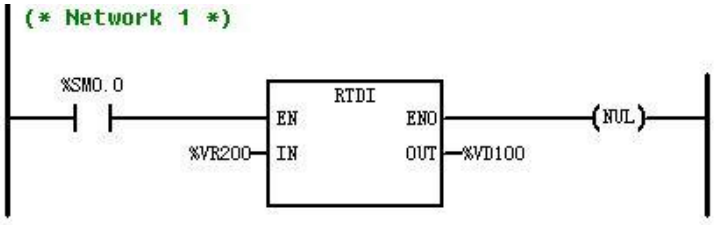
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

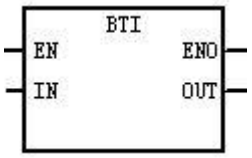
  该指令的执行不影响 CR 值。

指令使用举例：

LD	**IL**
	(* Network 1 *) LD %SM0.0 (* 建立 CR，其值为 1 *) RTDI %VR200, %VD100 (* 将 VR200 转换为双整数并赋给 VD100 *)
举例结果如下：	
VR200 值：356.7 执行 RTDI 指令后 VD100 的值：357	

5.7.3 BTI（字节型转整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	BTI		
IL	BTI	BTI *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	INT	Q、M、V、L、SM、AQ

该指令将输入的字节型数据 *IN* 转换为整数并赋给 *OUT*。

- **LD**

  如果 *EN* 为 1，则该指令被执行。

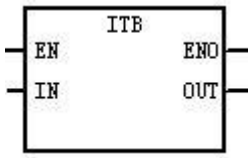
- **IL**

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

5.7.4 ITB（整型转字节型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ITB		
IL	ITB	ITB *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令将输入 *IN* 的低字节赋给 *OUT*。

- **LD**

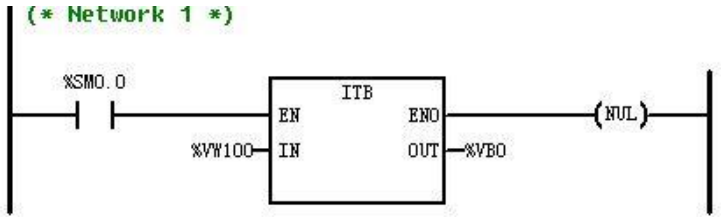
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

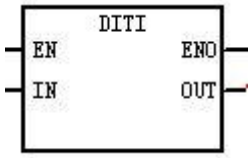
  该指令的执行不影响 CR 值。

指令使用举例

LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) ITB %VW100, %VB0 (* 将 VW100 的低字节赋给 VB0 *)
举例结果如下：	
VW200 值：I#16#ABCD 执行 ITB 指令后 VB0 的值：I#16#CD	

5.7.5 DITI（双整型转整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	DITI		
IL	DITI	DITI *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
OUT	输出	INT	Q、M、V、L、SM、AQ

该指令将输入 *IN* 的低字赋给 *OUT*。

- **LD**

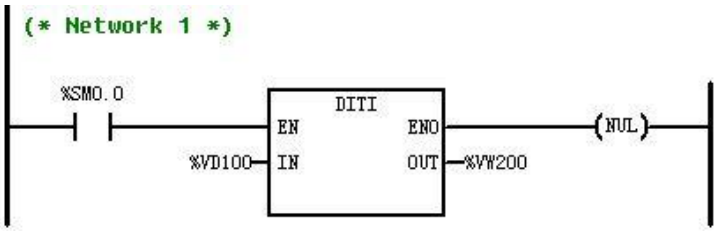
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

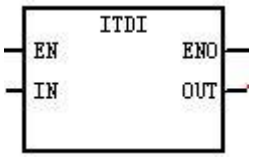
  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) DITI %VD100, %VW200 (* 将 VD100 的低字节赋给 VW200 *)
举例结果如下：	
VD100 值：DI#16#ABCD1234 执行 DITI 指令后 VW200 的值：I#16#1234	

5.7.6 ITDI（整型转双整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ITDI		
IL	ITDI	ITDI *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
OUT	输出	DINT	Q、M、V、L、SM

该指令将输入的整数 *IN* 转换为双整数并赋给 *OUT*。

- **LD**

如果 *EN* 为 1，则该指令被执行。

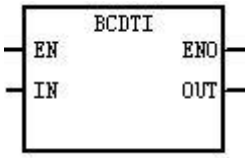
- **IL**

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

5.7.7 BCDTI（BCD 码转整型）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	BCDTI		
IL	BCDTI	BCDTI *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	WORD	I、Q、M、V、L、SM、常量
OUT	输出	INT	Q、M、V、L、SM、AQ

该指令将输入的 BCD 码 *IN* 转换为整数并赋给 *OUT*。

注意：BCD 码采用 8421 码。*IN* 的有效范围是 0∽9999 BCD，每个 16 进制位有效范围是 0∽9BCD，如果数值不在有效范围内，特殊寄存器 SM1.4 置位。

- **LD**

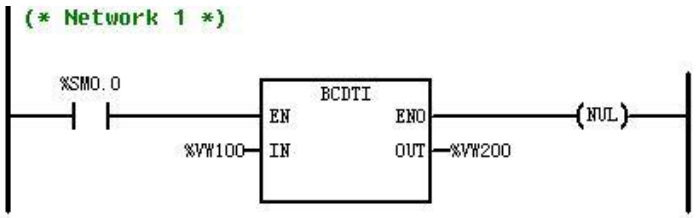
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

指令使用举例：

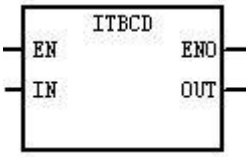
LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) BCDTI %VW100, %VW200 (* 将 VW100 由 BCD 转换为整数并赋给 VW200 *)

续

举例结果如下：
VW100 值：16#1234 执行 BCDTI 指令后 VW200 的 1234

5.7.8 ITBCD（整型转 BCD 码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ITBCD		
IL	ITBCD	ITBCD *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
OUT	输出	WORD	Q、M、V、L、SM

该指令将输入的整数 *IN* 转换为 BCD 码并赋给 *OUT*。

注意：BCD 码采用 8421 码。*IN* 的有效范围是 0∽9999，如果数值不在有效范围内，特殊寄存器 SM1.4 置位。

- **LD**

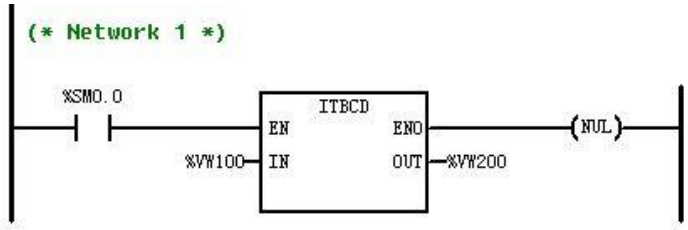
如果 *EN* 为 1，则该指令被执行。

- **IL**

如果 CR 值为 1，则该指令被执行。

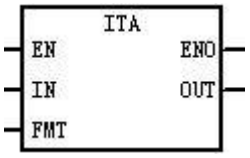
该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) ITBCD %VW100, %VW200 (* 将 VW100 由整数转换为 BCD 并赋给 VW200 *)
举例结果如下：	
VW100 值：1234 执行 ITBCD 指令后 VW200 的 16#1234	

5.7.9 ITA（整型转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ITA		
IL	ITA	ITA *IN, OUT, FMT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

该指令将输入的整数 *IN* 转换成一个 ASCII 字符串并将转换结果格式化输出到输出缓冲区中。正数转换后不带符号，负数转换后带负号。

参数 *OUT* 定义了输出缓冲区的起始地址，该区域占用连续 8 个字节的内存空间。字符串在缓冲区中右对齐，缓冲区中未使用的地址被赋值为空格符（ASCII 为 32）。

参数 *FMT* 定义了输出字符串的表示形式，它的组成如下图：

MSB							LSB
7	6	5	4	3	2	1	0
0	0	0	0	c	n	n	n

① nnn --- 低3位，指定了输出字符串中小数部分的位数。

nnn的有效范围是0到5。若指定为0，则表示没有小数部分。

② c --- 第4位，指定了整数部分和小数部分的分隔符。

0表示用小数点（ASCII为46），1表示用逗号（ASCII为44）。

③ 高4位必须全为0。

- **LD**

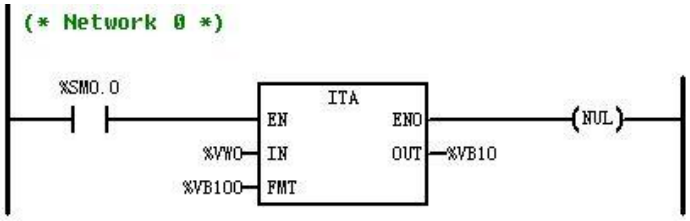
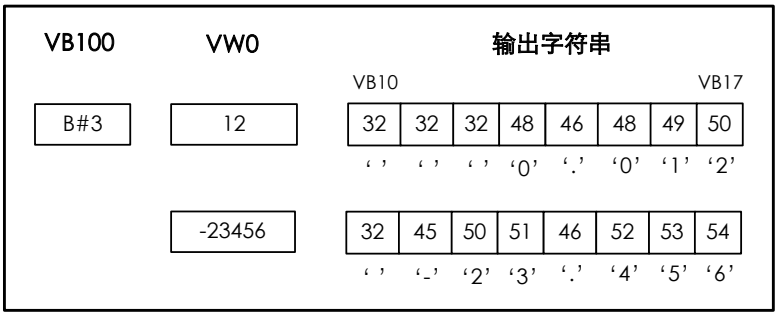
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

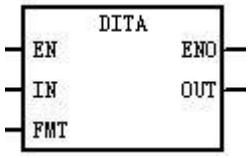
  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 1 *) LD %SM0.0 (* 建立 CR，其值为 1 *) ITA %VW0, %VB10, %VB100 (*将 VW0 的值转换为字符串并格式化输出到 VB10 开始的连续 8 个字节中。 *)
举例结果如下：	
	

5.7.10 DITA（双整型转 ASCII）

➢ 指令及其操作数说明

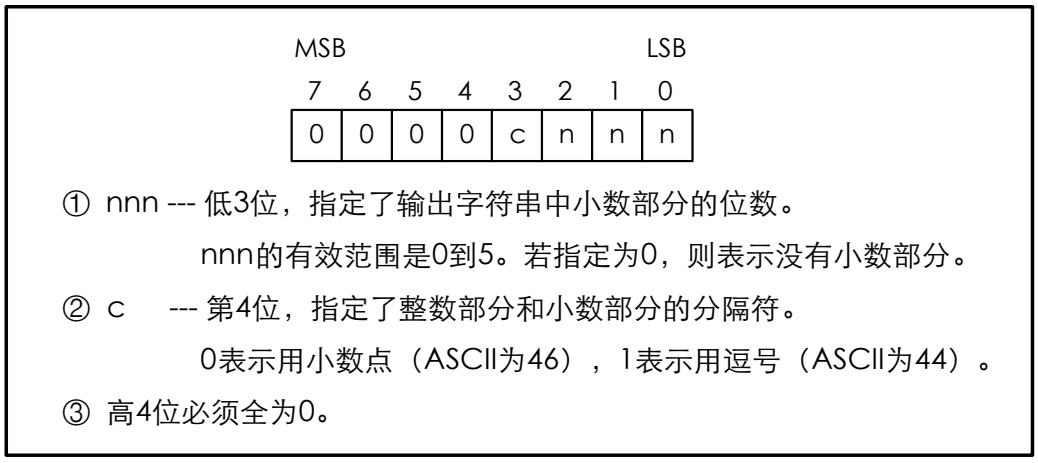
	名称	指令格式	影响 CR 值
LD	DITA		
IL	DITA	DITA *IN, OUT, FMT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

该指令将输入的双整数 *IN* 转换成一个 ASCII 字符串并将转换结果格式化输出到输出缓冲区中。正数转换后不带符号，负数转换后带负号。

参数 *OUT* 定义了输出缓冲区的起始地址，该区域占用连续 12 个字节的内存空间。字符串在缓冲区中右对齐，缓冲区中未使用的地址被赋值为空格符（ASCII 为 32）。

参数 *FMT* 定义了输出字符串的表示形式，它的组成如下图：



- **LD**

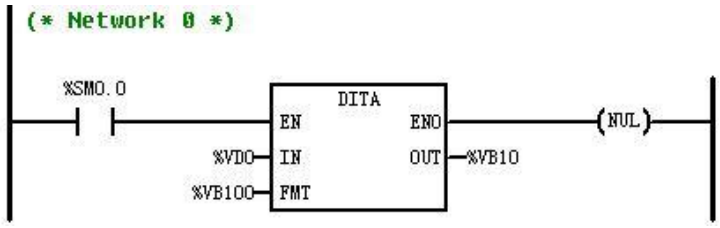
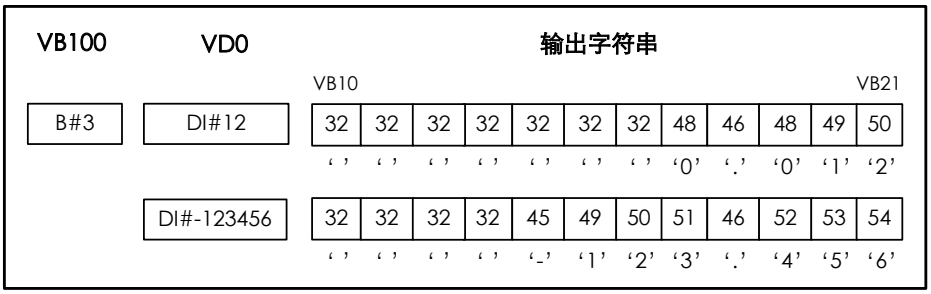
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

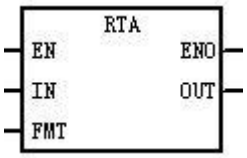
  该指令的执行不影响 CR 值。

指令使用举例：

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* Network 1 *)
LD %SM0.0(* 建立 CR，其值为 1 *)
DITA %VD0, %VB10, %VB100
(*将 VD0 的值转换为字符串并格式化输出到 VB10 开始的连续 12 个字节中*)</td></tr>
<tr><td colspan="2">举例结果如下：</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

5.7.11 RTA（实型转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	RTA		
IL	RTA	RTA *IN, OUT, FMT*	U

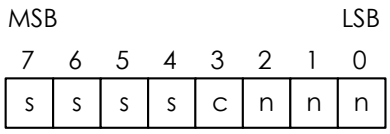
参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

该指令将输入的实数 *IN* 转换成一个 ASCII 字符串并将转换结果格式化输出到输出缓冲区中。正数转换后不带符号，负数转换后带负号。若 *IN* 的小数部分的位数大于参数 *FMT* 中 nnn 指定的小数位数，则转换之前首先将 *IN* 四舍五入，若小于则将 *IN* 的小数位数不足的部分补 0。

参数 *OUT* 定义了输出缓冲区的起始地址，该区域的大小在 *FMT* 中指定。字符串在缓冲区中

右对齐，缓冲区中未使用的地址被赋值为空格符（ASCII 为 32）。

参数 *FMT* 定义了输出字符串的表示形式，它的组成如下图：



① nnn --- 低3位，指定了输出字符串中小数部分的位数。有效范围是0到5。
若指定为0，则表示没有小数部分；若大于5，则CPU自动按5进行处理。

② c --- 第4位，指定了整数部分和小数部分的分隔符。
0表示用小数点（ASCII为46），1表示用逗号（ASCII为44）。

③ ssss --- 高4位，指定了输出缓冲区的大小。有效范围是3到15且必须大于nnn。
若小于3，则CPU自动按3进行处理。

需要注意的是，若输入 *IN* 的转换结果的长度超出了输出缓冲区的长度，则输出缓冲区全部用空格符（ASCII 为 32）填充。

- **LD**

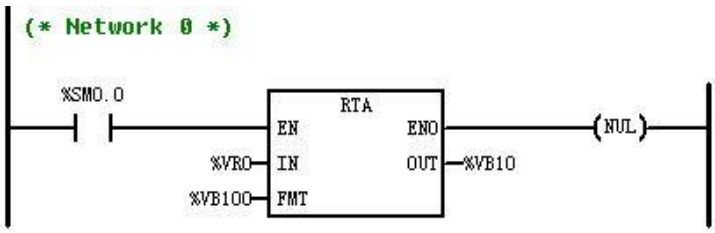
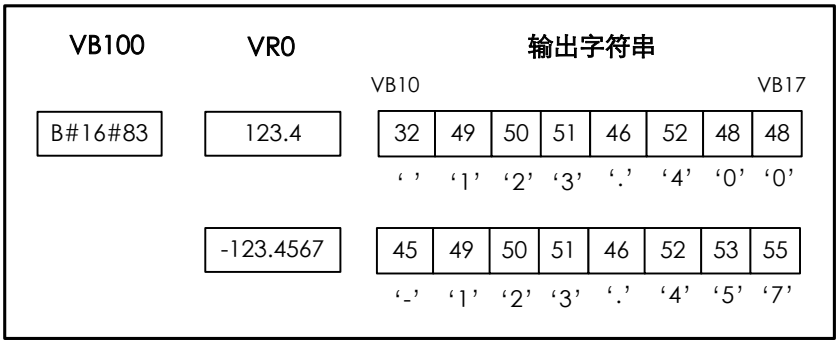
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

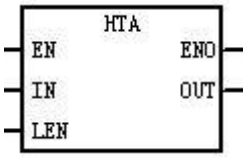
  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) RTA %VR0, %VB10, %VB100 (*VR0 的值转换为字符串并格式化输出到 VB10 开始的缓冲区中*)
例结果如下：	
	

5.7.12 HTA（16 进制数转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	HTA		
IL	HTA	HTA *IN, OUT, LEN*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM
LEN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令将地址 *IN* 开始的连续 *LEN* 个字节的十六进制数转换成一个 ASCII 字符串并输出到地址 *OUT* 开始的输出缓冲区中。注意：由于每 4 个二进制位表示一个十六进制数，因此每个输入字节包含了 2 个十六进制数，输出缓冲区共占用了 *LEN*×*2* 个字节的空间。

- **LD**

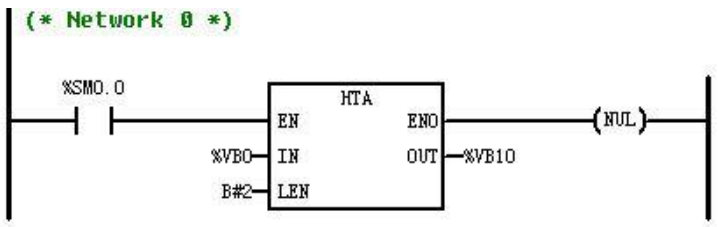
  如果 *EN* 为 1，则该指令被执行。

- **IL**

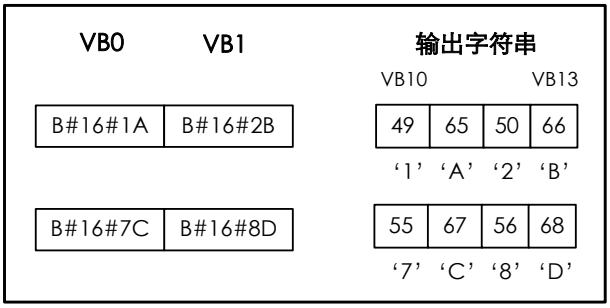
  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

指令使用举例：

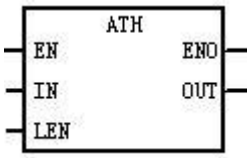
LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) HTA %VB0, %VB10, B#2 (*将 VB0 开始连续 2 个字节的十六进制数转换为字符串并输出到 VB10 开始的连续 4 个字节中*)

举例结果如下：



5.7.13 ATH（ASCII 转 16 进制数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ATH		
IL	ATH	ATH *IN, OUT, LEN*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM
LEN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令将地址 *IN* 开始的连续 *LEN* 个 ASCII 字符转换成十六进制数并输出到地址 *OUT* 开始的输出缓冲区中。注意：由于一个十六进制数采用 4 个二进制位来表示，因此每个输入字节（表示一个 ASCII 字符）转换后在输出缓冲区中占用 4 个二进制位（半个字节）的空间。

有效的 ASCII 输入范围是：B#16#30∽B#16#39（表示字符 0∽9），B#16#41∽B#16#46（表示字符 A∽F）。

注意：如果 ASCII 输入的数据不在有效范围内，则指令不执行，且特殊寄存器 SM1.6 置位。

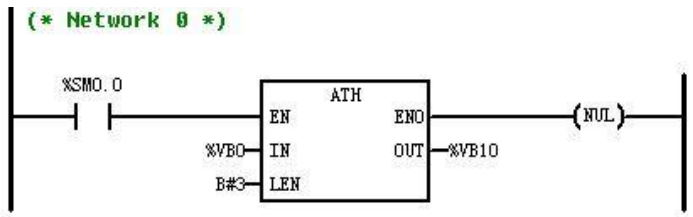
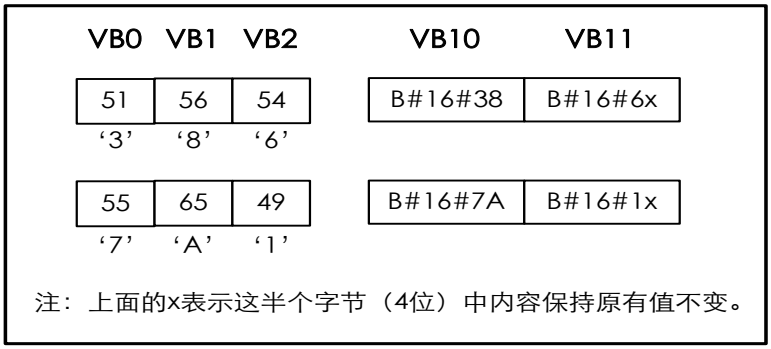
- **LD**

  如果 *EN* 为 1，则该指令被执行。

- **IL**

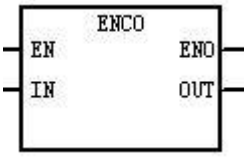
  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

指令使用举例：

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* Network 1 *)
LD %SM0.0(* 建立 CR，其值为 1 *)
ATH %VB0, %VB10, B#3 (*将 VB0 开始的连续 3 个字节的 ASCII 字符转换成十六进制数并输出到 VB100 开始的输出缓冲区中*)</td></tr>
<tr><td colspan="2">举例结果如下：</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

5.7.14 ENCO（编码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ENCO		
IL	ENCO	ENCO *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	WORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令从输入字 *IN* 的最低位开始计算，将第一个为 1 的位的位号写入输出字节 *OUT*。注意：若 *IN* 的值等于 0，那么计算结果是无意义的。

- **LD**

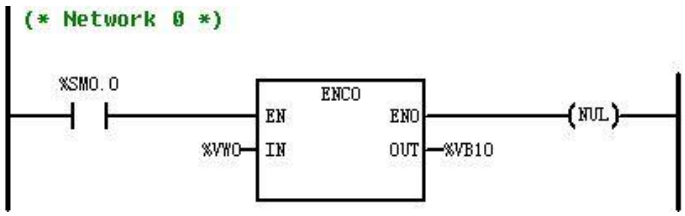
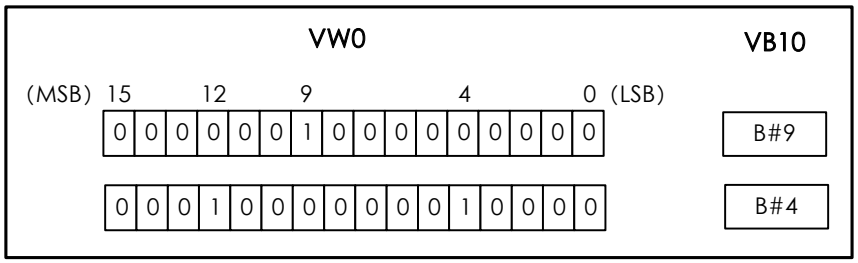
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

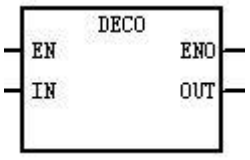
  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) ENCO %VW0, %VB10 (*从 VW0 中的最低位开始计算，将第一个等于 1 的位的位号写入 VB10 中*)
举例结果如下：	
VW0 的值采用了二进制来表示 	

5.7.15 DECO（解码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	DECO		
IL	DECO	DECO *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	WORD	Q、M、V、L、SM

该指令用输入字节 *IN* 的低四位所表示的数值来指定一个位号，然后根据这个位号将输出字

OUT 中对应的位赋值为 1，其它位全部赋值为 0。

- **LD**

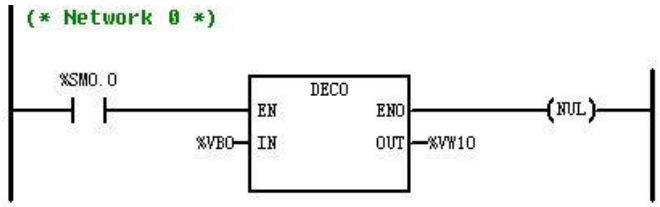
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

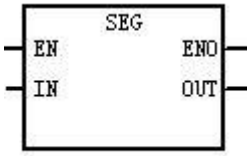
  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) DECO %VB0, %VW10 (*VB0 的低四位所表示的数值代表了位号，根据这个位号将 VW0 中的相应位赋值为 1，其它为全部赋值为 0*)
举例结果如下：	
VW10 的值采用了二进制来表示 	

5.7.16 SEG（七段码显示）

➢ 指令及其操作数说明

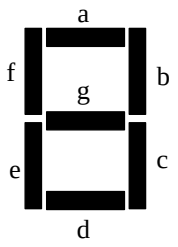
	名称	指令格式	影响 CR 值
LD	SEG		
IL	SEG	SEG *IN, OUT*	U

续

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

输入字节 *IN* 的低四位表示了要显示的数值，SEG 指令根据该数值来产生七段码的段码值并输出至 *OUT* 中。IN 的高四位数据不起作用。

IN（低四位）	显示	*OUT*（- g f e d c b a）
0	0	0 0 1 1 1 1 1 1
1	1	0 0 0 0 0 1 1 0
2	2	0 1 0 1 1 0 1 1
3	3	0 1 0 0 1 1 1 1
4	4	0 1 1 0 0 1 1 0
5	5	0 1 1 0 1 1 0 1
6	6	0 1 1 1 1 1 0 1
7	7	0 0 0 0 0 1 1 1



IN（低四位）	显示	*OUT*（- g f e d c b a）
8	8	0 1 1 1 1 1 1 1
9	9	0 1 1 0 0 1 1 1
A	A	0 1 1 1 0 1 1 1
B	B	0 1 1 1 1 1 0 0
C	C	0 0 1 1 1 0 0 1
D	D	0 1 0 1 1 1 1 0
E	E	0 1 1 1 1 0 0 1
F	F	0 1 1 1 0 0 0 1

- **LD**

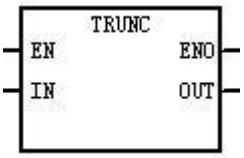
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

5.7.17 TRUNC（取整）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	TRUNC		
IL	TRUNC	TRUNC *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L 、常量
OUT	输出	DINT	M、V、L、SM

该指令将输入的实数 *IN* 转换为双整数（小数部分被舍弃）并赋给 *OUT*。

- **LD**

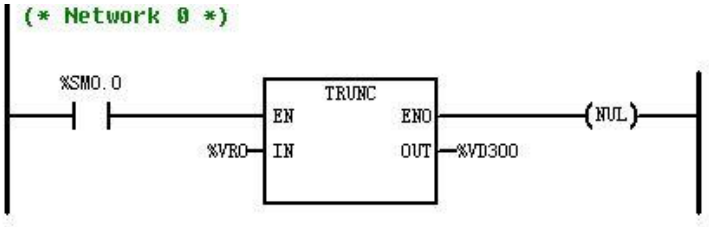
  如果 *EN* 为 1，则该指令被执行。

- **IL**

  如果 CR 值为 1，则该指令被执行。

该指令的执行不影响 CR 值。

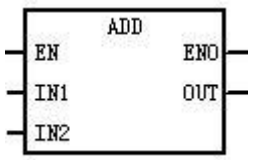
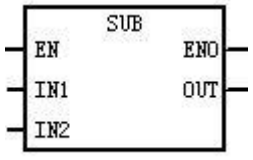
指令使用举例：

LD	IL
	(* Network 1 *) LD %SM0.0(* 建立 CR，其值为 1 *) TRUNC %VR0, %VD300 (*将实数 VR0 舍弃小数部分后转换为双整数并赋给 VD300*)
举例结果如下：	
VR0 值： 1234.5 执行 TRUNC 指令后 VD300 的 1234	

5.8 数学运算

5.8.1 ADD（加法）、SUB（减法）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ADD		
	SUB		
IL	ADD	ADD *IN1, OUT*	U
	SUB	SUB *IN1, OUT*	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
IN2	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
OUT	输出	INT、DINT、REAL	Q、AQ、M、V、L、SM、指针

- **LD**

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

若 *EN* 值为 1，则指令被执行。其中，*ADD* 指令的功能是：*OUT*＝*IN1*＋*IN2*；*SUB* 指令的功能是：*OUT*＝*IN1*－*IN2*。

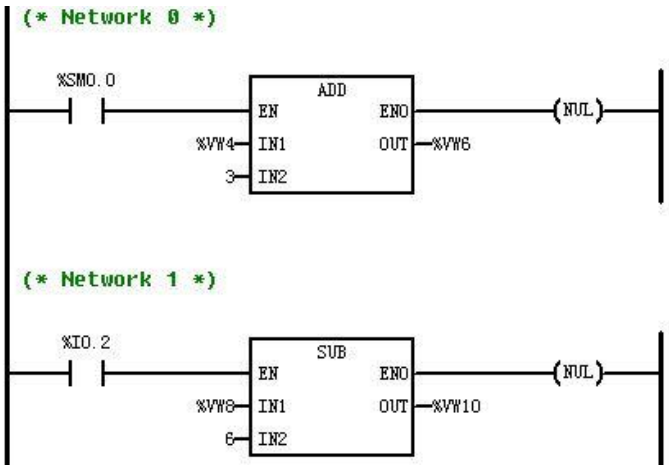
- **IL**

参数 *IN1*、*OUT* 的数据类型必须一致。

若 CR 值为 1，则指令被执行。其中，*ADD* 指令的功能是：*OUT*= *OUT* + *IN1*；*SUB* 指令的功能是：*OUT*= *OUT*－*IN1*。ADD、SUB 指令的执行不影响 CR 值。

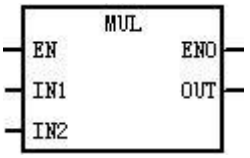
指令使用举例：

LD	IL

	(* Network 0 *)
	LD %SM0.0
	(* 建立 CR，其值为 1 *)
	MOVE %VW4, %VW6
	ADD 3, %VW6
	(*VW6 = VW4 + 3*)
	(* Network 1 *)
	LD %I0.2
	(* 建立 CR，其值为 I0.2 的值若 CR 为 1,怎执行 VW10 = VW8 – 6，若 CR 为 0，则不执行 SUB*)
	MOVE %VW8, %VW10
	SUB 6, %VW10

5.8.2 MUL（乘法）、DIV（除法）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	MUL		
	DIV		
IL	MUL	MUL *IN1, OUT*	U
	DIV	DIV *IN1, OUT*	

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
IN2	输入	INT、DINT、REAL	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
OUT	输出	INT、DINT、REAL	Q、AQ、M、V、L、SM、指针

- **LD**

参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

若 *EN* 值为 1，则指令被执行。其中，*MUL* 指令的功能是：*OUT*＝*IN1*×*IN2*；*DIV* 指令的功能是：*OUT*＝*IN1*÷*IN2*。

- **IL**

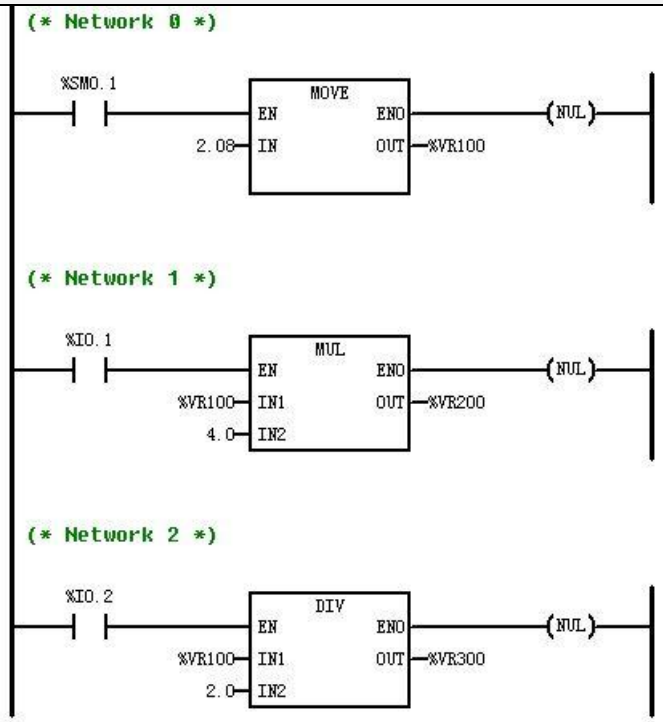
参数 *IN1*、*OUT* 的数据类型必须一致。

如果 CR 值为 1，则指令被执行。其中，*MUL* 指令的功能是：*OUT*＝*OUT*×*IN1*；*DIV* 指令的功能是：*OUT*＝*OUT*÷*IN1*。MUL、DIV 指令的执行不影响 CR 值。

注意： DIV 指令，如果除以 0 时，特殊寄存器 SM1.3 会置位。

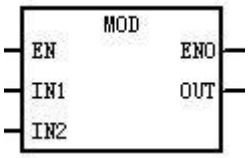
指令使用举例：

LD	IL

	(* Network 0 *)
	LD %SM0.1
	MOVE 2.08, %VR100
	(* Network 1 *)
	LD %I0.1
	MOVE %VR100, %VR200
	MUL 4.0, %VR200
	(*若 I0.1 为 1 则执行 VR200= VR100*4.0*)
	(* Network 2 *)
	LD %I0.2
	MOVE %VR100, %VR300
	DIV 2.0, %VR300
	(*若 I0.2 为 1 则执行 VR300= VR100÷2.0*)

5.8.3 MOD（求余数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	MOD		
IL	MOD	MOD *IN1*, *OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN1	输入	INT、DINT	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
IN2	输入	INT、DINT	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针
OUT	输出	INT、DINT	Q、AQ、M、V、L、SM、指针

- **LD**

  参数 *IN1*、*IN2*、*OUT* 的数据类型必须一致。

  若 *EN* 值为 1，则该指令被执行：将 *IN1* 除以 *IN2*，并将余数赋给 *OUT*。

- **IL**

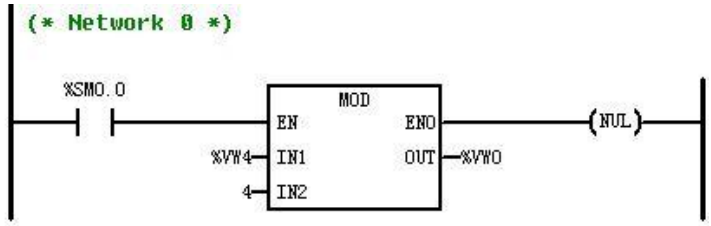
参数 *IN1*、*OUT* 的数据类型必须一致。

如果 CR 值为 1，则该指令被执行：将 *OUT* 除以 *IN1*，并将余数赋给 *OUT*。

MOD 指令的执行不影响 CR 值。

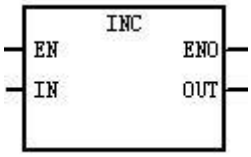
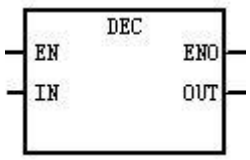
注意：如果指令执行时，尝试除以 0，特殊寄存器 SM1.3 会置位。

指令使用举例：

LD	IL
	(* Network 0 *) LD %SM0.0 MOVE %VW4, %VW0 MOD 4, %VW0 (*将 VW4 除以 4，得到的余数赋给 VW0*)
举例结果如下：	
VW4 值： 9 执行 MOD 指令后 VW0 的值： 1	

5.8.4 INC（加 1）、DEC（减 1）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	INC		
	DEC		
IL	INC	INC *OUT*	U
	DEC	DEC *OUT*	

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、INT、DINT	I、Q、AI、AQ、M、V、L、SM、T、C、HC、常量、指针

OUT	输出	BYTE、INT、DINT	Q、AQ、M、V、L、SM、指针

- **LD**

参数 *IN*、*OUT* 的数据类型必须一致。

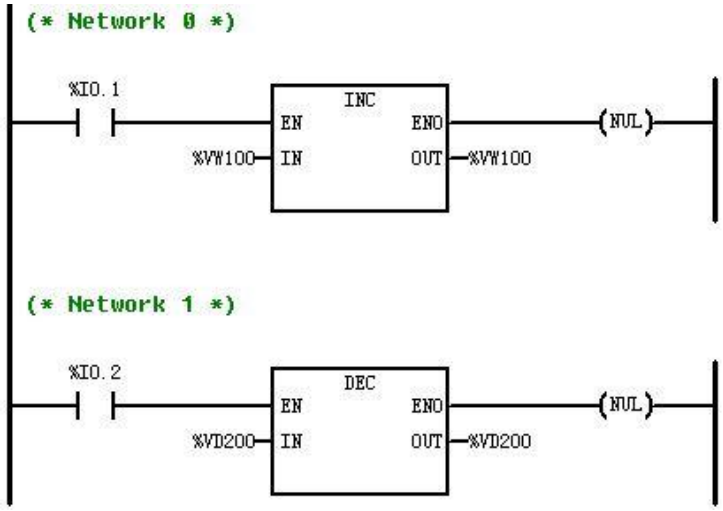
若 *EN* 值为 1，则指令被执行。其中，*INC* 指令的功能是：*OUT*＝*IN*＋1；*DEC* 指令的功能是：*OUT*＝*IN*－1。

- **IL**

如果 CR 值为 1，则指令被执行。其中，*INC* 指令的功能是：*OUT*＝*OUT*＋1；*DEC* 指令的功能是：*OUT*＝*OUT*－1。

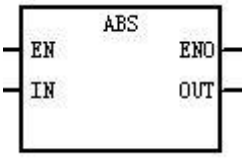
INC、DEC 指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.1 INC %VW100 (*若 I0.1 为 1 则执行：VW100 =VW100＋I#1 ,若 I0.1 为 0 则不执行 INC 指令*) (* Network 1 *) LD %I0.2 DEC %VD200 (*若 I0.2 为 1 则执行：VD200 =VD200-DI#1 ,若 I0.2 为 0 则不执行 DEC 指令*)

5.8.5 ABS（绝对值）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ABS		
IL	ABS	ABS *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	INT、DINT、REAL	I、Q、V、M、L、SM、T、C、AI、AQ、HC、常量、指针
OUT	输出	INT、DINT、REAL	Q、V、M、L、SM、AQ、指针

参数 *IN*、*OUT* 的数据类型必须一致。

该指令将输入 *IN* 求绝对值并将结果赋给 *OUT*，如下式所示：*OUT*＝|*IN*|。

- **LD**

  若 *EN* 值为 1，则该指令被执行。

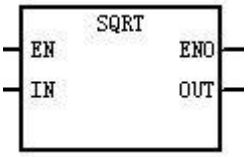
- **IL**

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

5.8.6 SQRT（平方根）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	SQRT		
IL	SQRT	SQRT *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

该指令将输入 *IN* 开平方并将结果赋给 *OUT*，如下式所示：*OUT*＝$\sqrt{IN}$ 。

- **LD**

  若 *EN* 值为 1，则该指令被执行。

- **IL**

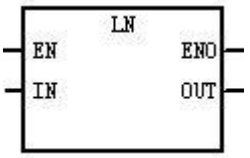
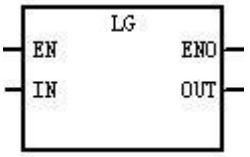
  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

注意：如果 IN 为负数，指令不执行，且特殊寄存器 SM1.5 置位。

5.8.7 LN（自然对数）、LG（常用对数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	LN		
	LG		
IL	LN	LN *IN, OUT*	U
	LG	LG *IN, OUT*	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

LN 指令将输入 *IN* 求自然对数并将结果赋给 *OUT*，如下式所示：$OUT = \log_e(IN)$。

LG 指令将输入 *IN* 求常用对数并将结果赋给 *OUT*，如下式所示：$OUT = \log_{10}(IN)$。

- **LD**

若 *EN* 值为 1，则指令被执行。

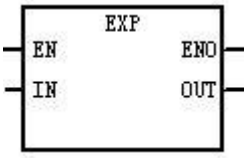
- **IL**

如果 CR 值为 1，则指令被执行。

LN、LG 指令的执行不影响 CR 值。

5.8.8 EXP（以 e 为底的指数）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	EXP		
IL	EXP	EXP *IN, OUT*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

该指令将输入 *IN* 求以 e 为底的指数并将结果赋给 *OUT*，如下式所示：*OUT*＝e^{IN}。

- **LD**

  若 *EN* 值为 1，则该指令被执行。

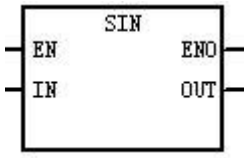
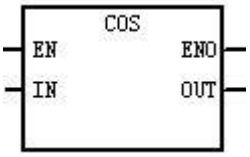
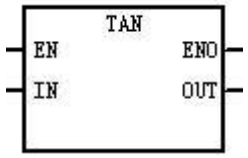
- **IL**

  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

5.8.9 SIN（正弦）、COS（余弦）、TAN（正切）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	SIN		
	COS		
	TAN		
IL	SIN	SIN *IN, OUT*	U
	COS	COS *IN, OUT*	
	TAN	TAN *IN, OUT*	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

输入 *IN* 表示的是弧度值（将角度从度数转换为弧度需要乘以 π /180）。

SIN 指令将 *IN* 求正弦并将结果赋给 *OUT*，如下式所示：*OUT*＝SIN(*IN*)。

COS 指令将 *IN* 求余弦并将结果赋给 *OUT*，如下式所示：*OUT*＝COS(*IN*)。

TAN 指令将 *IN* 求正切并将结果赋给 *OUT*，如下式所示：*OUT*＝TAN(*IN*)。

- **LD**

  若 *EN* 值为 1，则指令被执行。

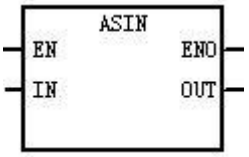
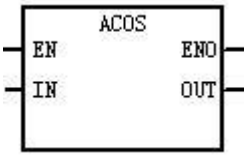
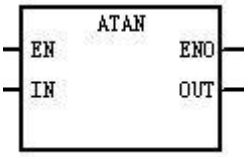
- **IL**

  如果 CR 值为 1，则指令被执行。

  SIN、COS、TAN 指令的执行不影响 CR 值。

5.8.10 ASIN（反正弦）、ACOS（反余弦）、ATAN（反正切）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ASIN		
	ACOS		
	ATAN		
IL	ASIN	ASIN *IN, OUT*	U
	ACOS	ACOS *IN, OUT*	
	ATAN	ATAN *IN, OUT*	

参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L、常量、指针
OUT	输出	REAL	V、L、指针

ASIN 指令将 *IN* 求反正弦并将结果赋给 *OUT*，如下式所示：*OUT*＝ARCSIN(*IN)*。

ACOS 指令将 *IN* 求余弦并将结果赋给 *OUT*，如下式所示：*OUT*＝ARCCOS(*IN)*。

ATAN 指令将 *IN* 求正切并将结果赋给 *OUT*，如下式所示：*OUT*＝ARCTAN(*IN)*。

注意：结果是弧度值。

- **LD**

  若 *EN* 值为 1，则指令被执行。

- **IL**

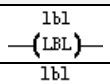
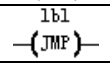
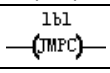
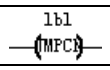
  如果 CR 值为 1，则指令被执行。

  ASIN、ACOS、ATAN 指令的执行不影响 CR 值。

5.9 程序控制

5.9.1 标号及跳转指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	LBL		
	JMP		
	JMPC		
	JMPCN		
IL	标号	*lbl:*	U
	JMP	JMP *lbl*	
	JMPC	JMPC *lbl*	
	JMPCN	JMPCN *lbl*	

参数	描 述
lbl	合法的标识符

注意：跳转指令中指定的 *lbl* 标号必须存在而且必须与该指令在同一程序中。

- **LD**

LBL 指令用于在当前位置定义一个标号，该标号将作为跳转指令的目的地。标号不允许重复定义。LBL 指令是无条件执行的，不建议用户在 LBL 指令的左端加入任何元件。实际上，在编译时编译器会将 *LBL* 指令的左端的所有元件忽略掉。

JMP 指令用于无条件地将程序跳转到 *lbl* 标号处继续执行。

JMPC 指令的作用是：若指令左侧能量流的值为 1，则将程序跳转到 *lbl* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

JMPCN 指令的作用是：若指令左侧能量流的值为 0，则将程序跳转到 *lbl* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

- **IL**

标号的定义格式是：*lbl:*。标号的定义占用独立的一行。标号不允许重复定义。

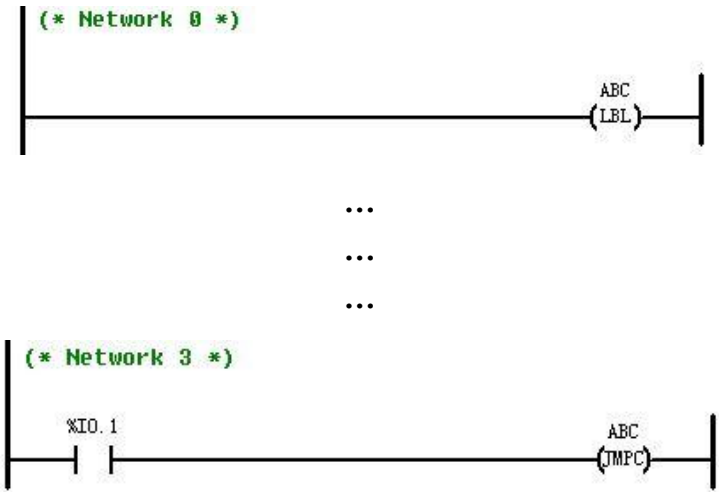
JMP 指令用于无条件地将程序跳转到 *lbl* 标号处继续执行。

JMPC 指令的作用是：若 CR 值为 1，则将程序跳转到 *lbl* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

JMPCN 指令的作用是：若 CR 值为 0，则将程序跳转到 *lbl* 标号处继续执行，否则该指令不起作用，程序继续向下依次执行。

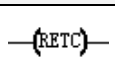
注意：跳转指令的执行不影响 CR 值，因此必要时用户应该在跳转的目的标号之后重新建立新的 CR 值，以免程序执行出现错误。

指令使用举例：

LD	IL
	(* Network 0 *) ABC： … … … (* Network 3 *) LD %I0.1 JMPC ABC

5.9.2 返回指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	RETC		
	RETCN		
IL	RETC	RETC	U
	RETCN	RETCN	

返回指令只能在子程序和中断服务程序中使用，用于终止所在程序并返回到该程序的调用点继续执行。

在每个子程序和中断服务程序的结尾，EuraProg 软件都自动地隐含调用了 RETC 指令。

- **LD**

RETC 指令：若指令左侧能量流的值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

RETCN 指令：若指令左侧能量流的值为 0，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

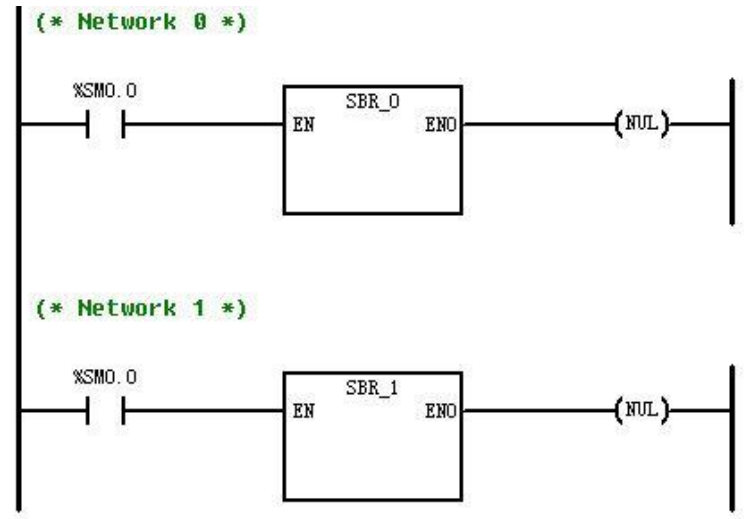
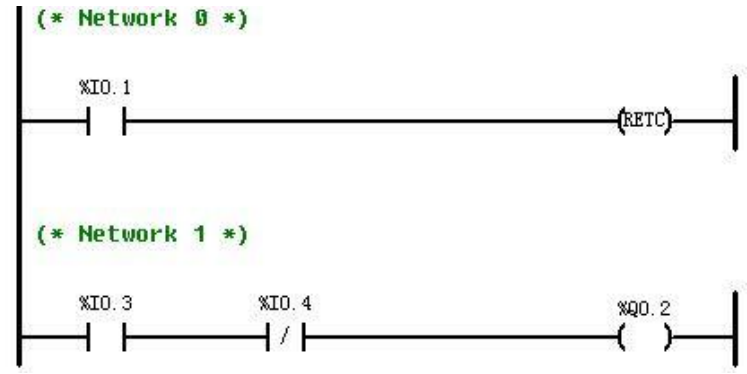
- **IL**

RETC 指令：若 CR 值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

RETCN 指令：若 CR 值为 0，则该指令执行，否则该指令不起作用，程序继续向下依次执行。

注意：另外，返回指令的执行不影响 CR 值，因此必要时用户应该在程序的返回点之后重新建立新的 CR 值，以免程序执行出现错误。

指令使用举例：

LD	IL
MAIN：  SBR_0：  	MAIN： (* Network 0 *) LD %SM0.0 CAL SBR_0 (* Network 1 *) LD %SM0.0 CAL SBR_1 SBR_0： (* Network 0 *) LD %I0.1 (* 建立 CR，其值为 I0.1 的值 *) RETC (* 若 CR 为 1：则返回到主程序中并继续执行 CAL SBR_1 指令；若 RETC 指令没起作用，则该指令及后续指令得以执行 *) (* Network 1 *) LD %I0.3 ANDN %I0.4 ST %Q0.2

5.9.3 CAL（调用子程序）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	CAL	子程序名 EN ENO IN1 OUT1 IN_OUT1	
IL	CAL	CAL 子程序名，子程序实参 1， 子程序实参 2，…	U

该指令用于调用并执行指定名称的子程序，子程序执行完成后，将返回至 CAL 之后的指令继续执行。CAL 指令调用的子程序在用户工程中必须已经存在。

在 CAL 指令中输入的实参必须与被调用子程序的形参（在该子程序的局部变量表中定义）的

数据类型和变量类型相匹配。用户必须依照下列次序来输入子程序的实参：所有的输入参数、所有的输入/输出参数、所有的输出参数。

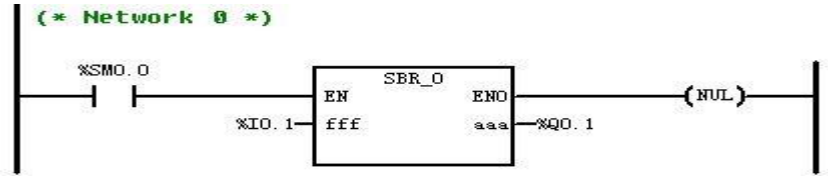
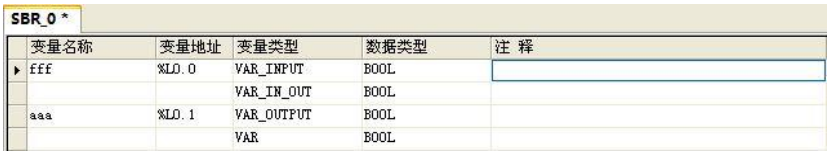
- **LD**

若用户在工程中编写了一个子程序，则该子程序的名字将出现在指令树的【**子程序**】组中，双击这个名字就可以在程序中相应的位置加入该子程序的调用指令。若调用指令左侧的能量流的值为 1，则该子程序就被调用并执行。

- **IL**

若 CR 值为 1，则调用并执行指定的子程序，否则该指令不起作用，程序继续向下依次执行。CAL 指令的执行不影响 CR 值，但是需要注意的是：CR 值可能在子程序中发生了变化。

➢ 指令使用举例

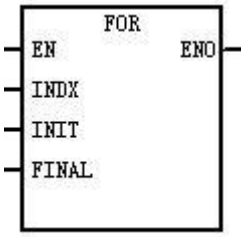
LD	IL
MAIN：  **子程序 SBR_0 中的局部变量表：** 	**MAIN：** (* Network 0 *) LD %SM0.0 CAL SBR_0 %I0.1, %Q0.1

SBR_0 *

变量名称	变量地址	变量类型	数据类型	注 释
fff	%L0.0	VAR_INPUT	BOOL	
		VAR_IN_OUT	BOOL	
aaa	%L0.1	VAR_OUTPUT	BOOL	
		VAR	BOOL	

5.9.4 FOR/NEXT（循环指令）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	FOR		
	NEXT	—(NEXT)—	
IL	FOR	FOR *INDX, INIT, FINAL*	U
	NEXT	NEXT	

参数	输入/输出	数据类型	允许的内存区
INDX	输入	INT	M、V、L、SM
INIT	输入	INT	M、V、L、SM、T、C、常量
FINAL	输出	INT	M、V、L、SM、T、C、常量

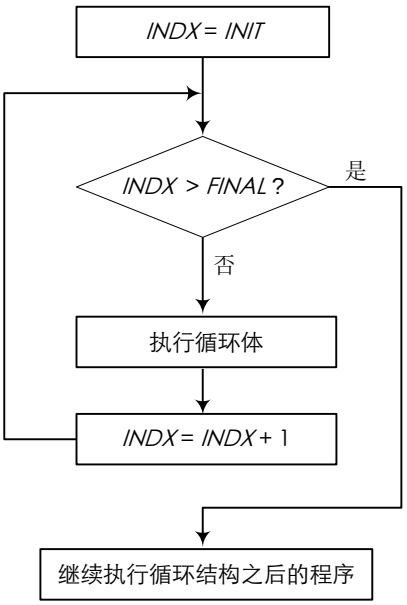
FOR/NEXT 用来实现循环，适用于循环次数已经确定的情况。

FOR 指令和 NEXT 指令必须成对使用，其中 FOR 标记了循环的开始，NEXT 标记了循环的结束。在 FOR 和 NEXT 之间的语句被称为循环体。FOR 指令和与其匹配的 NEXT 指令以及两者之间的循环体共同组成了一个完整的循环结构。

FOR/NEXT 指令反复执行循环体内的语句直至达到指定的循环次数，其中，循环次数的计数值存放在参数 *INDX* 内，而 *INIT* 指定了 *INDX* 的初始值，*FINAL* 指定了 *INDX* 的终值。

一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。FOR/NEXT 循环支持嵌套，嵌套深度最大为 8 层。

FOR/NEXT 循环的执行过程如下图：



使用 FOR/NEXT 循环时，用户需要注意如下方面：

✧ FOR 指令必须是一个网络（Network）中的第二条指令；

NEXT 指令必须单独占用一个网络（Network）。

✧ 在循环体内，用户可以改变终值 *FINAL*，从而改变循环的结束条件。

✧ 若循环次数比较多，则程序的执行时间可能就会超过系统看门狗的定时值，从而导致 CPU 扫描超时的故障；

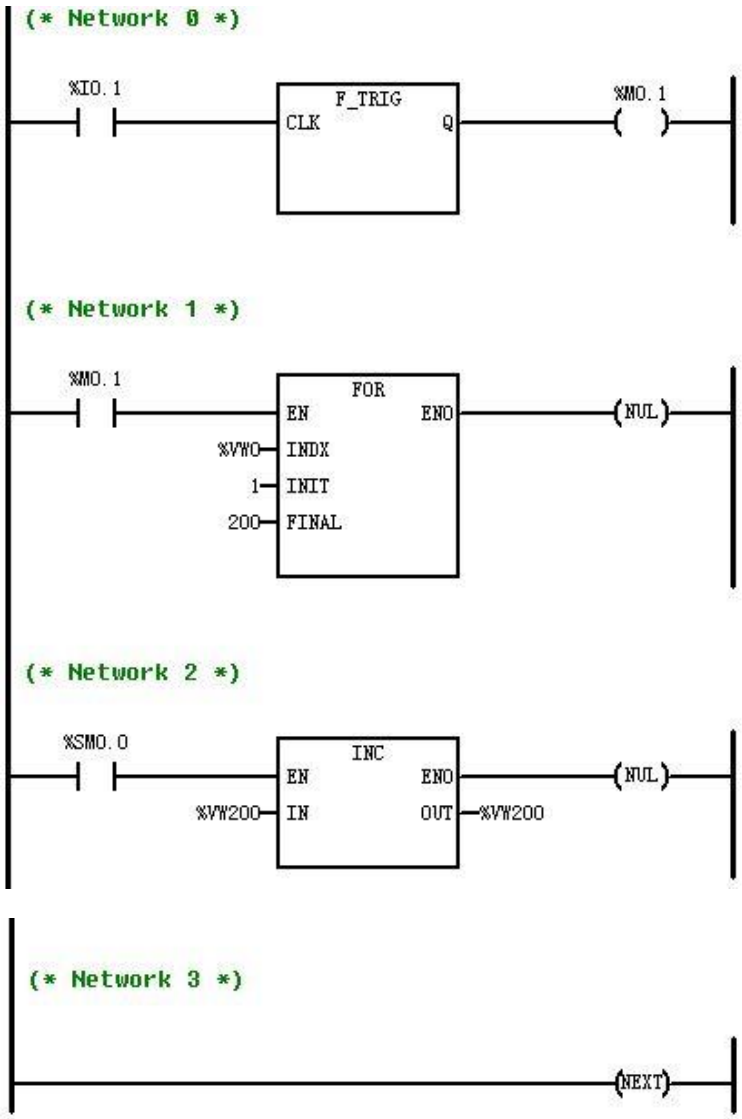
- **LD**

若 FOR 指令左侧能量流的值为 1，则该 FOR/NEXT 循环被执行，否则将跳过该循环，继续向下执行循环结构之后的程序。

- **IL**

若 FOR 指令处的 CR 值为 1，则该 FOR/NEXT 循环被执行，否则将跳过该循环，继续向下执行循环结构之后的程序。

指令使用举例：

LD	IL
	(* Network 0 *) LD %I0.1 F_TRIG ST %M0.1 (* Network 1 *) (*若检测到 I0.1 的下降沿，则下面的循环体被执行 200 次*) LD %M0.1 FOR %VW0, 1, 200 (* Network 2 *) LD %SM0.0 INC %VW200 (* Network 3 *) LD TRUE NEXT

5.9.5 END（终止主程序）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	END	—(END)—	
IL	END	END	U

该指令只能在主程序中使用，用于终止主程序的执行。

在主程序的结尾，EuraProg 软件自动地隐含调用了 END 指令。

- **LD**

若指令左侧能量流的值为 1，则该指令被执行，否则该指令将不起作用，程序继续向下依次执行。

- **IL**

若 CR 值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

5.9.6 STOP（停止 CPU）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	STOP	—(STOP)—	
IL	STOP	STOP	U

该指令将 CPU 从运行（RUT）状态立即转变为停止（STOP）状态。

- **LD**

若指令左侧能量流的值为 1，则该指令被执行，CPU 立即被转入停止（STOP）状态，否则该指令将不起作用，程序继续向下依次执行。

- **IL**

若 CR 值为 1，则该指令被执行，CPU 立即被转入停止（STOP）状态，否则该指令不起作用，程序继续向下依次执行。

5.9.7 WDR（看门狗复位）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	WDR	—(WDR)—	
IL	WDR	WDR	U

该指令将系统的看门狗复位并重新启动看门狗定时器。

使用 WDR 指令可以增加一次扫描所允许的时间，从而使耗时较长的程序能够顺利执行下去。但需要用户注意的是，如果程序耗时过长，那么下述任务就有可能不会被及时完成

✧CPU 自诊断

✧读输入（读取物理输入通道的信号并刷新输入映像区）

✧通讯

✧写输出（此处指的是将输出映像区中的数据写至物理输出通道，不包括立即输出指令）

✧10ms 和 100ms 定时器的计时

- **LD**

若指令左侧能量流的值为 1，则该指令被执行，否则该指令将不起作用，程序继续向下依次执行。

- **IL**

若 CR 值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

5.10 中断指令

采用中断技术的目的是提高 CPU 的执行效率，实现对内部或者外部各种预定义中断事件的快速响应。EC100/EC200 系列 PLC 定义了数十种中断事件，每一种中断都被指定了唯一的事件号以供 CPU 识别。

5.10.1 EC100/EC200 如何处理中断事件

用户可以在程序中调用 ENI、DISI 指令来全局允许、禁止 CPU 处理中断事件。EC100/200 默认全局允许处理中断事件。

若用户需要 CPU 响应某个中断事件，则必须在程序中使用 ATCH（中断连接）指令将该中断事件（用事件号进行标识）与一个中断服务程序连接起来。这样当该中断事件发生时， CPU 将自动调用一次它连接的中断服务程序进行处理。一旦中断服务程序中的最后一条指令执行完成，CPU 将返回到主循环的断点处继续执行。用户也可以在中断服务程序中调用 RETC 或者 RETCN 指令来退出该程序。

一个中断事件只能对应一个中断服务程序，但一个中断服务程序可以对应多个中断事件。中断服务程序没有参数，但允许使用局部变量。

由于有快速响应的要求，因此用户应当尽量优化中断服务程序，使其短小精干。

5.10.2 中断优先级和队列

中断事件各有不同的优先级。当中断事件发生时会按照优先级和发生的时序进行排队：队列中优先级高的中断事件优先得到处理；优先级相同的中断按照发生的时序进行处理，先发生的就先进行处理。

任何时刻都只允许有一个中断服务程序在执行。一旦一个中断服务程序开始执行，它会一直执行完成，而不会被其它任何中断服务程序打断，在它执行期间发生的任何中断事件都会进入队列等候处理。

5.10.3 中断事件分类

- 通讯口中断

  用于 Port0、Port1 的自由协议通讯。这一类中断的优先级最高。

  发送完成中断在发送完用户指定的数据时发生。

  接收完成中断在接收到用户指定数量的数据时发生。

- I/O 中断

  包含了上升沿或下降沿中断、高速计数器中断和脉冲输出（PTO）中断。它们的优先级中等。

  上升沿、下降沿中断只能由 CPU 本体集成 DI 的第 0 至 3 通道（地址为 I0.0---I0.3）捕获。

  高速计数器中断在计数器复位、改变计数方向或者计数值等于预设值时发生。

  PTO 中断是在指定数目的脉冲输出完成时发生。它的一个典型应用是控制步进电机。

- 时间中断

  包含了定时中断和定时器中断。这一类中断的优先级最低。

  定时中断周期性的产生，周期单位为 ms，可以利用它来完成周期性的任务。

  定时器中断在 T2、T3 的当前值等于预设值时发生，可以利用它来及时响应定时事件。

5.10.4 中断事件列表

下表是 EC100/EC200 的中断事件的完整列表。

需要注意的是，对于具体的 CPU 型号来说，由于不具备某些功能，所以相应的中断事件也就不支持。

表 5-1　EC100/EC200 的中断事件列表

事　件号	中断描述	分类	优先级
32	Port1：发送数据完成	通讯中断 (高优先级)	最高
31	Port1：接收数据完成		
30	Port0：发送数据完成		
29	Port0：接收数据完成		
28	PTO 0 完成	I/O 中断 (中等优先级)	
27	PTO 1 完成		
26	I0.4 检测到下降沿		
25	I0.4 检测到上升沿		
24	I0.5 检测到下降沿		
23	I0.5 检测到上升沿		
22	I0.6 检测到下降沿		
21	I0.6 检测到上升沿		
20	I0.7 检测到下降沿		
19	I0.7 检测到上升沿		
18	HSC0　CV=PV（当前值=预设值）		
17	HSC0　输入方向改变		

16	HSC0　外部复位		
15	HSC1　CV=PV（当前值=预设值）		
12	HSC2　CV=PV（当前值=预设值）		
11	HSC2　输入方向改变		
10	HSC2　外部复位		
9	HSC3　CV=PV（当前值=预设值）		
4	定时中断 1。周期在 SMW20 中指定，单位为 ms。周期范围为 1~65535ms 。	时间中断 (低优先级)	
3	定时中断 0。周期在 SMW18 中指定，单位为 ms。周期范围为 1~65535ms 。		
2	定时器 T3　ET=PT（当前值=预设值）		
1	定时器 T2　ET=PT（当前值=预设值）		最低

5.10.5　ENI（允许中断）、DISI（禁止中断）指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ENI	—(ENI)—	
	DISI	—(DISI)—	
IL	ENI	ENI	U
	DISI	DISI	

ENI、DISI 指令在程序中执行一次即可。PLC 默认是允许程序处理中断事件的。

- **LD**

ENI 指令的作用是：若指令左端能量流的值为 1，则全局允许处理中断事件（即当中断发生时 CPU 允许调用中断服务程序），否则不执行该指令。

DISI 指令的作用是：若指令左端能量流的值为 1，则全局禁止处理中断事件，否则不执行该指令。

- **IL**

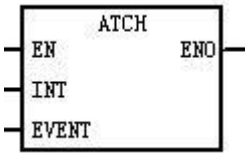
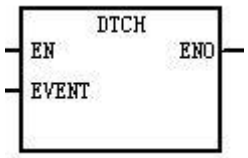
ENI 指令的作用是：若 CR 值为 1，则全局允许处理中断事件，否则不执行该指令。

DISI 指令的作用是：若 CR 值为 1，则全局禁止处理中断事件，否则不执行该指令。

ENI、DISI 指令的执行均不影响 CR 值。

5.10.6　ATCH（中断连接）、DTCH（中断分离）指令

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	ATCH		
	DTCH		
IL	ATCH	ATCH *INT, EVENT*	U
	DTCH	DTCH *EVENT*	

参数	输入/输出	数据类型	参数描述
INT	输入	标识符	用户工程中已存在的中断服务程序的名称
EVENT	输入	INT 型常量	中断事件号

ATCH 指令的功能是：将 *EVENT* 指定的中断事件与 *INT* 指定的中断服务程序连接起来，并进行初始化。这样若 CPU 允许处理中断事件，则当该中断事件发生时，将自动调用该中断服务程序。多个中断事件可以连接一个中断服务程序，但一个中断事件不允许与多个中断服务程序连接。

DTCH 指令实现的功能是：取消参数 *EVENT* 指定的中断事件与所有中断服务程序的连接。

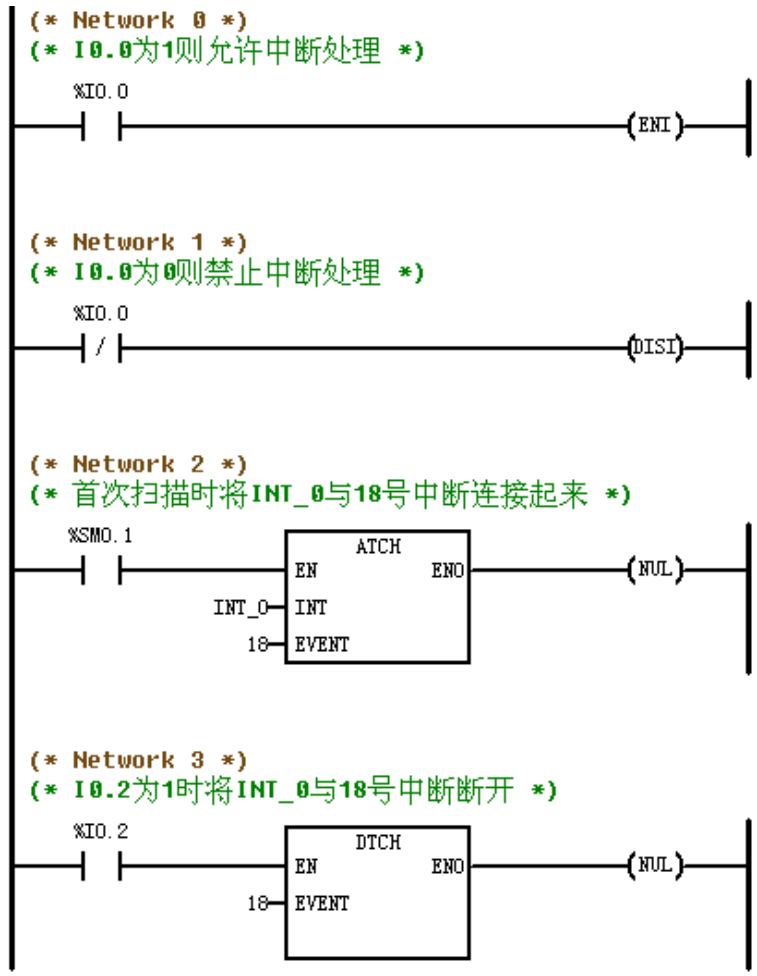
- **LD**

  若 *EN* 值为 1，则 *ATCH*、*DTCH* 指令 被执行，否则不执行。

- **IL**

  如果 CR 值为 1，则 *ATCH*、*DTCH* 指令被执行，否则不执行。它们的执行不影响 CR 值。

指令使用举例

LD	IL
	(* Network 0 *) (*I0.0 为 1 则允许中断处理*) LD %I0.0 ENI (* Network 1 *) (*I0.0 为 0 则禁止中断处理*) LDN %I0.0 DISI (* Network 2 *) (*首次扫描时将 INT_0 与 18 号中断连接起来*) LD %SM0.1 ATCH INT_0, 18 (* Network 3 *) (*I0.2 为 1 时将 INT_0 与 18 号中断断开*) LD %I0.2 DTCH 18

5.11 实时时钟

CPU 本体内置一个实时时钟(RTC)，可提供实时的时间/日历表示。实时时钟/日历的秒至年采用 BCD 格式编码，自动进行闰年调整。断电时，实时时钟使用后备电池供电。常温下，后备的时间累计不低于 50000 小时。

5.11.1 调整 CPU 时钟

实时时钟在正式使用之前必须进行至少一次时间调整，将其时间调整为当前的实际时间。在调整之前，PLC 内的时间值可能是一个随机值。

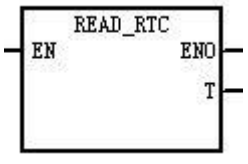
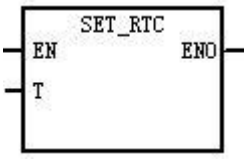
执行【**PLC**】→【**调整 PLC 时钟**…】菜单命令进入“调整 PLC 时钟”对话框，如下图。

图 5-1 调整 CPU 时钟

- ✧ **系统当前时间**：显示编程所用 PC 机当前的日期、时间。
- ✧ **PLC 当前时间**：显示所连 CPU 内实时时钟的当前时间。若背景色是绿色，代表成功地从 CPU 内读到了实时时钟。 若背景色是黄色，则代表从 CPU 内读取实时时钟失败。
- ✧ **将 PLC 时间调整为**：用户可以在此输入将要设置的 CPU 实时时钟的新时间值。用户既可以在输入框内直接键盘输入，也可以使用鼠标左键单击输入框右端的箭头然后进行选择、调整。
- ✧ 单击【**修改 PLC 时钟**】按钮，则用户输入的新的时间值将被写入 CPU 内的实时时钟中。

5.11.2 READ_RTC（读实时时钟）、SET_RTC（设置实时时钟）

- ➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	READ_RTC		
	SET_RTC		
IL	READ_RTC	READ_RTC *T*	U
	SET_RTC	SET_RTC *T*	

参数	输入/输出	数据类型	允许的内存区
T	输入（SET_RTC）	BYTE	V
	输出（READ_RTC）		

READ_RTC 指令用于从硬件时钟中读取当前的日期和时间并放入时间缓冲区中。

SET_RTC 指令用于将时间缓冲区指定的日期和时间写入硬件时钟。

参数 *T* 定义了时间缓冲区的起始地址，该区域占用连续 8 个字节的内存空间。缓冲区内所有的数值均以 BCD 格式编码。时间缓冲区内的数据存储形式如下表所示。

表 5-2　实时时钟指令的时间缓冲区

内存字节	数据的含义	备注
T	星期	范围：1~7，其中 1 代表星期一，7 代表星期日。
T+1	秒	范围：0~59
T+2	分	范围：0~59
T+3	时	范围：0~23
T+4	日	范围：1~31
T+5	月	范围：1~12
T+6	年	范围：0~99
T+7	世纪	固定为 20

提示：

(1)　执行 EuraProg 中的【PLC】→【调整 PLC 时钟...】菜单命令也可以读写 CPU 时钟。建议用户在使用实时时钟之前首先通过上述命令为 CPU 设置好正确的时钟。

(2)　CPU 不检查用户输入的日期、时间是否有效，无效的日期（比如 2 月 30 日）也会被 CPU 接受，因此用户在设置实时时钟时必须确保输入有效的日期、时间。

- **LD**

  若 *EN* 值为 1，则执行 READ_RTC、SET_RTC 指令，否则不执行。

- **IL**

  若 CR 值为 1，则执行 READ_RTC、SET_RTC 指令，否则不执行。

  READ_RTC、SET_RTC 指令的执行不影响 CR 值。

5.12　通讯指令

5.12.1　自由通讯指令

本节介绍的指令用于自由通讯。所谓的自由通讯，是指 CPU 串行通讯口的通讯过程完全由用户程序进行控制。自由通讯方式支持 ASCII 和二进制数据通讯。用户可以使用自由通讯方式来编写各种自定义的通讯协议与其它设备进行通讯。

EC100/EC200 系列 CPU 本体集成了 2 个串行通讯口，这些串口默认采用 Modbus RTU 协议

并作为从站。当执行用户程序中的自由通讯指令时，自由通讯方式就被激活，通讯口完全被自由通讯占用。当自由通讯完成后，CPU 又自动将通讯口切换到 Modbus RTU 协议。若 CPU 处于 STOP 状态，则自由通讯被禁止。

下面简要描述了用户进行自由通讯编程的总体步骤：

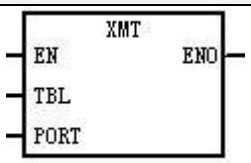
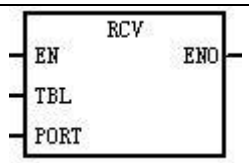
1) 在特殊寄存器 SMB30 和 SMB130 中设置所用通讯口的串行通讯参数寄存器。使用 XMT、RCV 指令时，按照串行通讯参数设置进行通讯。

2) 设置自由通讯控制寄存器（详见控制寄存器的定义）。注意：控制寄存器必须先设置好。

3) 调用 XMT、RCV 指令并配合自由通讯的状态寄存器、通讯中断进行编程。

5.12.1.1 XMT（发送数据）、RCV（接收数据）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	XMT		
	RCV		
IL	XMT	XMT *TBL, PORT*	U
	RCV	RCV *TBL, PORT*	

参数	输入/输出	数据类型	允许使用的内存区
TBL	输入	BYTE	I、Q、M、V、L、SM
PORT	输入	INT	常量（0 或 1）

XMT 指令用于发送存放在数据缓冲区中的数据。参数 *PORT* 定义了所用通讯口 0 或者 1。参数 *TBL* 定义了数据缓冲区的起始地址，缓冲区的第一个字节中定义了本次将要发送的字节数（1--255），后边依次存放着待发送的数据字节。若发送字节数被设置为 0，则 *XMT* 指令不执行任何操作。

RCV 指令用于接收数据并将接收到的数据存放在数据缓冲区中。参数 *PORT* 定义了所用通讯口。参数 *TBL* 定义了数据缓冲区的起始地址，缓冲区的第一个字节中存放着本次接收到的字节数，后边依次存放着接收到的有效数据字节。

- **LD**

  若 *EN* 值为 1，则执行 XMT、RCV 指令，否则不执行。

- **IL**

  若 CR 值为 1，则执行 XMT、RCV 指令，否则不执行。这些指令的执行不影响 CR 值。

➢ 自由通讯的状态寄存器及控制寄存器

EC100/EC200 在 SM 区中为自由通讯提供了多个状态寄存器和控制寄存器。在编写通讯程序时，用户必须对这些控制寄存器进行设置。另外，在通讯过程中 CPU 会自动对通讯状态进行检测，并将检测结果写入相关的状态寄存器，用户可以读取这些状态信息并在程序中进行相应的处理。

(1) SMB30 和 SM130（自由口通讯参数字节）

位（只读）		描述
Port 0	**Port 1**	
SMB30 的格式	SMB130 的格式	自由端口模式控制字节 MSB7---LSB0：xxpp dbbb
SM30.0---SM30.2	SM130.0---SM130.2	bbb: 自由端口波特率(bps) 000 : 1200 001: 2400 010 : 4800 011 : 9600 100 : 19200 101 : 38400 110 : 57600 111: 115200
SM30.3	SM130.3	d : 数据位数 0 : 8 位(1 位停止位) 1 : 7 位(2 位停止位)
SM30.4、SM30.5	SM130.4、SM130.5	pp: 奇偶校验位 00 : 无校验 01 : 偶校验 10 : 奇校验
SM30.6、SM30.7	SM130.6、SM130.7	xx:保留

(2) SMB86 和 SM186（接收状态字节）

位（只读）		值	含 义
Port0	**Port1**		
SM86.0	SM186.0	1	终止接收：接收到的字符存在奇偶校验错误。
SM86.1	SM186.1	1	终止接收：达到最大接收字节数。
SM86.2	SM186.2	1	终止接收：接收一个字符超时。
SM86.3	SM186.3	1	终止接收：系统接收超时。
SM86.4	SM186.4	-	保留。
SM86.5	SM186.5	1	终止接收：接收到了用户定义的结束字符。
SM86.6	SM186.6	1	终止接收：参数设置错误，无起始条件接收或者无接收结束条件等。
SM86.7	SM186.7	1	终止接收：用户使用了禁止接收命令。

(3) SMB87 和 SMB187（接收控制字节）

位		值	描 述
Port0	**Port1**		
SM87.0	SM187.0	-	保留。

SM87.1	SM187.1	0	当发送或者接收完成时禁止生成相应的中断。
		1	当发送或者接收完成时允许生成相应的中断。
SM87.2	SM187.2	0	忽略 SMW92/SMW192 中用户定义的接收字符超时值。
		1	使用 SMW92/SMW192 中用户定义的接收字符超时值。
SM87.3	SM187.3	-	保留。
SM87.4	SM187.4	0	忽略 SMW90/SMW190 中用户定义的接收准备时间检测。
		1	使用 SMW90/SMW190 中用户定义的接收准备时间检测。
SM87.5	SM187.5	0	忽略 SMB89/SMW189 中用户定义的接收结束字符。
		1	使用 SMB89/SMW189 中用户定义的接收结束字符。
SM87.6	SM187.6	0	忽略 SMB88/SMW188 中用户定义的接收起始字符。
		1	使用 SMB88/SMW188 中用户定义的接收起始字符。
SM87.7	SM187.7	0	禁止接收数据。此禁止条件优先于其它所有的接收控制字。
		1	允许接收数据。

(4) 其它控制寄存器

控制字（字节）		描　述
Port0	**Port1**	
SMB88	SMB188	用于存放用户定义的接收起始字符。 执行 RCV 指令后，CPU 收到了起始字符就开始进入有效接收状态，之前收到的数据都会被丢弃。CPU 将起始字符作为接收的第一个有效数据并保存在接收缓冲区。 如果要使本设置值生效，需要将 SM87.6/SM187.6 置为 1。
SMB89	SMB189	用于存放用户定义的接收结束字符。 CPU 将以该字符作为接收的最后一个字节。收到该字符后，无论其它的结束条件如何 CPU 都会立刻终止接收状态。 如果要使本设置值生效，需要将 SM87.5/SM187.5 置为 1。
SMW90	SMW190	用于存放用户定义的接收准备时间（范围为 0~65535ms）。 执行 RCV 指令后，经过了该时间值，CPU 将自动进入有效接收状态而不管是否收到起始字符，此后接收到的数据将被认为是有效数据。接收准备时间为 0 表示不需要接收准备，直接接收数据。 如果要使本设置值生效，需要将 SM87.4/SM187.4 置为 1。

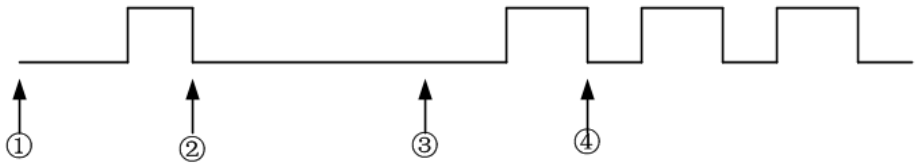
SMW92	SMW192	用于存放用户定义的接收字符超时值（范围为 1~65535ms）。 执行 RCV 指令并进入有效接收状态后，若在此超时时间内没有接收到任何字符，CPU 就会结束接收状态。 如果要使本设置值生效，需要将 SM87.2/SM187.2 置为 1。 若该设置为 0，则 RCV 指令直接结束执行。
SMB94	SMB194	用于存放用户定义的每次接收字符数（1~255）。 CPU 只要接收到了此数量个有效字符，无论其它的结束条件如何，都会马上终止接收状态。本设置值总是有效。 若该值设置为 0，则 RCV 指令将直接退出。

➢ 接收指令的启动和结束方式

RCV 指令使用自由通讯各控制寄存器数据来控制指令的启动、结束。

接收指令启动方式：

1) . 空闲检测：首先使能空闲检测控制位 SM87.4 或者 SM187.4，根据 SMW90 或者 SMW190 中指定的接收准备时间毫秒数。RCV 指令执行后，系统进行空闲检测，在达到接收准备时间之后，接收消息功能开始接收数据，接收到的第一个数据作为第一个有效数据。如果未达到接收准备时间，接收到数据，接收准备时间计时将进行重新计数。如下图所示。



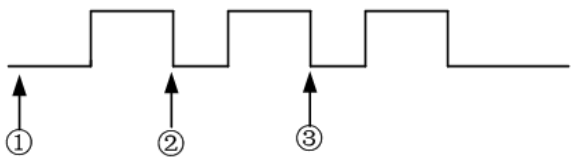
① 使能 SM87.4/SM187.4 接收准备时间检测，设置好接收准备时间 SMW90/SMW190，执行 RCV 指令，启动接收准备时间定时器。

② 接收准备时间定时未到，而接收到字符，未满足空闲检测，重新启动接收准备时间定时器。

③ 接收准备时间定时到，满足空闲检测，启动字符接收。

④ 将接收字符存入数据接收缓冲区。

2). 起始字符检测：首先使能起始字符检测控制位 SM87.5 或 SM187.5，根据 SMB88 或 SMB188 设置的起始字符进行检测字符，通讯口接收到与起始字符相同的字符作为缓冲区第一个字符。通讯口忽略之前接收的所有字符，将起始字符与之后接收的字符一并存入接收缓冲区。如下图所示。



① 使能起始字符检测控制位 SM87.5/SM187.5，设置好起始字符 SMB88/SMB188，执行 RCV 指令。

② 接收到的字符，不符合起始字符，接收计数器仍为 0，继续检测起始字符。

③ 接收到的字符是设定的起始字符，接收计数器计为 1，并将该字符存入接收缓冲区，根据结束条件是否接收接下来的字符。

3). 空闲与起始字符检测：空闲检测控制位 SM87.4 或者 SM187.4 与起始字符检测控制位 SM87.5 或 SM187.5 同时设置时执行空闲与起始字符检测。通讯口首先进行空闲检测。在空闲检测成功后，转入开始字符检测。如果此时检测到的数据不是起始字符，再重新依次进行空闲检测、起始字符检测，直到空闲检测与起始字符检测都成功。两个条件同时满足之前检测到的数据全部忽略。起始字符与之后接收的数据存入接收缓冲区。

4). 任意字符开始：如果接收指令没有使能空闲检测控制位 SM87.4 或者 SM187.4，而且空闲检测时间 SMW90 或者 SMW190 为 0。此时起始字符设置不起作用，通讯口接收 RCV 指令使能后的任意数据，字节存入接收缓冲区。

接收指令结束方式：

接收指令满足结束指令的任意一种，自由口接收指令都停止接收数据。

1). 结束字符检测：在使能结束字符检测功能时，当起始条件满足后，接收指令接收到结束字符后，立即结束接收。如果起始字符与结束字符设置为相同字符，则在接受到起始字符的同时结束接收。

2). 接收字符超时：在使能接收字符超时检测功能时，当起始条件满足后，如果在接收超时设置时间内没有接收到字符，接收立即停止。

3). 接收字符数检测：在使能接收字符数检测功能时，当接收的字符数满足后，接收立即停止。

4). 系统超时：如果 RCV 指令没有使能接收字符超时检测功能，则系统超时检测将开启，如果在系统超时时间 65535ms 内没有接收到数据，接收立即停止。

5). 用户控制停止：将 SM87.7 或者 SM187.7 允许接收数据位设置为 0，并执行一次 RCV 指令，接收将停止。

6). 奇偶校验错误：通讯口接收到的数据出现奇偶校验错误时，接收将停止。

➢ 关于通讯中断

EC100/EC200 提供了多种中断用于自由通讯，通讯中断具有最高的中断优先级。若用户需要了解更多关于中断的信息，请参阅 5.10.1 EC100/EC200 如何处理中断事件。

用户可以使用控制位 SM87.1 来禁止或允许 CPU 产生通讯中断。若将 SM87.1 设置为 1，则允许生成通讯中断：CPU 在发送完缓冲区中的最后一个字符时就会自动产生一个发送完成中断（对于 Port0 中断事件号为 30，对于 Port1 中断事件号为 32）；CPU 在退出接收后（无论是正常还是异常退出）就会自动产生一个接收完成中断（对于 Port0 中断事件号为 29，对于 Port1 中断事件号为 31）。

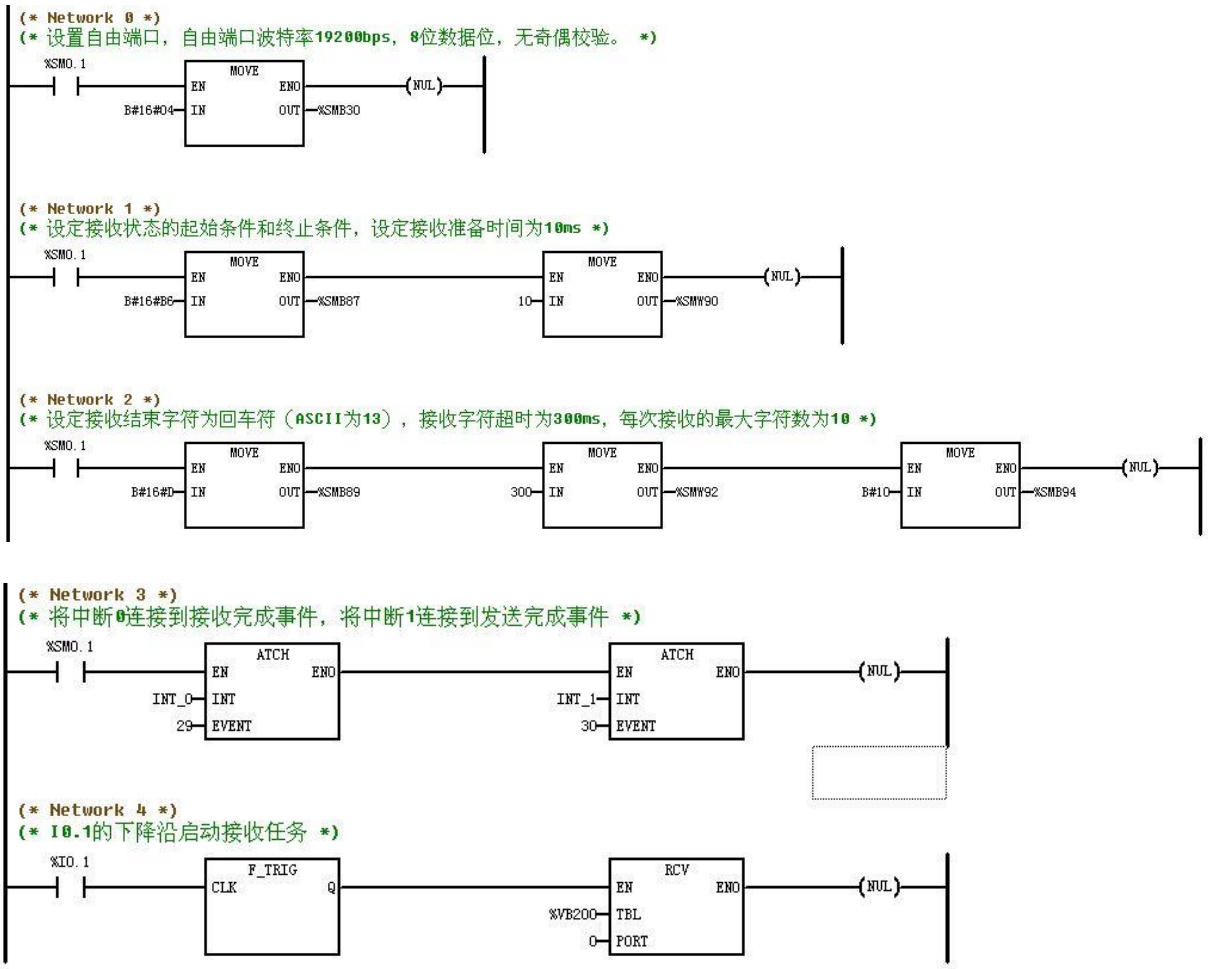
➢ 指令使用举例

下面将举例说明自由通讯的使用。

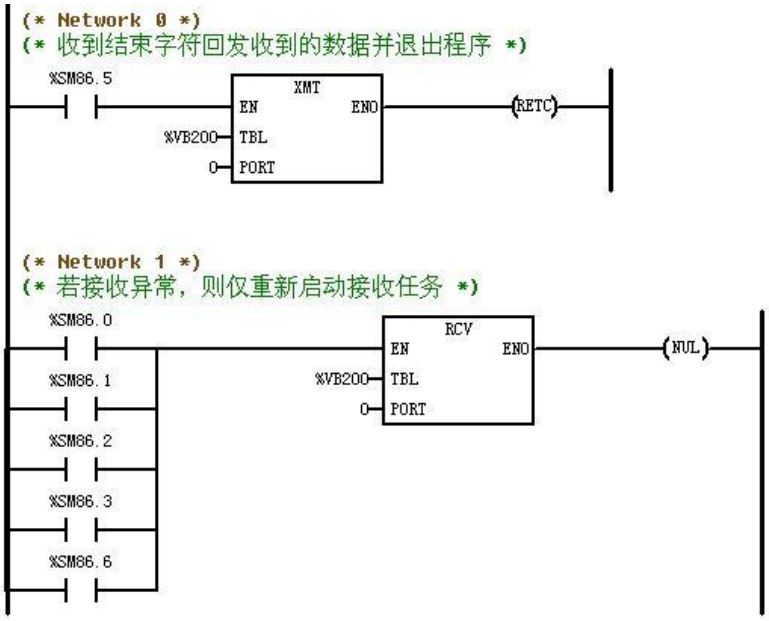
在示例中，CPU 将接收一串数据，以回车符作为接收结束字符。若接收正常完成，则把接收到的数据又发送回去并再次启动接收，若是异常退出接收状态（比如通讯错误、接收超时等），则忽略接收到的数据并再次启动接收。

LD

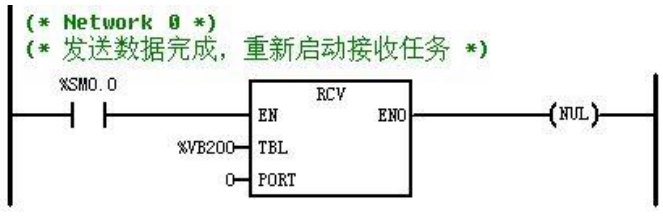
MAIN：



INT_0 接收完成中断服务程序：



INT_1 发送完成中断服务程序：



IL

MAIN：

```
(* Network 0 *)
(*设置自由端口，自由端口波特率 19200bps，8 位数据位，无奇偶校验。*)
LD          %SM0.1
MOVE        B#16#04, %SMB30
(* Network 1 *)
(*设定接收状态的起始条件和终止条件，设定接收准备时间为 10ms*)
LD          %SM0.1
MOVE        B#16#B6, %SMB87
MOVE        10, %SMW90
(* Network 2 *)
(*设定接收结束字符为回车符（ASCII 为 13），接收字符超时为 300ms，每次接收的最大字符数为 10*)
LD          %SM0.1
MOVE        B#16#D, %SMB89
MOVE        300, %SMW92
MOVE        B#10, %SMB94
(* Network 3 *)
(*将中断 0 连接到接收完成事件，将中断 1 连接到发送完成事件*)
LD          %SM0.1
ATCH        INT_0, 29
ATCH        INT_1, 30
(* Network 4 *)
(*I0.1 的下降沿启动接收任务*)
LD          %I0.1
F_TRIG
RCV         %VB200, 0
```

INT_0 接收完成中断服务程序：

```
(* Network 0 *)
(*收到结束字符回发收到的数据并退出程序*)
LD          %SM86.5
XMT         %VB200, 0
RETC
(* Network 1 *)
(*若接收异常，则仅重新启动接收任务*)
LD          %SM86.0
OR          %SM86.1
OR          %SM86.2
OR          %SM86.3
OR          %SM86.6
RCV         %VB200, 0
```

INT_1 发送完成中断服务程序：

```
(* Network 0 *)
(*发送数据完成，重新启动接收任务*)
LD          %SM0.0
RCV         %VB200, 0
```

5.12.2 Modbus RTU 主站指令

Modbus RTU 协议是目前使用最广泛的串行通讯协议之一。因此，为了方便用户的应用，EC100/EC200 提供 Modbus RTU 主站指令，用户直接调用这些指令就可以实现 Modbus RTU 主站的功能。目前，Modbus RTU 主站仅用于 Port1。

下面简要描述了用户进行 Modbus 主站编程的总体步骤：

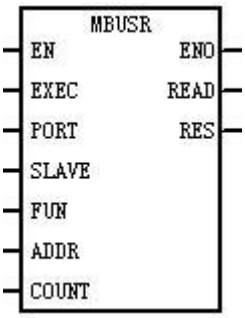
1) 在**【硬件配置】**设置所用通讯口的串行通讯参数（包括站号、波特率、奇偶校验等），并选中**【作为 Modbus 主站】**项。具体请参阅 3.2.3 配置模块参数 中的描述。

2) 在程序中直接调用 MBUSR、MBUSW 指令进行编程。

关于 Modbus RTU 协议的详细描述，请参阅附录 A 中 2、Modbus RTU 报文基本格式 。

5.12.2.1 MBUSR（Modbus 主站读）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	MBUSR		
IL	MBUSR	MBUSR *EXEC, PORT, SLAVE, FUN, ADDR, COUNT, READ, RES*	U

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BYTE	I、Q、V、M、L、SM、RS、SR
PORT	输入	INT	常量（1）
SLAVE	输入	BYTE	I、Q、M、V、L、SM、常量
FUN	输入	INT	常量（MODBUS 功能码）
ADDR	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
COUNT	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
READ	输出	BOOL、WORD、INT	Q、M、V、L、SM、AQ
RES	输出	BYTE	Q、M、V、L、SM

EC100/EC200 作为 Modbus RTU 主站时，MBUSR 指令用于读取从站内的数据。该指令适用的功能码有 1（读 DO）、2（读 DI）、3（读 AO）和 4（读 AI）。

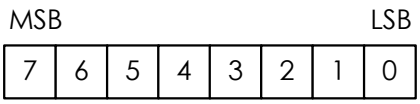
参数 *PORT* 定义了所用的通讯口，范围是 0 或 1。*SLAVE* 定义了目标从站的站号，允许的站号范围是 1~31。*FUN* 定义了功能码。*ADDR* 定义了要读取的寄存器的起始地址。*COUNT* 定义了读取的寄存器个数，允许的最大值是 32。

EXEC 输入端的上升沿跳变用于启动通讯。MBUSR 指令执行时，若检测到 *EXEC* 的上升沿跳变，MBUSR 就会进行一次通讯：按照用户输入的站号、功能码等参数来组织报文并完成 CRC 校验，然后将报文发送出去并等待从站的回应；当接收到从站返回的报文后，就对其进行 CRC 校验、地址校验和功能码校验，若校验后证明报文正确，那么所需的数据就会被写入数据缓冲区中，否则接收到的报文会被丢弃。

参数 *READ* 定义了数据缓冲区的起始地址，读取的数据（个数为 *COUNT*）就存放在该区域内。

READ 必须与功能码匹配。若功能码是 1、2，则输入 BOOL 型的地址变量；若功能码是 3、4，则输入 INT 或者 WORD 型的地址变量。

参数 *RES* 用于存放当前的状态信息和最近一次通讯的故障信息，它的组成如下图：



第7位 --- 当前的通讯状态。通讯时该位被置为0，否则被置为1。
第6位 --- 保留。
第5位 --- 非法的从站站号。
第4位 --- 数据个数错误。
第3位 --- 非法的寄存器地址。
第2位 --- 通讯口正被其它通讯任务占用。
第1位 --- 通讯超时：在用户指定的超时时间内仍没有收到从站的正确回应。
第0位 --- 接收数据帧错误，包括CRC校验错、功能码错等。

- **LD**

  如果 *EN* 为 1，则该指令被执行，否则不执行。

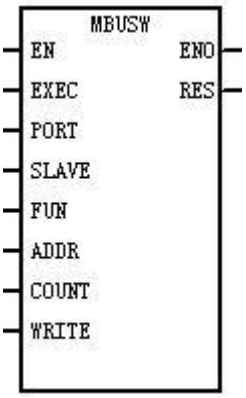
- **IL**

  如果 CR 值为 1，则该指令被执行，否则不执行。

  该指令的执行不影响 CR 值。

5.12.2.2 MBUSW（Modbus 主站写）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	MBUSW		
IL	MBUSW	MBUSW *EXEC, PORT, SLAVE, FUN, ADDR, COUNT, READ, RES*	U

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BYTE	I、Q、V、M、L、SM、RS、SR
PORT	输入	INT	常量（1）
SLAVE	输入	BYTE	I、Q、M、V、L、SM、常量
FUN	输入	INT	常量（MODBUS 功能码）
ADDR	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
COUNT	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
WRITE	输入	BOOL、WORD、INT	I、Q、RS、SR、V、M、L、SM、T、C、AI、AQ
RES	输出	BYTE	Q、M、V、L、SM

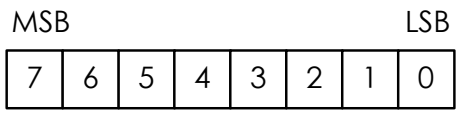
EC100/EC200 作为 Modbus RTU 主站时，MBUSW 指令用于将数据写入从站内。该指令适用的功能码有 5（写一个 DO）、6（写一个 AO）、15（写多个 DO）和 16（写多个 AO）。

参数 *PORT* 定义了所用的通讯口。*SLAVE* 定义了目标从站的站号，允许的站号范围是 1~31。*FUN* 定义了功能码。*ADDR* 定义了要写入的寄存器的起始地址。*COUNT* 定义了写寄存器的个数，允许的最大值是 32。

参数 *WRITE* 定义了数据缓冲区的起始地址，要写入从站的数据就存放在该区域内。*WRITE* 必须与功能码匹配。若功能码是 5、15，则输入 BOOL 型的地址变量；若功能码是 6、16，则输入 INT 或者 WORD 型的地址变量。

EXEC 输入端的上升沿跳变用于启动通讯。MBUSW 指令执行时，若检测到 *EXEC* 的上升沿跳变，MBUSW 就会进行一次通讯：按照用户输入的目标站号、功能码、目标寄存器、数量、写入的数据等参数来组织报文并完成 CRC 校验，然后将报文发送出去并等待从站的回应；当接收到从站返回的报文后，就对其进行 CRC 校验、地址校验和功能码校验，判读从站是否正确执行了刚才的写命令。

参数 *RES* 用于存放当前的状态信息和最近一次通讯的故障信息，它的组成如下图：



第7位 --- 当前的通讯状态。通讯时该位被置为0，否则被置为1。
第6位 --- 保留。
第5位 --- 非法的从站站号。
第4位 --- 数据个数错误。
第3位 --- 非法的寄存器地址。
第2位 --- 通讯口正被其它通讯任务占用。
第1位 --- 通讯超时：在用户指定的超时时间内仍没有收到从站的正确回应。
第0位 --- 接收数据帧错误，包括CRC校验错、功能码错等。

- **LD**

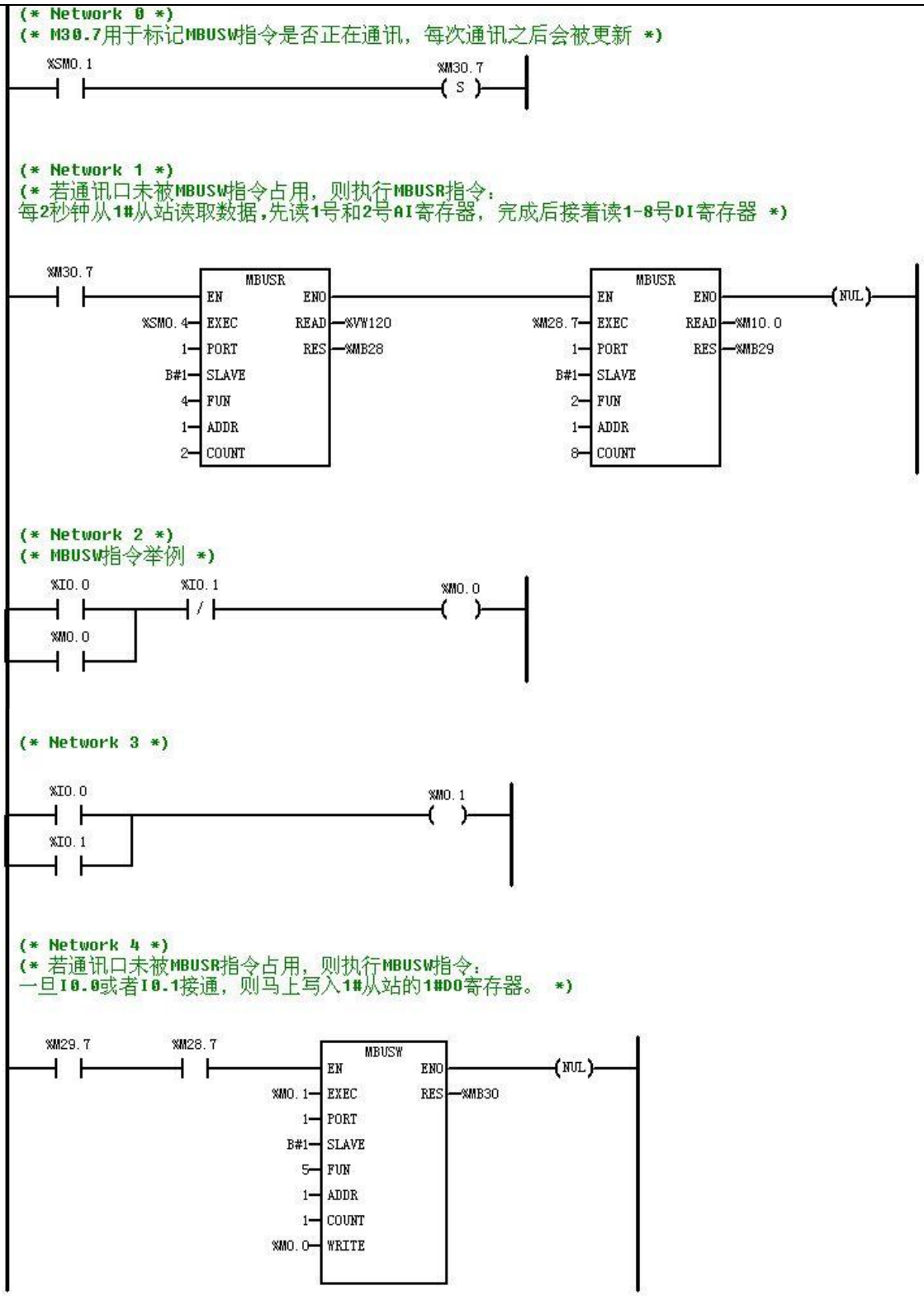
如果 *EN* 为 1，则该指令被执行，否则不执行。

- **IL**

  如果 CR 值为 1，则该指令被执行，否则不执行。该指令的执行不影响 CR 值。

5.12.2.3 MBUSR、MBUSW 使用举例

LD



续

IL	(* Network 0 *) (* M30.7 用于标记 MBUSW 指令是否正在通讯，每次通讯之后会被更新。*) LD %SM0.1 S %M30.7 (* Network 1 *) (* 若通讯口未被 MBUSW 指令占用，则执行 MBUSR 指令：*) (* 每 2 秒钟从 1#从站读取数据：*) (* 先读 1 号和 2 号 AI 寄存器，完成后接着读 1-8 号 DI 寄存器。*) LD %M30.7 MBUSR %SM0.4, 1, B#1, 4, 1, 2, %VW120, %MB28 MBUSR %M28.7, 1, B#1, 2, 1, 8, %M10.0, %MB29 (* Network 2 *) (*MBUSW 指令举例*) LD %I0.0 OR %M0.0 ANDN %I0.1 ST %M0.0 (* Network 3 *) LD %I0.0 OR %I0.1 ST %M0.1 (* Network 4 *) (* 若通讯口未被 MBUSR 指令占用，则执行 MBUSW 指令：*) (* 一旦 I0.0 或者 I0.1 接通，则马上写入 1#从站的 1#DO 寄存器。*) LD %M29.7 AND %M28.7 MBUSW %M0.1, 1, B#1, 5, 1, 1, %M0.0, %MB30

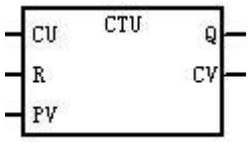
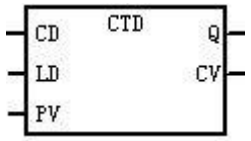
5.13 计数器

计数器是 IEC 61131-3 标准中定义的功能块之一，共有 CTU、CTD 和 CTUD 三种。

关于功能块及其实例的使用请参阅 2.6.5 关于功能块以及功能块实例 中的描述。注意，计数器实例不允许重复使用，在用户工程中不允许将同一个计数器实例分配给多个计数器指令。

5.13.1 CTU（增计数器）、CTD（减计数器）

- 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	CTU	Cx  	
	CTD	Cx  	
IL	CTU	CTU *Cx, R, PV*	P
	CTD	CTD *Cx, LD, PV*	

参数	输入/输出	数据类型	允许使用的内存区
Cx	-	计数器实例（*x* 表示编号）	C
CU	输入	BOOL	能量流
R	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
CD	输入	BOOL	能量流
LD	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
PV	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
Q	输出	BOOL	能量流
CV	输出	INT	Q、M、V、L、SM、AQ

- **LD**

CTU 指令对 *CU* 输入端的上升沿进行递增计数（每次加 1）并把 *Cx* 的计数值赋给 *CV*。*Cx* 将一直计数直至达到并保持在最大值 32767。当 *CV* 大于等于预设值 *PV* 时，*Q* 及 *Cx* 的状态值均被置为 1，否则均被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，*CV* 及其计数值全被清零。

CTD 指令对 *CD* 输入端的上升沿进行递减计数（每次减 1）并把 *Cx* 的计数值赋给 *CV*。当 *CV* 等于 0 时，*Cx* 停止计数，*Q* 及 *Cx* 的状态值均被置为 1，否则均被置为 0。若 *LD* 输入为 1，则 *Cx* 被复位，将 *PV* 重新赋给 *Cx* 的计数值及 *CV*。

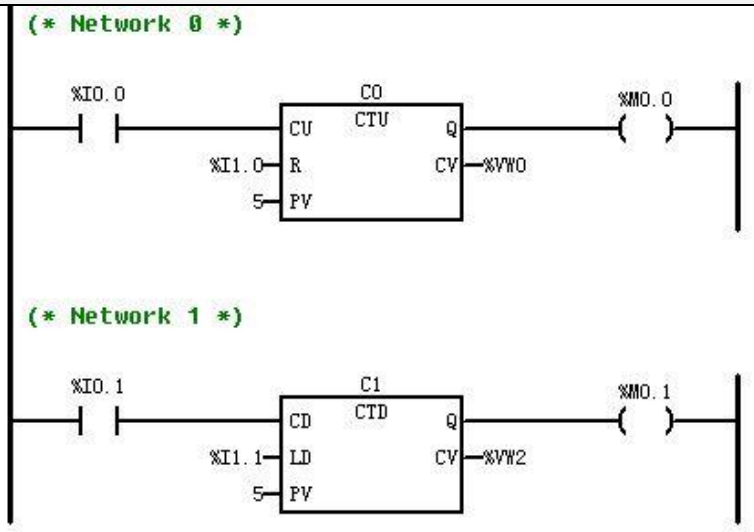
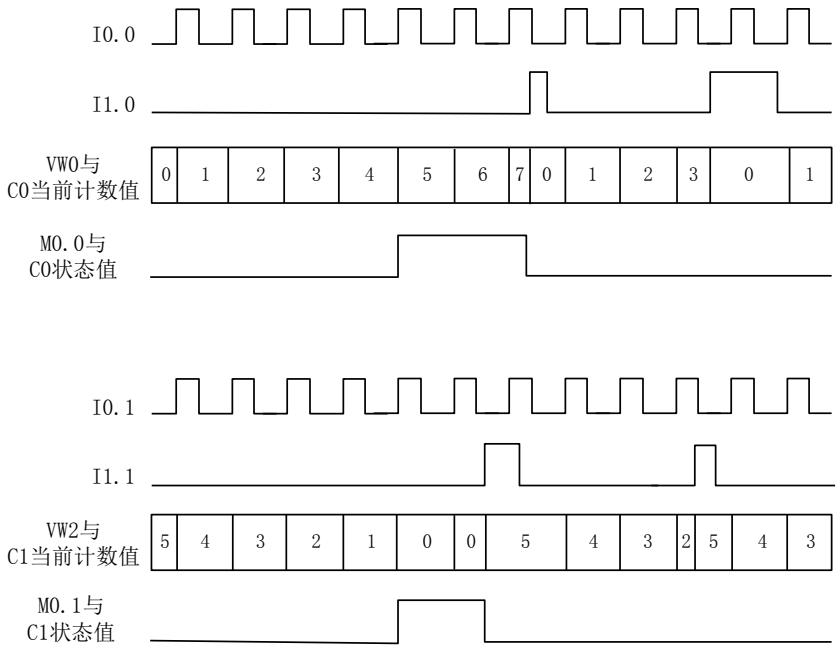
- **IL**

CTU 指令对 CR 值的上升沿进行递增计数（每次加 1）。*Cx* 将一直计数直至达到并保持在最大值 32767。当计数值大于等于预设值 *PV* 时，*Cx* 的状态值被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值被清零。

CTD 指令对 CR 值的上升沿进行递减计数（每次减 1）。当计数值等于 0 时，*Cx* 停止计数，其状态值被置为 1，否则被置为 0。若 *LD* 输入为 1，则 *Cx* 被复位，将 *PV* 重新赋给 *Cx* 的计数值。

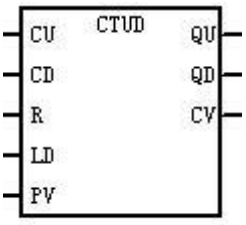
每次扫描 CTU、CTD 指令后，CR 值就被更新为 *Cx* 的状态值。

指令使用实例：

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* Network 0 *)
LD %I0.0
CTU C0, %I1.0, 5
MOVE C0, %VW0
ST %M0.0
(* Network 1 *)
LD %I0.1
CTD C1, %I1.1, 5
MOVE C1, %VW2
ST %M0.1</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

5.13.2 CTUD（增/减计数器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	CTUD	Cx  	
IL	CTUD	CTUD *Cx, CD, R, LD, PV, QD*	P

参数	输入/输出	数据类型	允许使用的内存区
Cx	-	计数器实例（*x* 表示编号）	C
CU	输入	BOOL	能量流
CD	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
R	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
LD	输入	BOOL	I、Q、M、V、L、SM、T、C、RS、SR
PV	输入	INT	I、Q、M、V、L、SM、AI、AQ、常量
QU	输出	BOOL	能量流
QD	输出	BOOL	Q、M、V、L、SM
CV	输出	INT	Q、M、V、L、SM、AQ

- **LD**

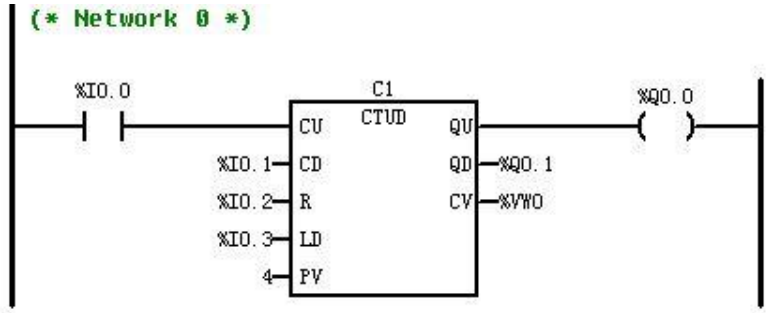
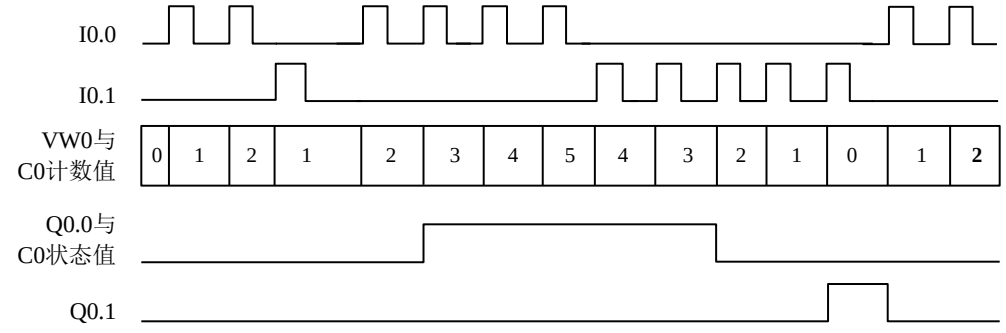
CTUD 指令对 *CU* 输入端的上升沿进行递增计数（每次加 1）并对 *CD* 输入端的上升沿进行递减计数（每次减 1），*Cx* 的计数值被赋给 *CV*。当 *CV* 大于等于预设值 *PV* 时，*QU* 及 *Cx* 的状态值均被置为 1，否则均被置为 0。当 *CV* 等于 0 时，*QD* 被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值及 *CV* 均被清零。若 *LD* 输入为 1，则将 *PV* 重新赋值给 *Cx* 的计数值及 *CV*。当 *R* 及 *LD* 同时输入为 1 时，*R* 优先。

- **IL**

CTUD 指令对 CR 值的上升沿进行递增计数（每次加 1）并对 *CD* 输入端的上升沿进行递减计数（每次减 1）。当计数值大于等于预设值 *PV* 时，*Cx* 的状态值被置为 1，否则被置为 0。当计数值等于 0 时，*QD* 被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值被清零。若 *LD* 输入为 1，则将 *PV* 重新赋值给 *Cx* 的计数值。当 *R* 及 *LD* 同时输入为 1 时，*R* 优先。

每次扫描 CTUD 指令后，CR 值就被更新为 *Cx* 的状态值。

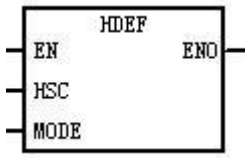
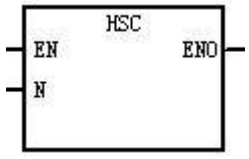
指令使用实例：

LD	IL
	(* Network 0 *) LD %I0.0 CTUD C1, %I0.1, %I0.2, %I0.3, 4, %Q0.1 MOVE C1, %VW0 ST %Q0.0
时序图	
	

5.13.3 HDEF（高速计数器定义）、HSC（高速计数器）

高速计数器的运行不受 CPU 扫描周期的影响，因此可以用于对高速的脉冲输入进行计数。

- 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	HDEF		
	HSC		
IL	HDEF	HDEF *HSC, MODE*	U
	HSC	HSC *N*	

参数	输入/输出	数据类型	描　述
HSC	输入	INT	常量（高速计数器编号）
MODE	输入	INT	常量（高速计数器的工作模式）
N	输入	INT	常量（高速计数器编号）

HDEF 指令的作用是：为参数 *HSC* 指定的高速计数器定义一种工作模式 *MODE*。

HSC 指令的作用是：依据 HDEF 指令指定的工作模式以及 SM 区中相应控制寄存器的值来配置高速计数器 *N* 的特性，然后启动高速计数器 *N*。

- **LD**

  若 *EN* 值为 1，则执行 HDEF、HSC 指令，否则不执行。

- **IL**

  若 CR 值为 1，则执行 HDEF、HSC 指令，否则不执行。HDEF、HSC 的执行不影响 CR 值。

5.13.3.1 EC100/EC200 中的高速计数器

	EC102、EC104、EC106、EC202、EC204、EC206
高速计数器	4 个，（HSC0 至 HSC3）
单相	4 个，最高计数频率 200KHz
双相	2 个，最高计数频率 50KHz

高速计数器具有各种各样的工作模式，在使用高速计数器之前，必须用 HDEF 指令为其指定一种工作模式。其中，HSC0、HSC2 有 8 种工作模式，HSC1、HSC3 有 1 种工作模式。所有的高速计数器在相同的工作模式下均具有相同的功能。

5.13.3.2 高速计数器工作模式和输入信号

高速计数器的输入信号包括如下几种：时钟（即输入脉冲）、方向和复位信号。

若复位信号为 0，则允许高速计数器计数。若复位信号为 1，则当前计数值被清零。对于外部方向控制的单相高速计数器，若方向信号为 1，则增计数，否则减计数。

在不同的工作模式下，所需要的输入信号也有所不同。下面各表详细描述了各个高速计数器所支持的工作模式及其输入信号的分配。

HSC0				
模式	**描述**	**I0.0**	**I0.1**	**I0.4**
0	带内部方向控制的单相增/减计数器 方向控制位：SM37.3	时钟		
1		时钟		复位
2	带外部方向控制的单相增/减计数器	时钟	方向	
3		时钟	方向	复位
4	带增/减计数时钟输入的双相计数器	时钟（增）	时钟（减）	
5		时钟（增）	时钟（减）	复位
6	A/B 相正交计数器	时钟 A 相	时钟 B 相	
7		时钟 A 相	时钟 B 相	复位

HSC1		
模式	**描述**	**I0.1**
0	带内部方向控制的单相增/减计数器 方向控制位：SM47.3	时钟

HSC2				
模式	**描述**	**I0.2**	**I0.3**	**I0.5**
0	带内部方向控制的单相增/减计数器 方向控制位：SM57.3	时钟		
1		时钟		复位
2	带外部方向控制的单相增/减计数器	时钟	方向	
3		时钟	方向	复位
4	带增/减计数时钟输入的双相计数器	时钟（增）	时钟（减）	
5		时钟（增）	时钟（减）	复位
6	A/B 相正交计数器	时钟 A 相	时钟 B 相	
7		时钟 A 相	时钟 B 相	复位

HSC3		
模式	**描述**	**I0.3**
0	带内部方向控制的单相增/减计数器 方向控制位：SM137.3	时钟

5.13.3.3 高速计数器的时序图

为了让用户更好地理解高速计数器，如下各图举例详细描述了高速计数器的工作时序。

图5-2适用于所有具有复位输入信号的工作模式。图5-3至图5-7描述了高速计数器各个工作模式的时序图。

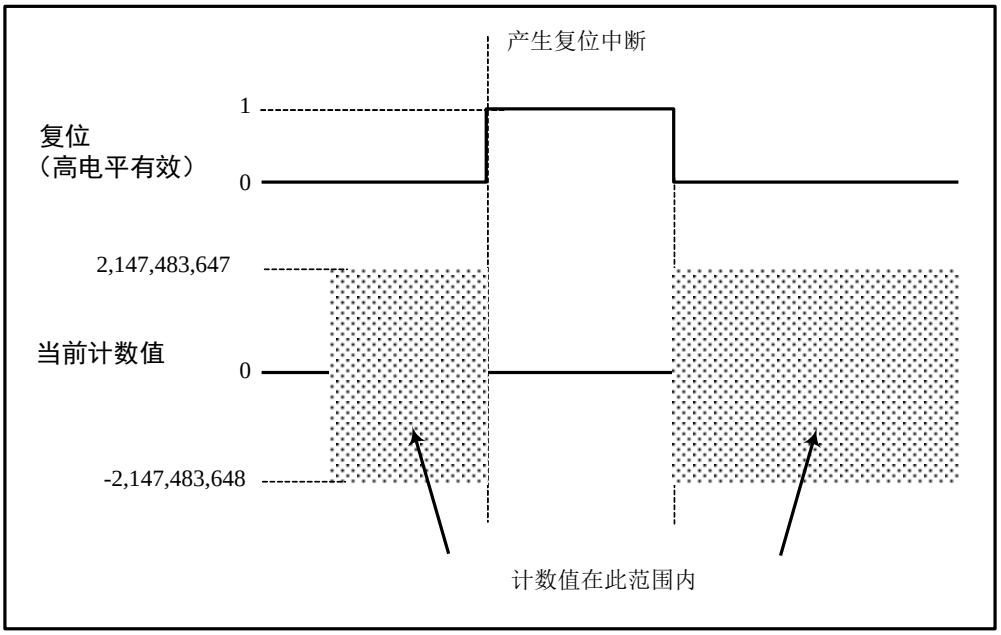


图 5-2　有复位信号的高速计数器时序图

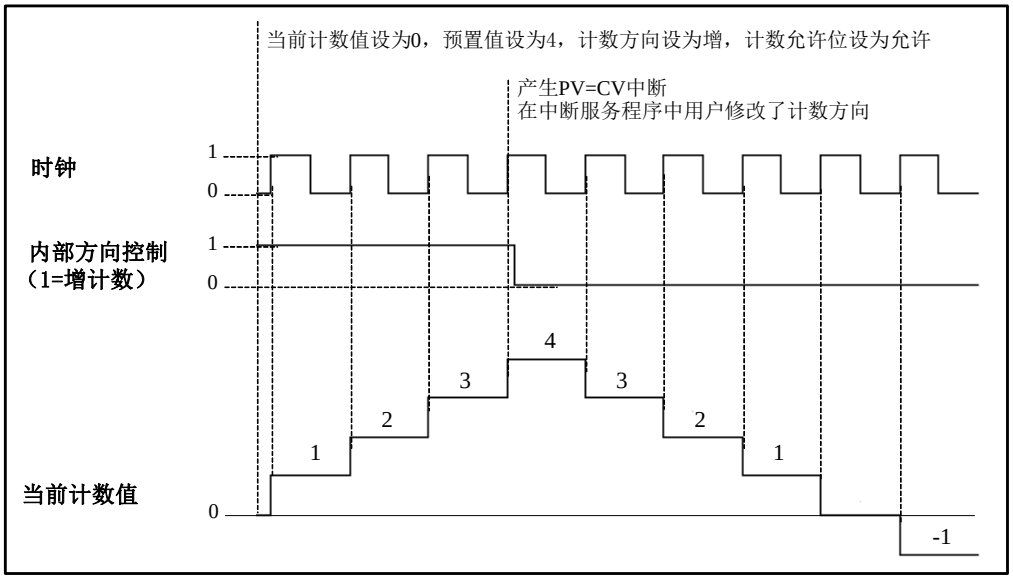


图 5-3　模式 0、1 的时序图

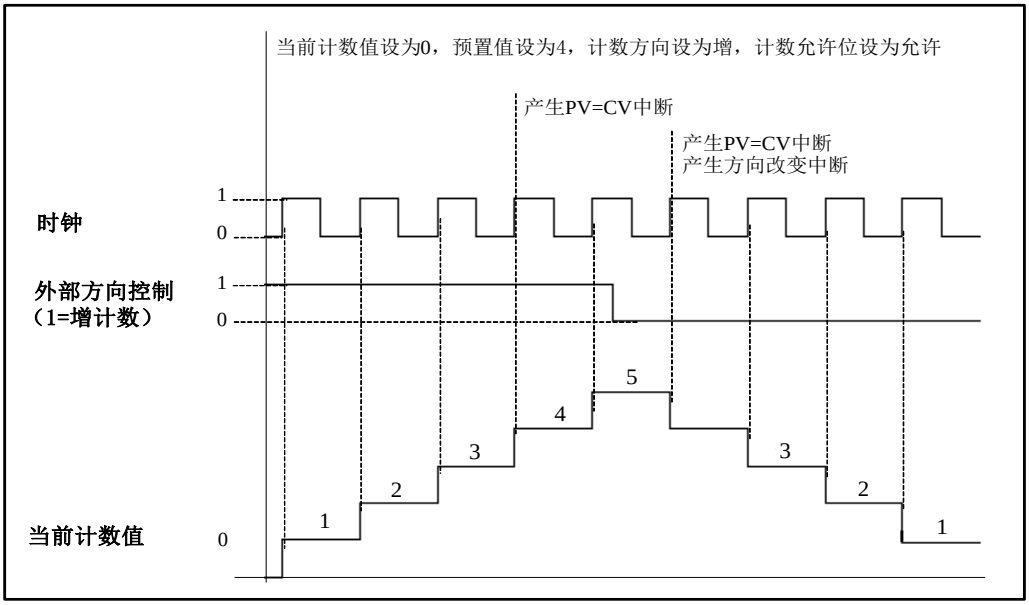


图 5-4　模式 2、3 的时序图

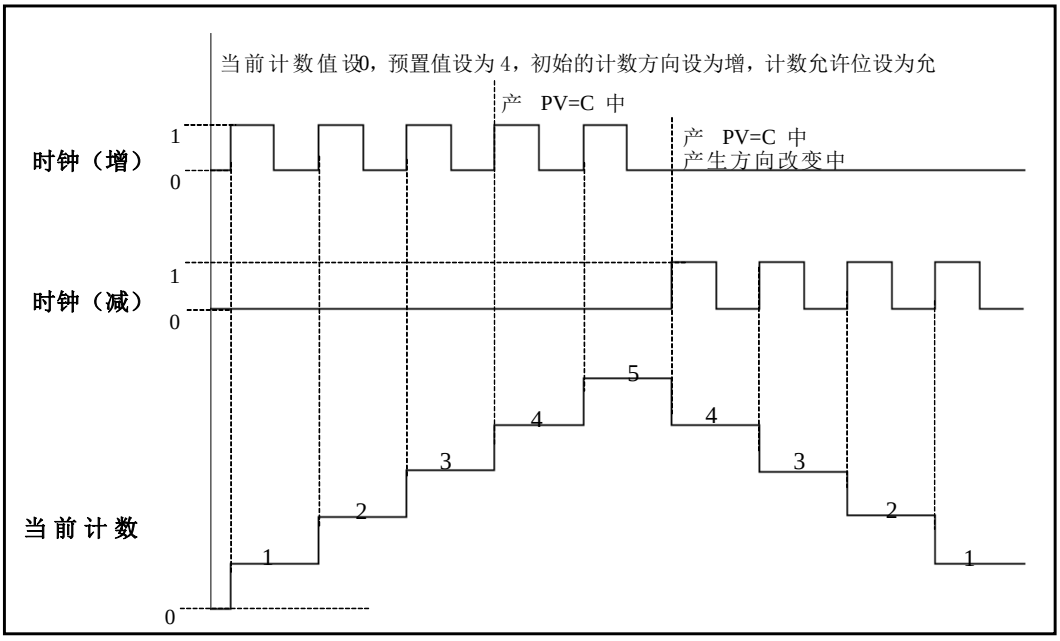


图 5-5　模式 4、5 的时序图

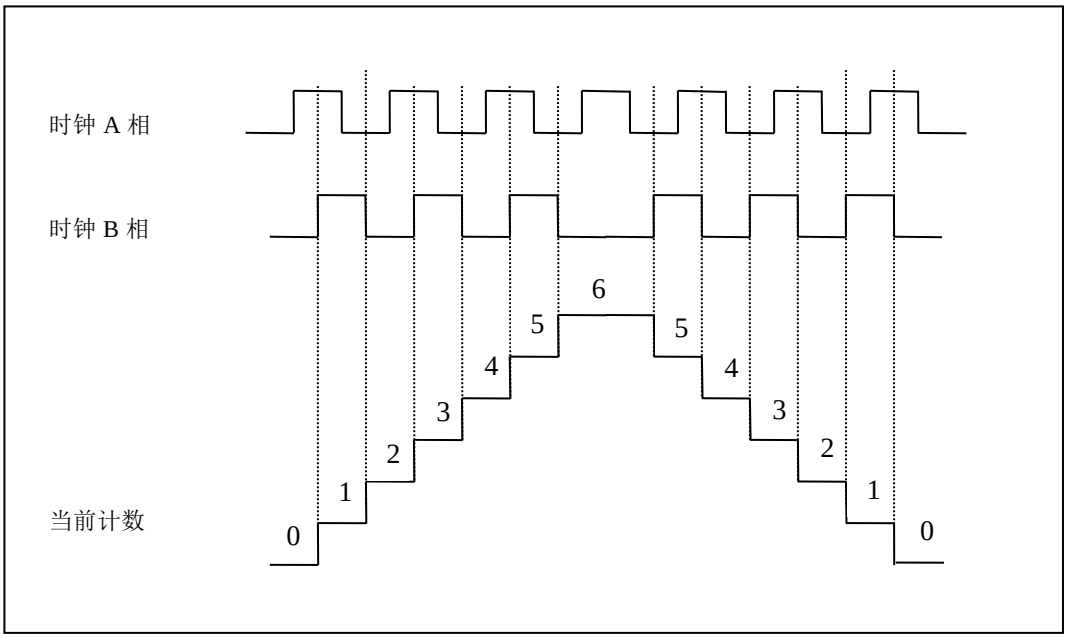


图 5-6　模式 6、7 的时序图（AB 相 2x）

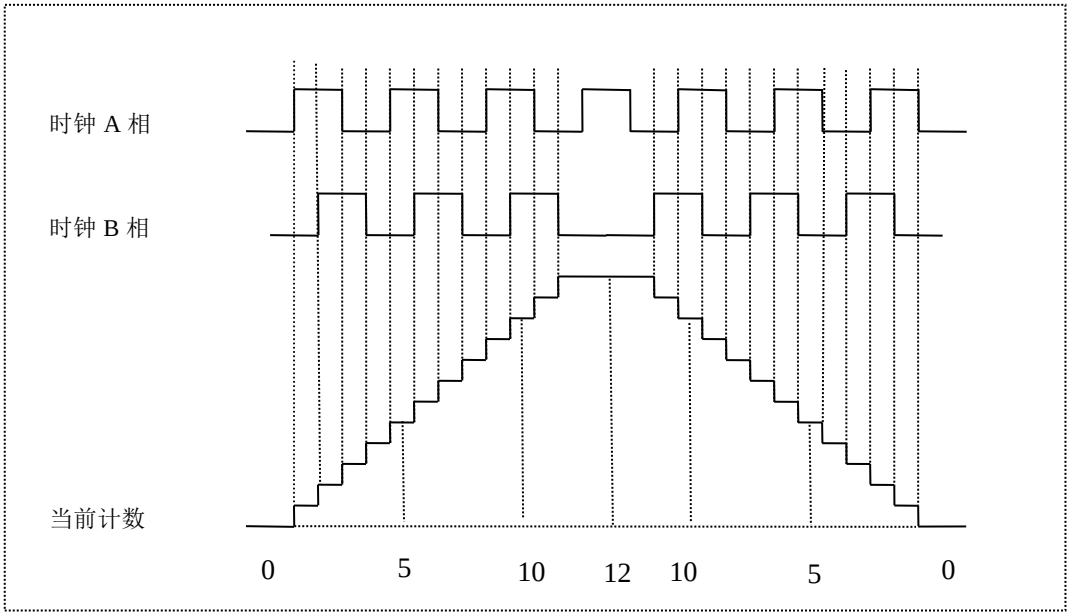


图 5-7　模式 6、7 的时序图（AB 相 4x）

5.13.3.4　控制寄存器和状态寄存器

- **控制寄存器**

在 SM 区中为每个高速计数器均提供了如下控制寄存器用于存放其配置数据：一个控制字节（8 位）、当前值（32 位有符号双整数）和预置值（32 位有符号双整数）各一个。当前值指定了计数的起始值，若将当前值写入了高速计数器，那么高速计数器就会立即从这个数值开始计数。

下表详细描述了这些控制寄存器。

表 5-3　高速计数器的控制寄存器

HSC0	HSC1	HSC2	HSC3	描 述
SM37.0	---	SM57.0	---	复位信号的有效电平： 0=高电平；1=低电平
SM37.1	SM47.1	SM57.1	SM137.1	保留
SM37.2	---	SM57.2	---	正交计数器速率： 0=2x 速率；1=4x 速率
SM37.3	SM47.3	SM57.3	SM137.3	计数方向： 0=减计数；1=增计数。
SM37.4	SM47.4	SM57.4	SM137.4	是否向 HSC 中写入计数方向： 0=否；1=是。
SM37.5	SM47.5	SM57.5	SM137.5	是否向 HSC 中写入新预置值： 0=否；1=是。
SM37.6	SM47.6	SM57.6	SM137.6	是否向 HSC 中写入新当前值： 0=否；1=是。
SM37.7	SM47.7	SM57.7	SM137.7	是否允许该高速计数器： 0=禁止；1=允许。
HSC0	**HSC1**	**HSC2**	**HSC3**	**描 述**
SMD38	SMD48	SMD58	SMD138	当前值
SMD42	SMD52	SMD62	SMD142	预置值

需要注意的是，控制字节中并非所有的控制位都适用于所有的工作模式。比如，“计数方向”和“是否向 HSC 中写入计数方向”这两个控制位就只用于模式 0、1（带内部方向控制的单相增/减计数器），若高速计数器所用的工作模式是采用外部的方向控制信号，那么这两个控制位就会被忽略。

控制字节、当前值和预置值的缺省值均为 0。用户修改高速计数器特性的方法是：首先设置好相应的控制寄存器，然后再执行 HSC 指令。

- **状态寄存器**

每个高速计数器都在 SM 区中占有一个状态字节，这个字节中的某些位指明了高速计数器当前的状态信息。下表列出了各个高速计数器的状态字节。

表 5-4　高速计数器的状态字节

HSC0	HSC1	HSC2	HSC3	Description
SM36.0	SM46.0	SM56.0	SM136.0	保留
SM36.1	SM46.1	SM56.1	SM136.1	保留
SM36.2	SM46.2	SM56.2	SM136.2	保留
SM36.3	SM46.3	SM56.3	SM136.3	保留
SM36.4	SM46.4	SM56.4	SM136.4	保留

SM36.5	SM46.5	SM56.5	SM136.5	当前计数方向（只适用于单相计数）： 0=减；1=增。
SM36.6	SM46.6	SM56.6	SM136.6	当前计数值是否等于预置值： 0=否；1=是。
SM36.7	SM46.7	SM56.7	SM136.7	当前计数值是否大于预置值： 0=否；1=是。

➢ **读取高速计数器的当前计数值**

高速计数器的计数值是 32 位的双整数，并且是只读的。

EC100/EC200 在内存区域中提供了高速计数器区（HC 区）用于存放各高速计数器的计数值。这个区域的寻址方式是内存区域类型（HC）和高速计数器的编号。比如，HC0 表示 HSC0 的当前计数值。请参阅 4.3.2.1 直接地址表示格式 了解更多信息。下图描述了这种寻址方式。

图 5-8　读取高速计数器的计数值示例

5.13.3.5　高速计数器中断

各高速计数器均支持中断：当计数值等于预置值时就会产生“PV=CV”中断；若是使用外部复位信号的高速计数器模式，那么在复位时就会产生“外部复位”中断；若是使用外部方向控制信号的计数器模式，那么在计数方向改变时就会产生“方向改变”中断。

请参阅 5.10 中断指令 了解更多信息。

5.13.3.6　使用高速计数器

用户可以按照如下步骤来编程使用一个高速计数器：

- 配置该高速计数器的控制字节，并指定当前值（也就是计数的起始值）和预置值；
- （可选）使用 ATCH 指令为高速计数器中断连接相应的中断服务程序；
- 使用 HDEF 指令来定义一个高速计数器及其工作模式；

  注意：在 CPU 进入 RUN 模式后，只允许为每个高速计数器执行一次 HDEF 指令。
- 使用 HSC 指令来配置并启动高速计数器。

下面将以 HSC0 为例来详细说明如何编程使用高速计数器。建议用户在工程中尽量编写单独

的初始化子程序，其中包含有 HDEF 指令和其它的初始化指令，这样可以使整个用户工程具有良好的结构。另外，每次调用 HSC 指令都会对高速计数进行初始化重新计数操作。

➢ **使用高速计数器**

下面将以模式 1 为例进行描述，但是描述的步骤在原理上同样适用于其它工作模式，用户根据实际需要稍作调整即可。

1) 根据期望的操作来设置控制字节 SMB37。

例如：SMB37 = B#16#FC 表明了：

- 允许HSC0计数；
- 允许向HSC0中写入新的当前值和预置值
- 设置计数方向为增计数，允许向HSC0中写入计数方向；
- 设置复位信号为高电平有效。

2) 将期望的当前值（32 位双整数，也就是计数的起始值）赋给 SMD38。

3) 将期望的预置值（32 位双整数）赋给 SMD42。

4) （可选）使用ATCH指令为“PV = CV”中断事件（事件号18）连接一个中断服务程序以实现对该中断事件的快速响应。

5) （可选）使用ATCH指令为“方向改变”中断事件（事件号17）连接一个中断服务程序以实现对该中断事件的快速响应。

6) （可选）使用 ATCH 令为“外部复位”中断事件（事件号 16）连接一个中断服务程序以实现对该中断事件的快速响应。

7) 执行 HDEF 指令，在 *HSC* 端输入为 0，*MODE* 端输入为 1（代表采用模式 1）；

8) 执行 HSC 指令来配置并启动 HSC0。

➢ **改变计数方向（模式 0、1）**

下面描述了如何来改变 HSC0 的计数方向（模式 0、1）。

1) 根据期望的操作来设置控制字节 SMB37。

例如，SMB37 = B#16#90 表明了：

- 允许计数；
- 向 HSC0 中写入新的计数方向；
- 设置计数方向为减计数。

2) 执行 HSC 指令来配置 HSC0。

➢ **改变当前值（适用于所有模式）**

下面描述了如何改变 HSC0 的当前值（也就是计数的起始值）。

(1) 根据期望的操作来设置控制字节 SMB37。

SMB37 = B#16#C0 表明了：

- 允许计数；
- 允许向 HSC2 中写入新的当前值。

(2) 将期望的当前值（32 位双整数）赋给 SMD38。

(3) 执行 HSC 指令来配置 HSC0。

➢ **改变预置值（适用于所有模式）**

下面描述了如何改变 HSC0 的预置值。

1) 根据期望的操作来设置控制字节 SMB37。

SMB37= B#16#A0 表明了：

- 允许计数；
- 允许更新 HSC2 的预置值。

2) 将期望的预置值（32 位双整数）赋给 SMD42。

3) 执行 HSC 指令来配置 HSC0。

➢ **禁止高速计数器（适用于所有模式）**

下面描述了如何禁止 HSC0。

1) 根据期望的操作来设置控制字节 SMB37。

SMB37= B#16#00 表明了：禁止该高速计数器。

2) 执行 HSC 指令以使 CPU 禁止 HSC0。

5.13.3.7 高速计数器示例

MAIN：

```
(* Network 0 *)
(* 调用高速计数初始化子程序 *)
%SM0.1 ──┤ ├──┤ SBR_0 ├──(NUL)
                EN  ENO
```

SBR_0：

```
(* Network 0 *)
(* 允许计数，允许更新当前值、预设值，初始计数方向为增，复位信号为高电平有效，
当前值为0，预设值为3000 *)
%SM0.0 ──┤ ├──┤ MOVE ├──┤ MOVE ├──┤ MOVE ├──(NUL)
              EN  ENO      EN  ENO      EN  ENO
B#16#F8 ──IN  OUT──%SMB37  DI#0 ──IN  OUT──%SMD38  DI#3000──IN  OUT──%SMD42

(* Network 1 *)
(* 连接中断 *)
%SM0.0 ──┤ ├──┤ ATCH ├──(NUL)
              EN  ENO
INT_0 ──INT
10 ──EVENT

(* Network 2 *)
(* HSC0的工作模式为1，并启动HSC *)
%SM0.0 ──┤ ├──┤ HDEF ├──┤ HSC ├──(NUL)
              EN  ENO      EN  ENO
0 ──HSC                0 ──N
1 ──MODE
```

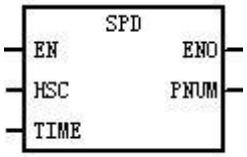
INT_0：

```
(* Network 0 *)
(* CV=PV中断程序：Q0.0取反 *)
%Q0.0 ──┤ ├──(/)──%Q0.0
```

```
MAIN：
(* Network 0 *)
(*调用高速计数初始化子程序*)
LD          %SM0.1
CAL         SBR_0
SBR_0：
(* Network 0 *)
(*允许计数，允许更新当前值、预设值，初始计数方向为增，复位信号为高电平有效，  当前值为 0，预设值为 3000*)
LD          %SM0.0
MOVE        B#16#FC, %SMB37
MOVE        DI#0, %SMD38
MOVE        DI#3000, %SMD42
(* Network 1 *)
(*连接中断*)
LD          %SM0.0
ATCH        INT_0, 18
(* Network 2 *)
(*HSC0 的工作模式为 1，并启动 HSC*)
LD          %SM0.0
HDEF        0, 1
HSC         0
INT_0：
(* Network 0 *)
(*允许写入当前值，当前值设为 0*)
LD          %SM0.0
MOVE        B#16#C0, %SMB37
MOVE        DI#0, %SMD38
(* Network 0 *)
(*CV=PV 中断程序：Q0.0 取反*)
LD          %Q0.0
STN         %Q0.0
```

5.13.4 SPD（脉冲密度）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	SPD		
IL	SPD	SPD *HSC, TIME, PNUM*	U

参数	输入/输出	数据类型	允许的内存区
HSC	输入	INT	常量（高速计数器编号）
TIME	输入	WORD	I、Q、M、V、L、SM、常量
PNUM	输出	DINT	Q、M、V、L、SM

SPD 指令用于计算高速计数器 *HSC* 在时间 *TIME*（单位：ms）内输入的脉冲个数，并将结果写入 *PNUM* 中。

- **LD**

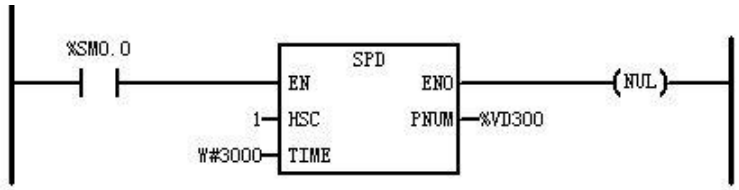
  如果 *EN* 为 1，则该指令被执行。

- **IL**

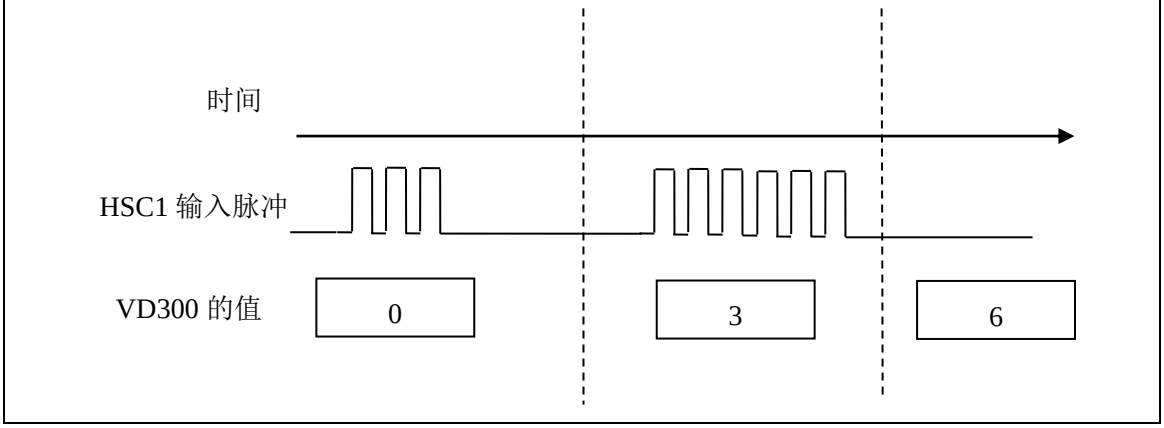
  如果 CR 值为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	（*HSC1 的脉冲计数值进行计算，每到 3000ms 就将这个时间内计算的脉冲个数输出至 VD0*） LD %SM0.0 SPD 1, W#3000, %VD300

举例结果如下：

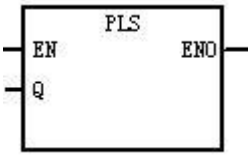


5.13.5 PLS（高速脉冲输出，仅适用于 EC200）

EC200 提供了两路高速脉冲输出（EC100 不提供该功能），输出通道分别使用使用 Q0.0 和 Q0.1。在这里，高速脉冲输出指的是 PTO 或者 PWM 输出。

- PTO：Pulse Train Output，脉冲串输出。
- PWM：Pulse-Width Modulation，脉宽调制。

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	PLS		
IL	PLS	PLS *Q*	U

参数	输入/输出	数据类型	允许的内存区
Q	输入	INT	常量（0 或 1）

PLS 指令的作用是：依据 SM 区中相应控制寄存器的值来配置高速脉冲输出的特性，然后启动高速脉冲输出。输出通道由参数 *Q* 指定，0 表示使用 Q0.0 输出，1 表示使用 Q0.1 输出。

- **LD**

  若 *EN* 值为 1，则 PLS 指令被执行。

- **IL**

  若 CR 值为 1，则 PLS 指令被执行。该指令的执行不影响 CR 值。

5.13.5.1 EC200 的高速脉冲输出功能

EC200 支持两路高速脉冲输出，相应地就提供了两个 PTO/PWM 脉冲发生器用于产生 PTO/PWM 输出，输出频率最高可达 200kHz。其中，一个脉冲发生器分配在 Q0.0，称为 PWM0 或者 PTO0；另外一个分配在 Q0.1，称为 PWM1 或者 PTO1。

PTO/PWM 发生器和输出映像寄存器共同使用内存地址 Q0.0 和 Q0.1。如果 Q0.0 或者 Q0.1 设定为 PTO 或者 PWM 功能，PTO/PWM 发生器将控制输出通道，并禁止通用功能的输出。当不使用 PTO/PWM 功能时，Q0.0、Q0.1 由输出映像寄存器控制。

在 SM 区中为每个 PTO/PWM 发生器提供了如下寄存器：一个控制字节（8 位），周期值和脉宽值各一个（均为 16 位无符号整数），一个脉冲计数值（32 位无符号双整数）。用户在使用高速输出功能时，需要先设置好这些寄存器，然后再执行 PLS 指令就可以实现所期望的 PTO/PWM 操作。例如，将控制字节的允许位（SM67.7 或 SM77.7）设置为 0，并执行 PLS 指令，就可以禁止相应的 PTO/PWM 输出。所有控制字节、周期、脉冲数的缺省值都是 0。

注意： 若Q0.0、Q0.1是继电器类型的，则避免使用高速脉冲输出功能！

- **PWM**

PWM 功能提供占空比可调的连续脉冲输出。用户可以控制输出的周期和脉宽。

周期和脉宽的单位可以选择微秒（μs）或毫秒（ms），相应地，周期的范围分别为 5~65535μs 或 2~65535 ms，脉宽的范围分别为 0~65535μs 或 0~65535 ms。当脉宽大于等于周期时，占空比自动地被设为 100%，输出一直接通。当脉宽为 0 时，占空比为 0%，输出断开。

用户可以采用如下方法来改变 PWM 波形的特性：

- **同步更新：**是指在不改变时间基准前提下进行的更新。所谓的时间基准是指周期的单位选择微秒还是毫秒。利用同步更新，波形特性的变化发生在周期边沿，因此能够提供平滑的特性变化。PWM 的同步操作既可以改变脉宽值也可以改变周期值，当指令触发后会在当前脉冲输出完毕后输出改变的脉冲，使用同步更新需要对控制寄存器（SM67 或 SM77）设置相应的控制位来更新周期值或更新脉宽值。
- **异步更新：**是指在改变时间基准前提下进行的更新。异步更新会造成 PWM 被瞬间禁止或者造成 PWM 波形的不同步，从而可能引起被控设备的振动。异步更新需要用户设置脉冲的周期值、脉冲宽度以及时间基准。

- **PTO**

PTO 功能能够产生指定脉冲个数的脉冲串方波（50%占空比）。用户可以控制输出方波的周期和输出脉冲的个数。脉冲周期的单位是微秒（μs ）或者毫秒（ms），相应地，周期的范围是 5~65535μs 或者 2~65535 ms。脉冲个数的范围是：1～4,294,967,295。如果指定脉冲数为 0，则 EC200 不发脉冲。

PTO 功能提供了单段操作和多段操作两种模式。

- **单段操作**

在单段操作模式下，每次执行 PLS 指令后仅会进行一次脉冲串输出。

单段操作模式允许 2 个脉冲串排队：在一个脉冲串尚未输出完时，允许再次执行 PLS 指令来配置并启动另一个脉冲串的输出，第二个脉冲串需要设置脉冲周期、脉冲个数以及时基（注意：如果改变时间基准可能会因为周期变化大而引起突然抖动现象，因此不建议改变第二个脉冲串的时基）。第二个脉冲串在队列中会一直保持到第一个脉冲串输出完成，一旦第一个脉冲串输出完成，第二个脉冲串就会立即输出。

如果已有排队的第二个脉冲串，则状态寄存器中的 PTO 单段排队标志位（SM66.6/SM76.6）被置位，PTO 单段脉冲输出将不再存储脉冲串数据。当排队的脉冲串已经输出时，PTO 排队标志位将被清 0。

- **多段操作**

在多段操作模式下，CPU 自动从 V 区的包络表中读出每个 PTO 段的设定值并依据设定值执行该段 PTO。

各段在包络表中的设置均占用 10 个字节，包括一个初始周期值（16 位无符号整数）、一个最终周期值（16 位无符号整数）、一个加/减速时间值（16 位无符号整数）和一个脉冲个数值（32 位无符号双整数）。多段输出可以存储 32 段脉冲输出的设置参数，其使用 PLS 指令来配置参数并启动多段输出。

*注意：*加/减速时间单位为 ms 级。例如，加/减速时间设置为 100，即加/减速时间为 100ms。

加/减速时间必须是初始周期和最终周期两者中最大值的 20 倍以上。即，$t \geq T_{MAX} \times 20$，t 单位为 ms。例如，当初始周期为 200us，最终周期为 50us，则加/减速时间 t=200us×20=4ms，因此，加/减速时间最少为 4ms。当初始周期为 100ms，最终周期为 20ms，则加/减速时间 t=100ms×20=2000ms，因此加/减速时间最少为 2000ms。

PTO 多段脉冲输出数据表的起始位置存储在 SMW168（对应 PTO0）、SMW178（对应 PTO1）中，时基通过 SM67.3（对应 PTO0）、SM77.3（对应 PTO1）设置，可以选择微秒或毫秒。数据表中的所有周期值必须使用同一个时基，并且在多段执行时不能改变。

多段操作的数据表设置格式如下表所示。

表 5-5　PTO 多段操作的数据表设置格式

字节偏移量（1）	长 度	段 数	描 述
0	8 位		段数（1 到 32）
1	16 位	第 1 段	初始周期（2 到 65535 时基）
3	16 位		最终周期（2 到 65535 时基）
5	16 位		加/减速时间（1 到 65535，单位 ms）
7	32 位		脉冲个数（1 到 4,294,967,295）
11	16 位	第 2 段	初始周期（2 到 65535 时基）
13	16 位		最终周期（2 到 65535 时基）
15	16 位		加/减速时间（1 到 65535，单位 ms）
17	32 位		脉冲数（1 到 4,294,967,295）
…		…	…

（1）所有偏移量均是相对于多段输出起始位置的偏移字节数。

注意：　多段输出的起始位置必须为 V 区中的奇数地址，如 VB3001。

PTO 多段操作功能非常有用，尤其对于步进电机的操作。例如，用户可以生成一个多段输出数据表，然后启动 PTO 多段操作，按照数据表的数值输出脉冲让步进电机进行多段的加速、减速、恒速运行。其 PTO 多段输出的示例如下图所示。

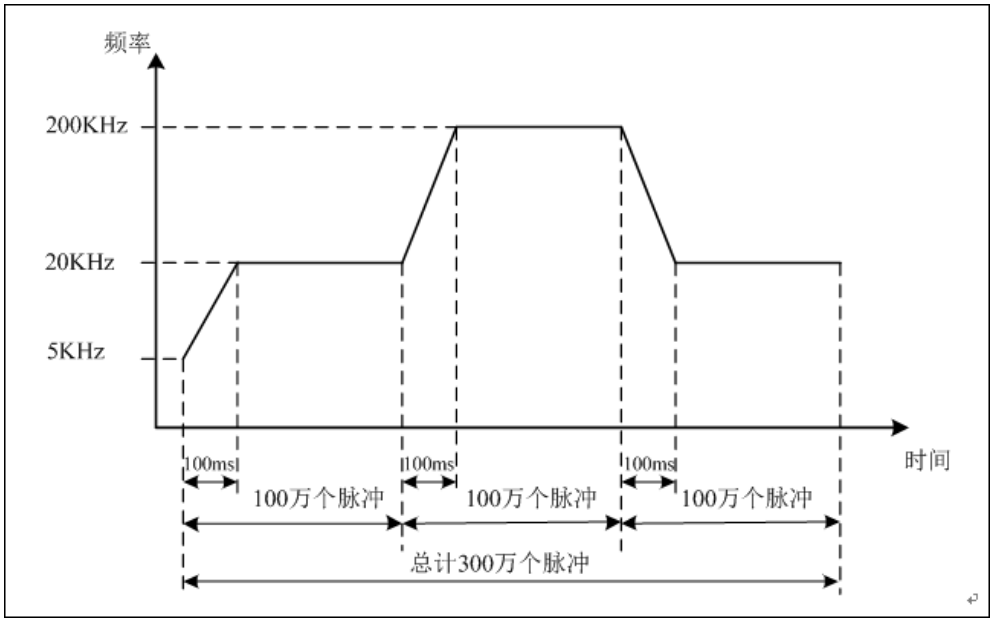


图 5-9 PTO 多段输出示例

本例中 PTO 多段输出共有 3 段，第一段的初始频率为 5KHz，最终频率为 20KHz，加/减速时间为 100ms，输出脉冲个数为 1000000；第二段的初始频率为 20KHz，最终频率为 200KHz，加/减速时间为 100ms，输出脉冲个数为 1000000；第三段的初始频率为 200KHz，最终频率为 20KHz，加/减速时间为 100ms，输出脉冲个数为 10000000。假定多段脉冲数据的存储起始位置从 VB61 开始，则多段脉冲数据存储格式如表 5-6 所示。

表 5-6　PTO 多段脉冲输出数据表

地址	值	描 述	
VB61	3	段数	
VW62	200	初始周期值(us)	第 1 段
VW64	50	最终周期值(us)	
VW66	100	加/减速时间(ms)	
VD68	1000000	脉冲个数	
VW72	50	初始周期值(us)	第 2 段
VW74	5	最终周期值(us)	
VW76	100	加/减速时间(ms)	
VD78	1000000	脉冲个数	
VW82	5	初始周期值(us)	第 3 段
VW84	50	最终周期值(us)	
VW86	100	加/减速时间(ms)	
VD88	1000000	脉冲个数	

5.13.5.2 PTO/PWM 寄存器

在 SM 区中为每个 PTO/PWM 发生器均提供了一些控制寄存器用于存放其配置数据。

下表详细描述了这些控制寄存器。

表 5-7 PTO/PWM 发生器的控制寄存器

Q0.0	Q0.1	描 述
SM67.0	SM77.0	PWM 是否更新周期值：0=否；1=是
SM67.1	SM77.1	PWM 是否更新脉宽值：0=否；1=是
SM67.2	SM77.2	保留
SM67.3	SM77.3	PTO/PWM 时基：0=1μs；1=1ms
SM67.4	SM77.4	PWM 更新方法：0=异步更新；1=同步更新
SM67.5	SM77.5	PTO 操作方式：0=单段操作；1=多段操作
SM67.6	SM77.6	功能选择：0= PTO；1=PWM
SM67.7	SM77.7	PTO/PWM 允许或禁止此功能：0=禁止；1= 允许
Q0.0	**Q0.1**	**描 述**
SMW68	SMW78	PTO/PWM 周期值，范围 2~65535
SMW70	SMW80	PWM 脉宽值，范围 0~65535
SMD72	SMD82	PTO 脉冲个数，范围 1~4,294,967,295
SMB166	SMB176	正在进行的 PTO 多段操作段数
SMW168	SMW178	多段输出数据表的起始位置（用相对于 VB0 的字节偏移来表示），仅用于 PTO 多段操作。

所有控制字节、周期、脉冲数的缺省值都是 0。用户修改 PTO/PWM 波形的特性的方法是：首先设置相应的控制寄存器，如果是 PTO 多段操作，数据表也得先设置好，然后再执行 PLS 指令。

在 SM 区中也为每个 PTO/PWM 发生器均提供了一个状态字节，用户可以通过访问状态字节来了解 PTO/PWM 发生器的当前状态信息。下表描述了 PTO/PWM 发生器的状态字节。

表 5-8 PTO/PWM 发生器的状态字节

Q0.0	Q0.1	描 述
SM66.4	SM76.4	PTO 多段输出是否由于加减速时间设置过短而终止：0=否；1=是
SM66.5	SM76.5	PTO 多段输出是否由于存储起始位置不为奇数地址而终止：0=否；1=是
SM66.6	SM76.6	PTO 单段排队输出是否已满：0=否；1=是
SM66.7	SM76.7	PTO 是否空闲：0＝忙；1＝空闲

PTO空闲位（SM66.7或者SM76.7）指明了PTO输出是否已经结束。

5.13.5.3 PTO 中断

各 PTO 均支持中断：当 PTO 输出完成时就会产生“PTO 完成”中断事件。其中，“PTO0 完成”中断的事件号是 28，“PTO1 完成”中断的事件号是 27。若采用的 PTO 多段操作模式，则在整个包络表完成后产生该中断事件。

请参阅 5.10 中断指令 了解更多信息。

5.13.5.4 使用 PTO

下面以 PTO0 为例来介绍如何编程使用 PTO 功能。总体上，使用 PTO 包括两个步骤：设置相关的控制寄存器，初始化 PTO；执行 PLS 指令。

建议用户在工程中尽量编写单独的初始化子程序，这样可以使整个用户工程具有良好的结构。另外，若有可能的话，尽量在主程序中以 SM0.1 为条件来调用这个初始化子程序，这样该子程序将只在 CPU 上电后的首次扫描中调用并执行一次，可以减少 CPU 的扫描时间。

➢ **执行PTO（单段操作）**

1) 根据期望的操作来设置控制字节SMB67。

   例如，SMB67 = B#16#80 表明了：

   - 允许 PTO/PWM 功能；
   - 选择使用 PTO 功能，单段操作；
   - 时基选择为 1μs；

2) 将期望的周期值赋给 SMW68。

3) 将期望的脉冲个数赋给 SMD72。

4) （可选）使用 *ATCH* 指令为“PTO0完成”中断事件（事件号28）连接一个中断服务程序以实现对该中断事件的快速响应。

5) 执行 PLS 指令来配置并启动 PTO0。

➢ **PTO排队（单段操作）**

按照前面“执行PTO”的步骤先启动第一个PTO，然后针对第二个PTO执行如下操作：

1) 根据期望的操作来设置控制字节SMB67。

   例如，SMB67 = B#16#80 表明了：

   - 允许 PTO/PWM 功能；
   - 选择使用 PTO 功能，单段操作；
   - 同步更新，时基选择为 1μs；

2) 将期望的周期值赋给 SMW68。

3) 将期望的脉冲个数赋给 SMD72。

4) 执行 PLS 指令来配置并启动 PTO0。正在执行的 PTO 输出完成后，具有新周期值和新脉冲个数的 PTO 就会立即接着启动。

➢ **执行PTO（多段操作）**

1) 根据期望的操作来设置控制字节SMB67。

例如，SMB67 = B#16#A0 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PTO 功能
- 选择多段操作；
- 时基选择为 1μs。

2) 将数据表的起始位置（奇数，表示数据表起始地址相对于VB0的字节偏移）赋给SMW168。
3) 设置数据表中的相关数值。
4) （可选）使用 *ATCH* 指令为“PTO0完成”中断事件（事件号28）连接一个中断服务程序以实现对该中断事件的快速响应。
5) 执行PLS 指令来配置并启动PTO0。

5.13.5.5 使用 PWM

下面以 PWM0 为例来介绍如何编程使用 PWM 功能。总体上，使用 PWM 包括两个步骤：设置相关的控制寄存器，初始化 PWM；执行 PLS 指令。

建议用户在工程中尽量编写单独的初始化子程序,这样可以使整个用户工程具有良好的结构。另外，若有可能的话，尽量在主程序中以 SM0.1 为条件来调用这个初始化子程序，这样该子程序将只在 CPU 上电后的首次扫描中调用并执行一次，可以减少 CPU 的扫描时间。

➢ **使用PWM**

1) 根据期望的操作来设置控制字节SMB67。

例如，SMB67 = B#16#D0 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PWM 功能；
- 选择使用同步更新方式；
- 时基选择为 1μs；

2) 将期望的周期值赋给 SMW68。
3) 将期望的脉宽值赋给 SMW70。
4) 执行 PLS 指令来配置并启动 PWM0。

➢ **改变周期**

下面描述了如何改变 PWM0 的周期。

1) 根据期望的操作来设置控制字节SMB67。

例如，SMB67 = B#16#D1 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PWM 功能；
- 选择使用同步更新方式；
- 时基选择为 1μs；
- 允许更新周期值。

2) 将期望的脉宽值赋给 SMW68。

3) 执行 PLS 指令来配置并启动 PWM0。

➢ **改变脉宽**

下面描述了如何改变 PWM0 的脉宽。

1) 根据期望的操作来设置控制字节SMB67。

例如，SMB67 = B#16#D2 表明了：

- 允许 PTO/PWM 功能；
- 选择使用 PWM 功能；
- 选择使用同步更新方式；
- 时基选择为 1μs；
- 允许更新脉宽值。

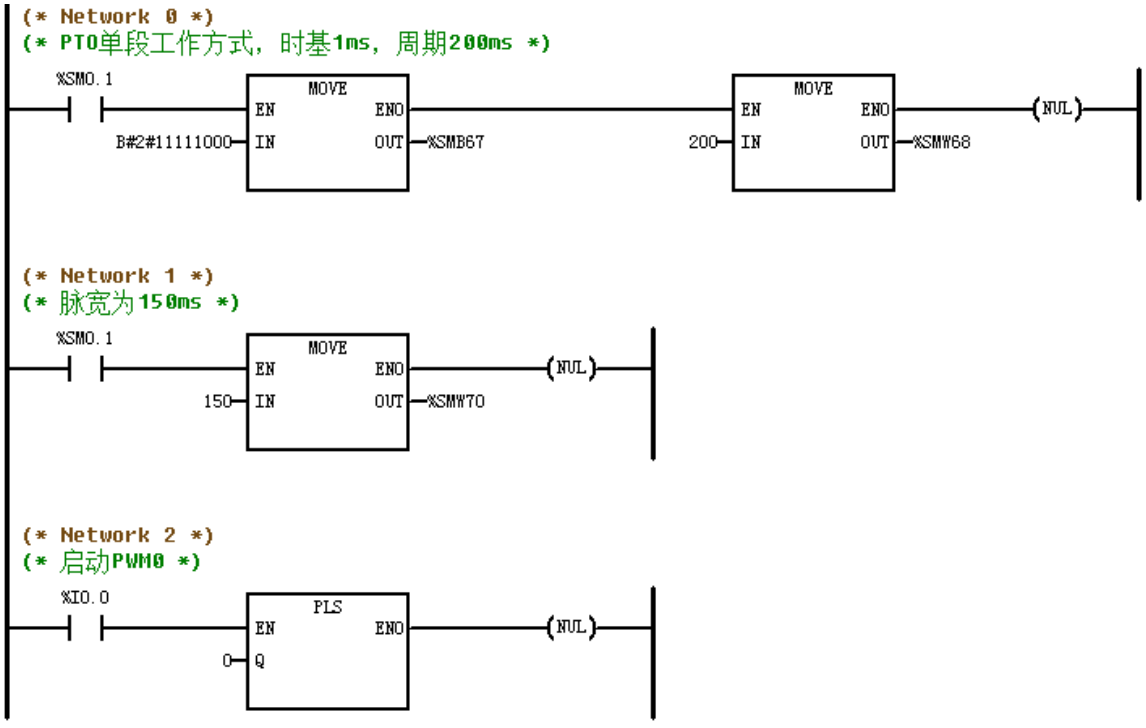
2) 将期望的脉宽值赋给 SMW70。

3) 执行 PLS 指令来配置并启动 PWM0。

5.13.5.6 示例

PWM 示例

LD

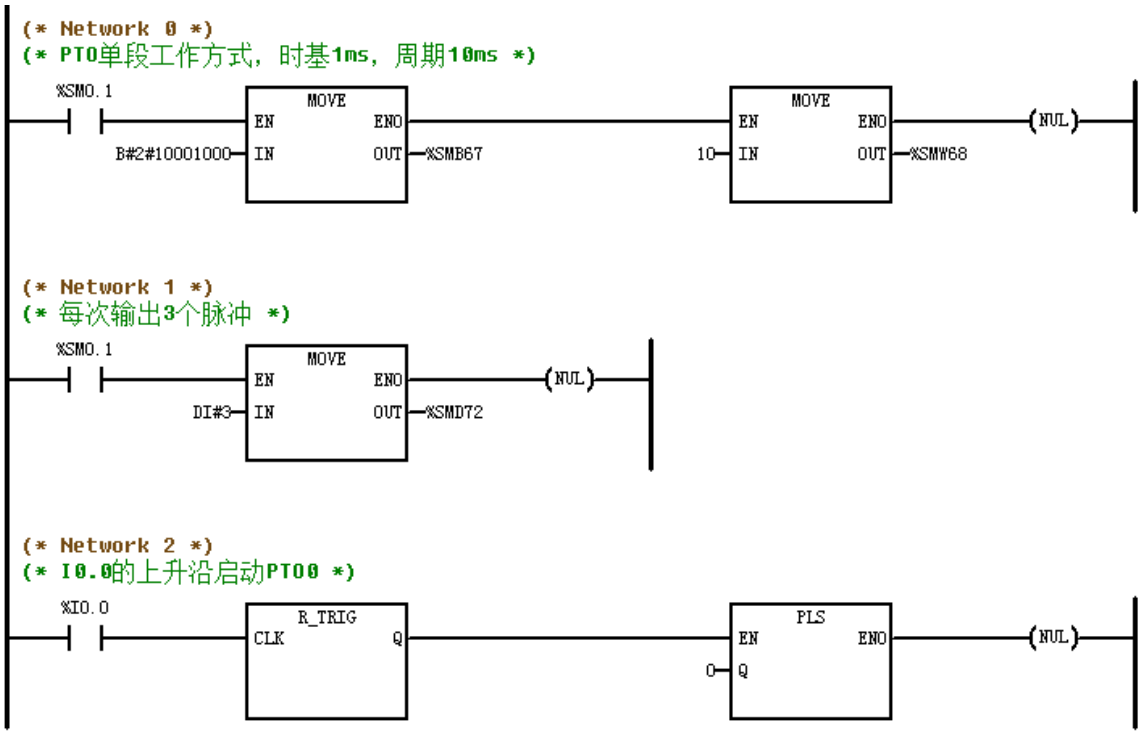


IL

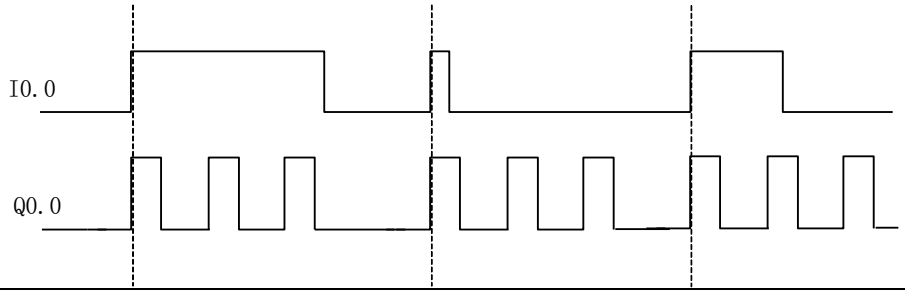
```
(* Network 0 *)
(*PWM 工作方式，时基为 1ms，脉冲周期为 200ms*)
LD          %SM0.1
MOVE        B#2#11111000, %SMB67
MOVE        200, %SMW68
(* Network 1 *)
(*脉宽为 150ms*)
LD          %SM0.1
MOVE        150, %SMW70
(* Network 2 *)
(*启动 PWM0*)
LD          %I0.0
PLS         0
```

PTO 单段示例

LD



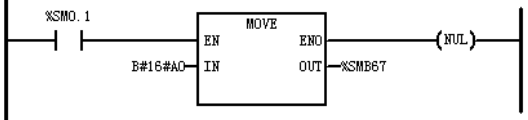
IL	(* Network 0 *)
	(*PTO 单段方式，时基为 1ms，脉冲周期为 10ms*)
	LD %SM0.1
	MOVE B#2#10001000, %SMB67
	MOVE 10, %SMW68
	(* Network 1 *)
	(*每次输出 3 个脉冲*)
	LD %SM0.1
	MOVE DI#3, %SMD72
	(* Network 2 *)
	(*I0.0 的上升沿启动 PTO*)
	LD %I0.0
	R_TRIG
	PLS 0

结果	

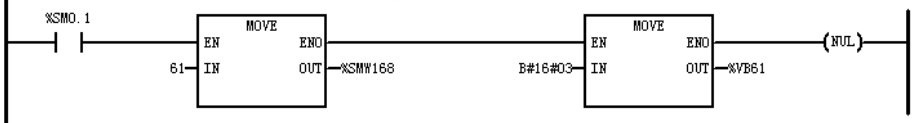
PTO 多段示例

LD

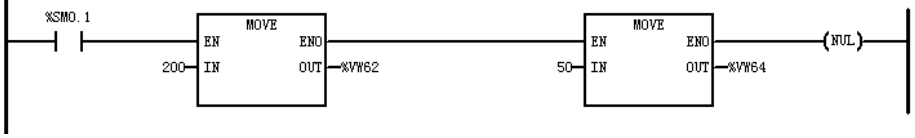
(* Network 0 *)
(* PTO多段模式，时基为1us *)



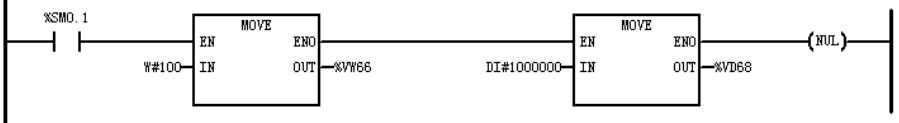
(* Network 1 *)
(* 包络表的起始地址为VB61，总段数为3段 *)



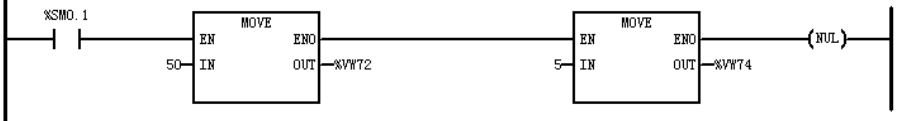
(* Network 2 *)
(* 第一段：初始周期200us，最终周期50us *)



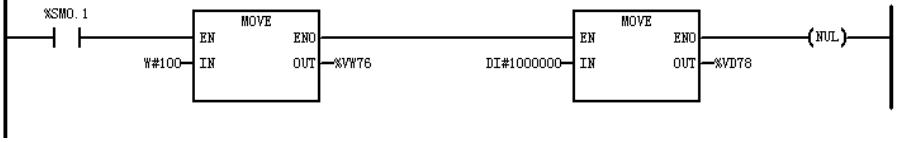
(* Network 3 *)
(* 第一段：加减速时间100ms，脉冲个数1000000 *)



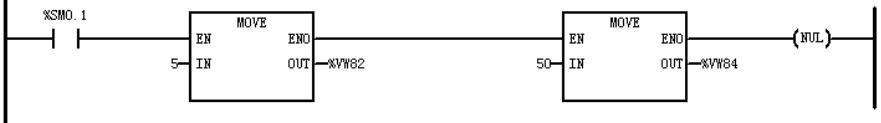
(* Network 4 *)
(* 第二段：初始周期50us，最终周期5us *)



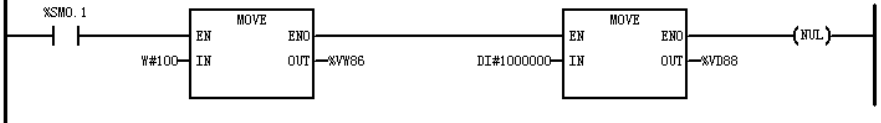
(* Network 5 *)
(* 第二段：加减速时间100ms，脉冲个数1000000 *)



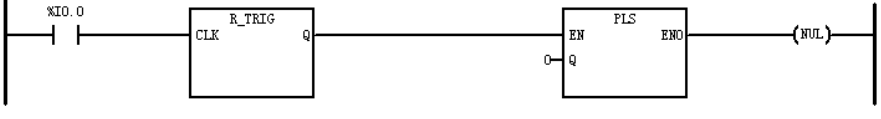
(* Network 6 *)
(* 第三段：初始周期5us，最终周期50us *)



(* Network 7 *)
(* 第三段：加减速时间100ms，脉冲个数1000000 *)



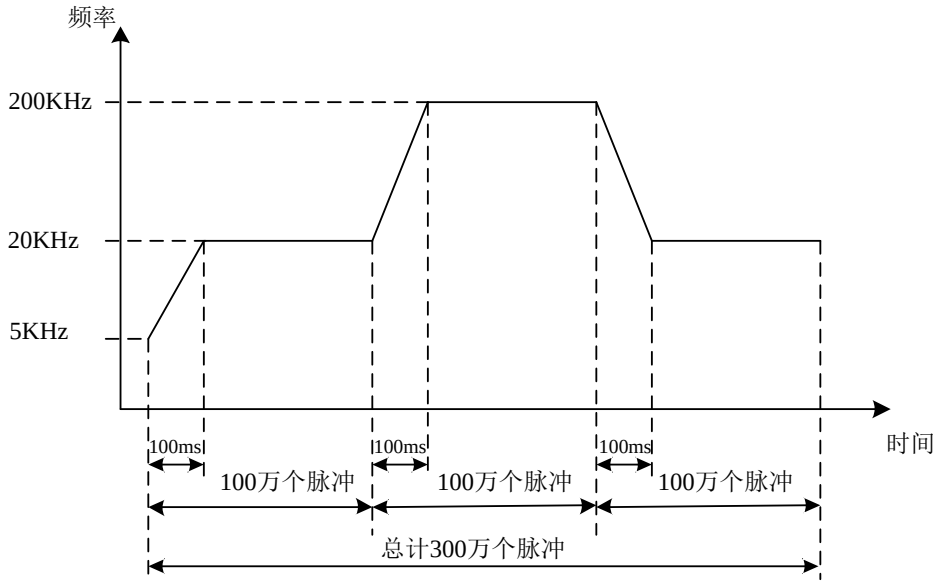
(* Network 8 *)
(* I1.0上升沿启动PTO0 *)



IL

```
(* Network 0 *)
(*PTO 多段模式，时基为 1us*)
LD        %SM0.1
MOVE      B#16#A0, %SMB67
(* Network 1 *)
(*包络表的起始地址为 VB61，总段数为 3 段*)
LD        %SM0.1
MOVE      61, %SMW168
MOVE      B#16#03, %VB61
(* Network 2 *)
(*第一段：初始周期 200us，最终周期 50us*)
LD        %SM0.1
MOVE      200, %VW62
MOVE      50, %VW64
(* Network 3 *)
(*第一段：加减速时间 100ms，脉冲个数 1000000*)
LD        %SM0.1
MOVE      W#100, %VW66
MOVE      DI#1000000, %VD68
(* Network 4 *)
(*第二段：初始周期 50us，最终周期 5us*)
LD        %SM0.1
MOVE      50, %VW72
MOVE      5, %VW74
(* Network 5 *)
(*第二段：加减速时间 100ms，脉冲个数 1000000*)
LD        %SM0.1
MOVE      W#100, %VW76
MOVE      DI#1000000, %VD78
(* Network 6 *)
(*第三段：初始周期 5us，最终周期 50us*)
LD        %SM0.1
MOVE      5, %VW82
MOVE      50, %VW84
(* Network 7 *)
(*第三段：加减速时间 100ms，脉冲个数 1000000*)
LD        %SM0.1
MOVE      W#100, %VW86
MOVE      DI#1000000, %VD88
(* Network 8 *)
(*I1.0 上升沿启动 PTO0*)
LD        %I0.0
R_TRIG
PLS       0
```

结果



5.14 定时器

定时器是 IEC 61131-3 标准中定义的功能块之一，共有 TON、TONR、TOF 和 TP 四种。

5.14.1 定时器的时基

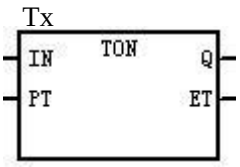
EC100/EC200 提供了三种时基的定时器。定时器号决定了其时基，如下表：

	EC102、EC104、EC106、EC202、EC204、EC206
时 基	T0 --- T3：1ms T4 --- T19：10ms T20 --- T255：100ms
最大定时时间	32767×时基

定时器的预设值和计时值均是其时基的倍数。比如，对于 10ms 时基的定时器，100 就代表 1000ms。

5.14.2 TON（接通延时定时器）

➢ 指令及其操作数说明

	名称	**指令格式**	**影响 CR 值**
LD	TON		
IL	TON	TON *Tx, PT*	P

参数	**输入/输出**	**数据类型**	**允许使用的内存区**
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

- **LD**

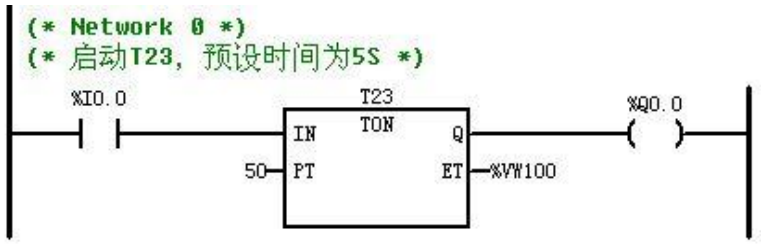
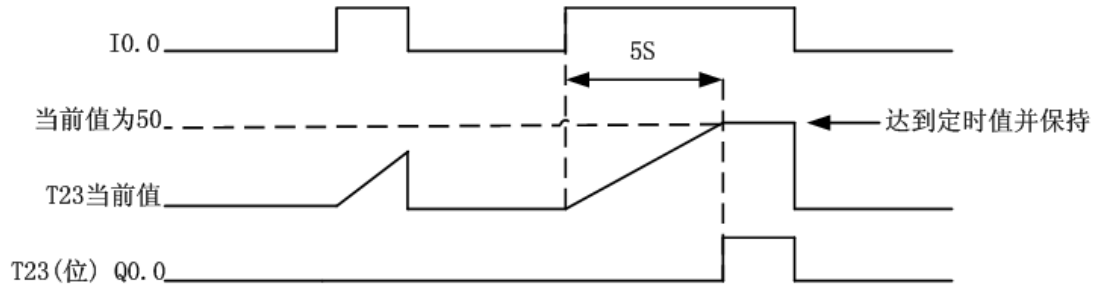
若检测到输入端 *IN* 的上升沿，则 *Tx* 开始启动定时，当计时值 *ET* 大于等于预设值 *PT* 时，*Tx* 停止，其输出 *Q* 及其状态值均被置为 1。若输入 *IN* 变为 0，则 *Tx* 被复位，其输出 *Q* 及状态值均被置为 0，同时计时值 *ET* 也被清零。

- **IL**

若检测到 CR 值的上升沿，则 *Tx* 开始启动定时，当计时值大于等于预设值 *PT* 时，*Tx* 停止，

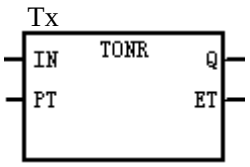
其状态值被置为 1。若 CR 值变为 0，则 *Tx* 被复位，其状态值及计时值均被清零。每次扫描 *TON* 后，CR 值均被设置为 *Tx* 的状态值。

指令使用举例：

LD	IL
	(* Network 0 *) (*启动 T23，预设时间为5S*) LD %I0.0 TON T23, 50 MOVE T23, %VW100 ST %Q0.0
举例结果如下：	
	

5.14.3 TONR（保持型接通延时定时器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	TONR		
IL	TONR	TONR *Tx, PT*	P

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

- **LD**

若检测到输入端 *IN* 的上升沿，则 *Tx* 开始启动定时，当计时值 *ET* 大于等于预设值 *PT* 时，*Tx* 停止，其输出 *Q* 及其状态值均被置为 1。若输入 *IN* 变为 0，*Tx* 保持不变，保持 TONR 定时器的当前值，其输出 *Q* 及状态值均保持不变，输入 IN 再次变为 1 时，TONR 定时器在当前值基础上继续定时。

- **IL**

若检测到 CR 值的上升沿，则 *Tx* 开始启动定时，当计时值大于等于预设值 *PT* 时，*Tx* 停止，其状态值被置为 1。*Tx* 保持不变，保持 TONR 定时器的当前值，其输出 *Q* 及状态值均保持不变，CR 再次变为 1 时，TONR 定时器在当前值基础上继续定时。每次扫描 *TONR* 后，CR 值均被设置为 *Tx* 的状态值。使用复位指令(R)可清除 TONR 的当前值。

说明：定时器指令结合复位指令(R)的使用。

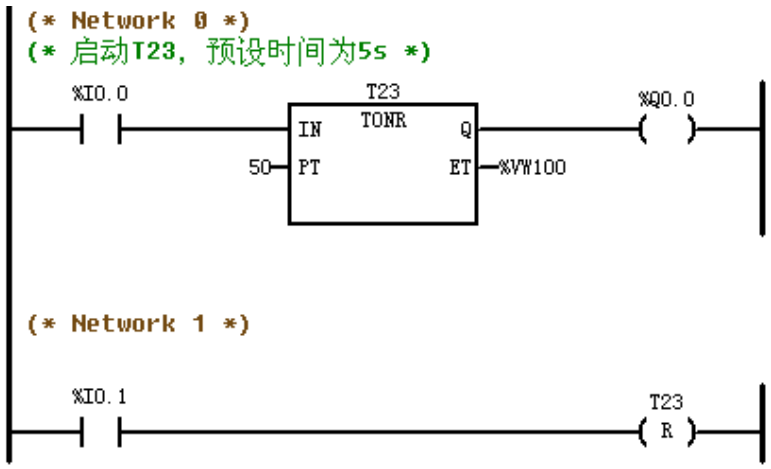
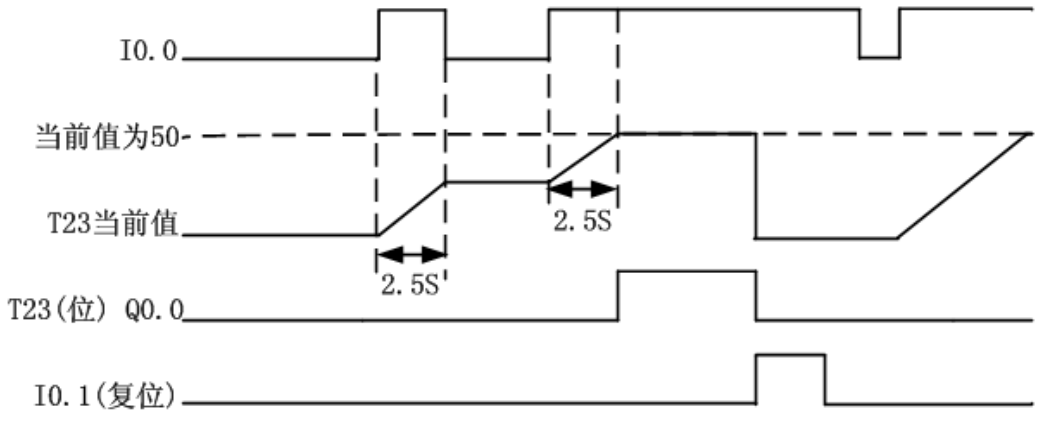
TONR 定时器只能用复位指令(R)复位，并且复位后，即使复位复位指令不在执行，TONR 使能端也需要上升沿才能重新启动。

TON、TOF、TP 定时器不可通过复位指令(R)复位。

复位指令将执行如下操作：定时器状态位清零；定时器当前值清零。

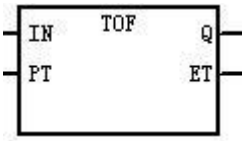
TONR 主要作为掉电保持定时器使用。

指令使用举例：

LD	IL
	(* Network 0 *) (*启动 T23，预设时间为5S*) LD %I0.0 TONR T23, 50 MOVE T23, %VW100 ST %Q0.0
举例结果如下：	
	

5.14.4 TOF（断开延时定时器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	TOF	Tx 	
IL	TOF	TOF *Tx, PT*	P

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

- **LD**

若检测到输入端 *IN* 的下降沿，则 *Tx* 开始启动定时，当计时值 *ET* 大于等于预设值 *PT* 时，*Tx* 停止，其输出 *Q* 及其状态值均被置为 0。若输入 *IN* 变为 1，则 *Tx* 被复位，其输出 *Q* 及状态值均被置为 1，同时计时值 *ET* 被清零。

- **IL**

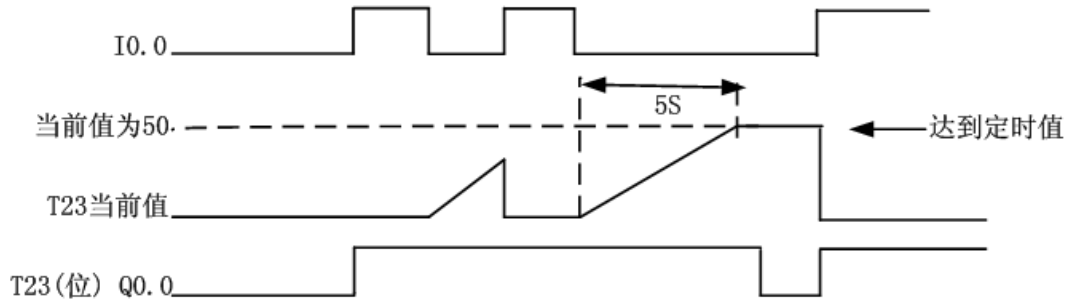
若检测到 CR 值的下降沿，则 *Tx* 开始启动定时，当计时值大于等于预设值 *PT* 时，*Tx* 停止，其状态值被置为 0。若 CR 值变为 1，则 *Tx* 被复位，其状态值被置为 1，且计时值被清零。每次扫描 *TOF* 后，CR 值均被设置为 *Tx* 的状态值。

指令使用举例：

LD	**IL**


```
(* Network 0 *)
(*启动 T23，延时关断为 5S*)
LD      %I0.0
TOF     T23, 50
MOVE    T23, %VW100
ST      %Q0.0
```

举例结果如下：



5.14.5 TP（脉冲定时器）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	TP	Tx IN TP Q PT ET	
IL	TP	TP *Tx, PT*	P

参数	输入/输出	数据类型	允许使用的内存区
Tx	-	定时器实例	T
IN	输入	BOOL	能量流
PT	输入	INT	I、AI、AQ、M、V、L、SM、常量
Q	输出	BOOL	能量流
ET	输出	INT	Q、M、V、L、SM、AQ

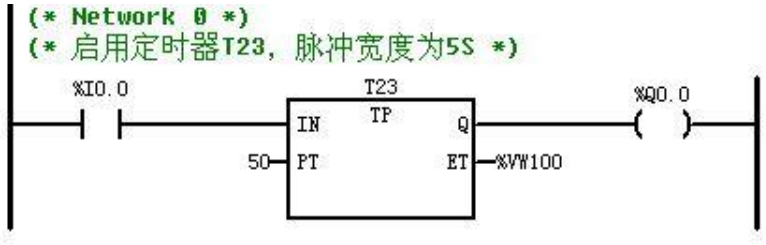
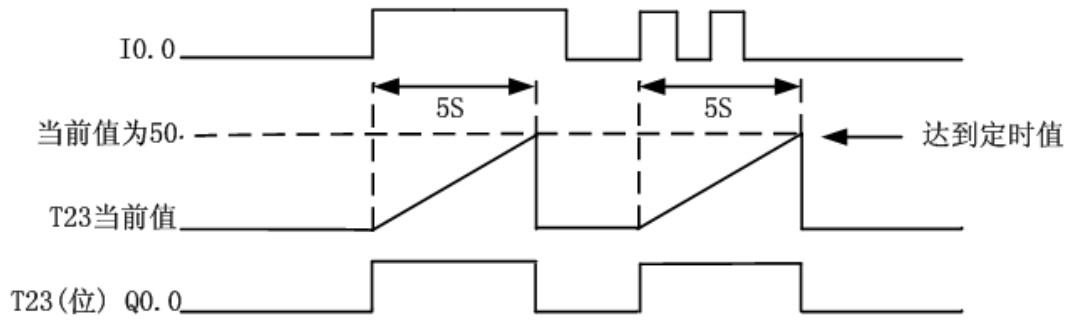
- **LD**

若检测到在输入端 *IN* 的上升沿，则 *Tx* 开始启动定时，在其输出端 *Q* 及其状态值输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。参数 *ET* 中存放着 *Tx* 的计时值。

- **IL**

若检测到 CR 值的上升沿，则 *Tx* 开始启动定时，它的状态值输出一个恒定宽度的脉冲，脉宽值为预设时间 *PT*。每次扫描 *TP* 后，CR 值均被设置为 *Tx* 的状态值。

指令使用举例

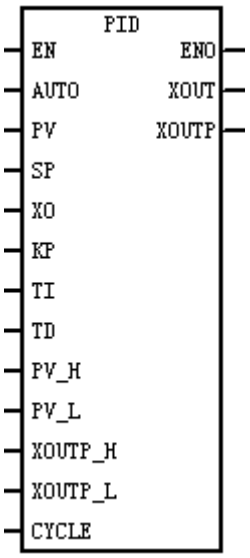
LD	IL
	(* Network 0 *) (*启动 T23，脉冲宽度为 5S*) LD %I0.0 TP T23, 50 MOVE T23, %VW100 ST %Q0.0
举例结果如下：	
	

5.15 PID 回路

在 EC100/EC200 中提供了 PID 指令，它是一种数字 PID 控制器。在用户程序中可以调用 PID 指令来实现 PID 控制功能。在一个 CPU 中用户可以同时使用多达 8 路 PID。

5.15.1 PID

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	PID		
IL	PID	PID *AUTO, PV, SP, XO, KP, TI, TD, PV_H, PV_L, XOUTP_H, XOUTP_L, CYCLE, XOUT, XOUTP*	U

参数	输入/输出	数据类型	允许的内存区	描述
AUTO	输入	BOOL	I、Q、V、M、SM、L、T、C	手动/自动标志位。 0＝手动状态，1＝自动状态。
PV	输入	INT	AI、V、M、L	PV 值，即被控量的测量值。
SP	输入	INT	V、M、L	设定值，即被控量的目标值。
XO	输入	REAL	V、L	手动值，范围是 0.0～1.0。 在手动状态下，它直接是 PID 的输出值。
KP	输入	REAL	V、L	比例系数
TI	输入	REAL	V、L	积分时间，单位：秒。0 表示无积分。
TD	输入	REAL	V、L	微分时间，单位：秒。0 表示无微分。
PV_H	输入	INT	V、L	*PV* 值的上限
PV_L	输入	INT	V、L	*PV* 值的下限
XOUTP_H	输入	INT	V、L	*XOUTP* 值的上限
XOUTP_L	输入	INT	V、L	*XOUTP* 值的下限
CYCLE	输入	DINT	V、M、L	采样周期，单位：ms。 依据采样周期，PLC 循环对 PV 值进行采样并执行 PID 运算。
XOUT	输出	REAL：	V、L	PID 的输出值，范围是 0.0～1.0。
XOUTP	输出	INT	AQ、V、M、L	对 *XOUT* 进行线性化处理后的输出值。

该 PID 指令采用的是位置型算法，连续输出的控制方式。

- **LD**

若 EN 值为 1，则执行 PID 指令：每隔采样周期时间（*CYCLE*），PLC 就对 *PV* 值进行一次采样并进行 PID 运算、输出。

- **IL**

若 CR 值为 1，则执行 PID 指令：每隔采样周期时间（*CYCLE*），PLC 就对 *PV* 值进行一次采样并进行 PID 运算、输出。

PID 指令的执行不影响 CR 值。

➢ PID 详细使用说明

- **手动/自动状态**

若手动/自动标志位 *AUTO* 的值为 0，则表示 PID 进入手动状态。在手动状态下，PID 指令不执行任何运算，直接将用户指定的手动值 *XO* 送至输出变量中。

若手动/自动标志位 *AUTO* 的值为 1，则表示 PID 进入自动状态。在自动状态下，PID 指令根据各输入参数的值进行正常的 PID 运算、输出。

在 PLC 控制系统正常运行时，应该让 PID 处于自动状态。

- **PV 值和 SP 值的归一化**

用户需要输入 *PV* 值、*SP* 值、PV 值的上限（*PV_H*）和 PV 值的下限（*PV_L*）参数。在进行 PID 运算之前，PLC 将依据这些参数自动对 *PV* 值和 *SP* 值进行归一化计算，从而方便了用户的使用。这些参数必须具有相同的量纲。

PV 值、*SP* 值参数均为 INT 型，用户可以灵活地输入与它们线性相关的数值。例如，假定对某个压力进行控制，测压所用压力变送器的量程为 0~40MPa，对应的输出为 4~20mA，自动控制的设定值为 25MPa；压力变送器的的输出直接接至 AI 模块的通道 AIW0（信号形式设置为 4~20mA，在 PLC 内的转换值为 4000~20000）。那么可以将 PID 指令的参数设置为：

	实际参数	**描 述**
PV	AIW0	AIW0 的测量值与实际压力值具有线性关系，因此可以直接作为 PV 值。
SP	14000	表示 14mA，因为 AIW0 测量 25MPa 得到的测量值为 14mA。
PV_L	4000	压力变送器的输出下限。
PV_H	20000	压力变送器的输出上限。

- **PID 的输出值**

PID 指令有两个输出值：*XOUT* 和 *XOUTP*。

XOUT 的范围是 0.0～1.0（也就是通常说的 0.0～100.0%）。

XOUTP 是根据用户指定的输出上限（*XOUTP_H*）和输出下限（*XOUTP_L*）进行线性化变换以后得到的整数值。*XOUTP* 的计算公式如下：

XOUTP ＝ （XOUTP_H － XOUTP_L ）× XOUT ＋ XOUTP_L

XOUTP 参数可以方便用户将 PID 的输出直接送至 AO 模块的某个通道。例如，假定 PID 的输出需要通过 AO 模块的通道 AQW0（信号形式设置为 4～20mA）送至某个调节阀，那么可以将 PID 指令的参数设置为：

	实际参数	描 述
XOUTP	AQW0	AQW0 直接控制调节阀，其值与阀门的开度具有线性关系，因此 AQW0 可以直接作为 PID 的输出参数。
XOUTP_L	4000	AQW0 的输出下限。
XOUTP_H	20000	AQW0 的输出上限。

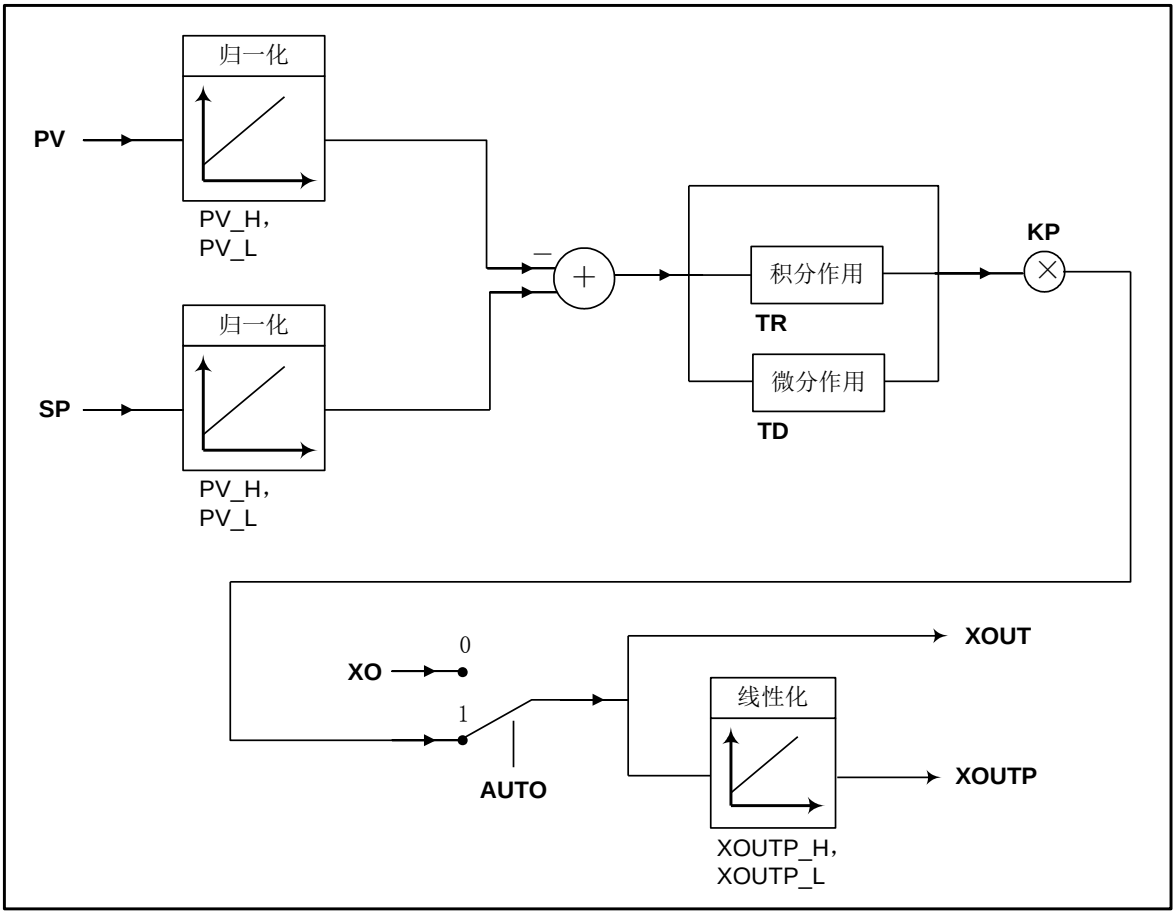
- **关于比例作用、积分作用和微分作用**

比例作用是按照比例来反映被控量的偏差，一旦出现偏差，比例调节器将立即产生调节作用来减少偏差。用户通过 *KP* 参数来设置 PID 的比例系数。若 *KP* 大于 0，则 PID 是正作用；若 *KP* 小于 0，则 PID 是反作用。比例系数大，则调节速度快，但容易造成超调，使系统的稳定性下降。

积分作用能够消除被控量的偏差。只要存在偏差，就会进行积分调节，直至偏差消除。用户通过 TI 参数来设置 PID 的积分时间常数（若 TI 为 0 则表示取消积分作用）。积分时间常数越小，则积分作用就越强；积分时间常数越大，则积分作用就越弱。积分作用强也容易使系统的稳定性下降。积分作用通常与其它两种控制率相结合组成 PI 或者 PID 控制器。

微分作用反映了偏差的变化速率，能够预测偏差的变化趋势，在偏差变得很大之前产生一个有效的修正。因此微分作用有助于改善系统的动态性能，增加系统的稳定性。用户通过 *TD* 参数来设置 PID 的微分时间常数（若 *TD* 为 0 则表示取消微分作用）。微分时间常数越大，则微分作用就越强；微分时间常数越小，则微分作用就越弱。微分作用通常与其它两种控制率相结合组成 PD 或者 PID 控制器。一般在大滞后的系统中（比如温度控制）需要加入微分控制。

PID 指令框图



➢ PID 使用举例

假定我们需要对水温进行控制：

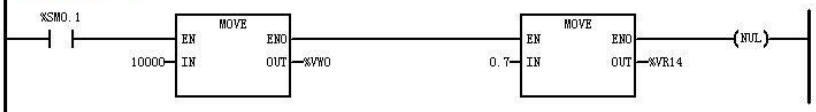
水温采用温度变送器进行测量。温度变送器的的测量范围是 0～250℃，输出信号是 4～20mA 并接至 AI 模块的 AIW0 通道。

我们需要将水温控制在 50℃。

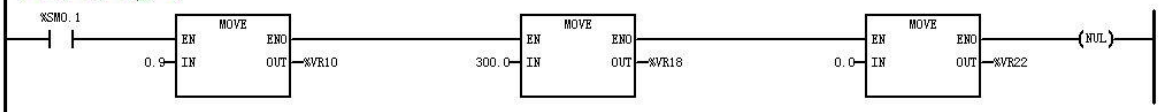
水的加热通过加热器来完成。PLC 通过 AO 模块的 AQW0 通道（信号形式是 4～20mA）来控制加热器。

L
D

```
(* Network 0 *)
(* PID参数初始化
各参数值只做简单举例，实际应用中用户需根据实际情况进行计算
SP值、XO值 *)
```

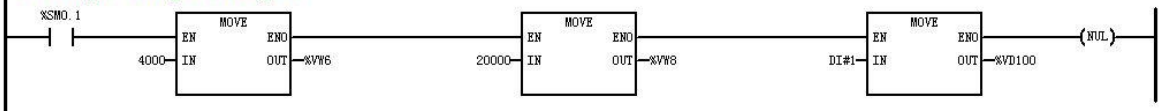



```
(* Network 1 *)
(* KP、TR、TD值 *)
```

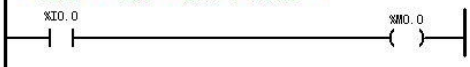



```
(* Network 2 *)
(* PV_H、PV_L值 *)
```

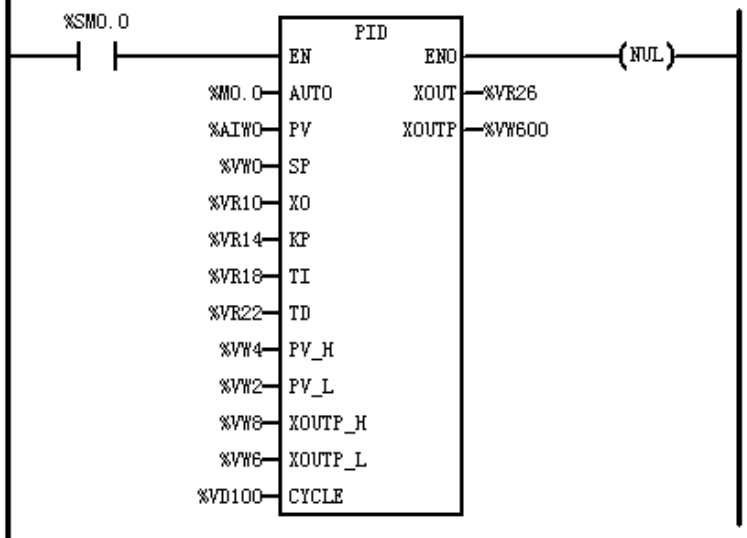

```
(* Network 3 *)
(* XOUTP_H、XOUTP_L、CYCLE值 *)
```



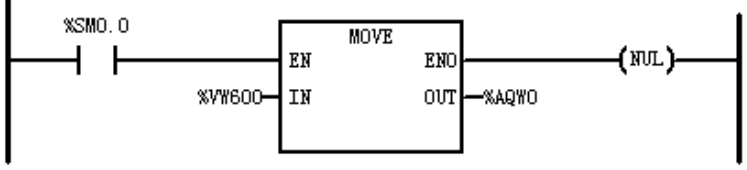

```
(* Network 4 *)
(* 通过M0.0选择PID自动/手动状态 *)
```




```
(* Network 5 *)
(* 调用PID指令进行控制 *)
```




```
(* Network 6 *)
```



续

I L	(* Network 0 *) (*PID 参数初始化　各参数值只做简单举例，实际应用中用户需根据实际情况进行计算　SP 值、XO 值*) LD　　%SM0.1 MOVE　　10000, %VW0 MOVE　　0.7, %VR14 (* Network 1 *) (*KP、TI、TD 值*) LD　　%SM0.1 MOVE　　0.9, %VR10 MOVE　　300.0, %VR18 MOVE　　0.0, %VR22 (* Network 2 *) (*PV_H、PV_L 值*) LD　　%SM0.1 MOVE　　4000, %VW2 MOVE　　20000, %VW4 (* Network 3 *) (*XOUTP_H、XOUTP_L、CYCLE 值*) LD　　%SM0.1 MOVE　　4000, %VW6 MOVE　　20000, %VW8 MOVE　　DI#1, %VD100 (* Network 4 *) (*通过 M0.0 选择 PID 自动/手动状态*) LD　　%I0.0 ST　　%M0.0 (* Network 5 *) (*调用 PID 指令进行控制*) LD　　%SM0.0 PID　　%M0.0, %AIW0, %VW0, %VR10, %VR14, %VR18, %VR22, %VW4, %VW2, %VW8, %VW6, %VD100, %VR26, %VW600 (* Network 6 *) LD　　%SM0.0 MOVE　　%VW600, %AQW0

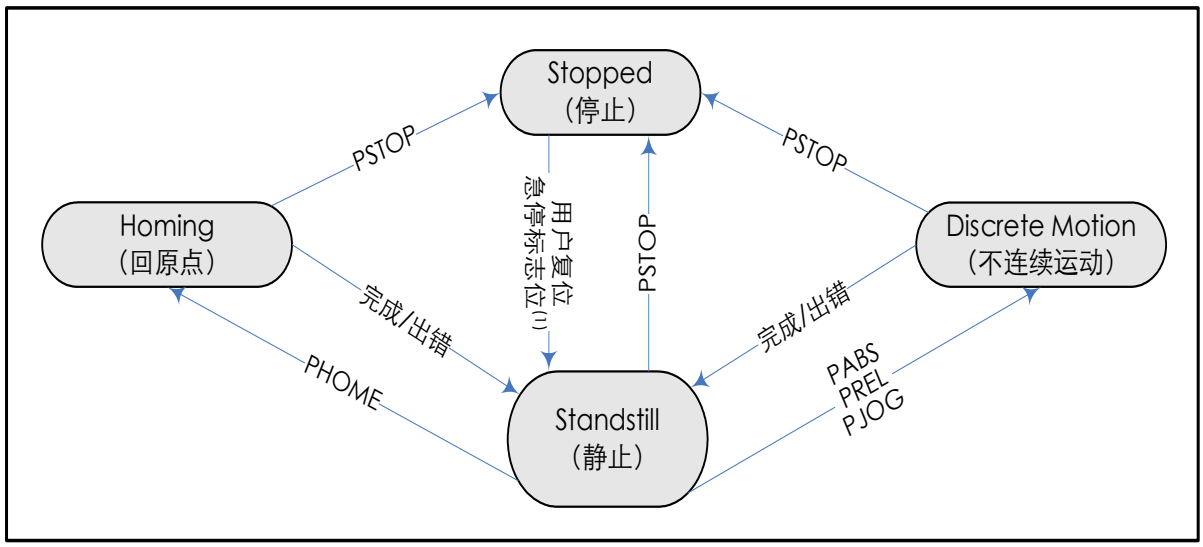
5.16 定位控制（仅适用于 EC200）

EC200 提供了 2 路高速脉冲输出，输出通道分别使用使用 Q0.0 和 Q0.1，可以进行 2 个轴的定位控制。在 5.13.5 PLS（高速脉冲输出）中，对于高速脉冲输出功能及 PLS 指令的使用进行了详细的描述。

本章中描述的定位控制指令是高速脉冲输出功能的又一种使用方法。与 PLS 指令相比，定位控制指令更适合于定位控制应用，用户能够使用它们很方便地实现常见的定位控制功能。需要注意的是，当使用定位控制指令时，输出脉冲频率的允许范围是 20Hz~200KHz，用户指定的输出频率必须在这个范围之内。

5.16.1 定位控制模型图

下图是定位控制的模型图。用户从图中可以了解如下信息：当定位控制指令执行后，相应高速输出通道的状态；在各个状态下，EC200 允许执行的定位控制指令。



(1) 急停标志位就是 SM201.7/ SM231.7，当 PSTOP 指令执行时，该位将自动置 1。

请参阅下面 5.16.2.2 控制寄存器和状态寄存器 一节中的详细描述。

5.16.2 定位控制指令的相关变量

5.16.2.1 电机方向控制信号

针对定位控制指令，EC100/EC200 为每路高速输出均指定了一个方向输出通道，同时还在 SM 区中提供了一个方向使能控制位。如下表。

	Q0.0	Q0.1
方向输出通道	Q0.2	Q0.3
方向使能控制位	SM201.3	SM231.3

方向输出通道用于输出电机的方向控制信号，方向输出通道设置为“0”对应输出方向为正逻辑，方向通道断开，相应的方向指示灯灭。 方向输出通道设置为“1”对应输出方向为负逻辑，方向通道导通，相应的方向指示灯亮。

方向使能控制位用来禁止或者允许使用相应的方向输出通道。方向使能控制位具有最高的优先级，若设置为禁止，那么定位控制指令执行时将不会输出方向控制信号，相应的输出通道就可以作为普通的 DO 点使用。

5.16.2.2 控制寄存器和状态寄存器

针对定位控制指令，EC200 在 SM 区中为每路高速输出均分配了一个控制字节，在应用中用户需要注意设置该控制字节。另外，还分配了一个当前值（DINT 型）寄存器，用于存放当前已经输出的脉冲个数（正转时增加，反转时减少）。下表详细描述了这些寄存器。

Q0.0	**Q0.1**	**描 述**
SM201.7	SM231.7	急停标志位。若该位为 1，则不执行任何定位控制指令。 当 PSTOP（急停）指令执行时，该位将自动置 1。用户需要使用程序将该位清 0。
SM201.4~SM201.6	SM231.4~SM231.6	保留
SM201.3	SM231.3	方向使能控制位。 1 --- 禁止使用指定的方向输出通道； 0 --- 允许使用指定的方向输出通道。
SM201.0~SM201.2	SM231.0~SM231.2	保留
Q0.0	**Q0.1**	**描 述**
SMD212	SMD242	脉冲个数当前值

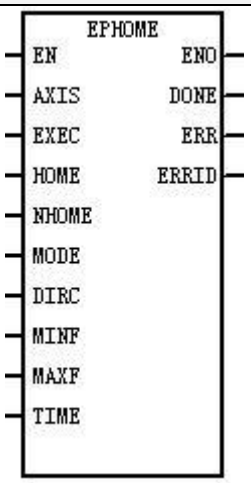
5.16.2.3 错误码

定位控制指令在执行时可能会产生非致命错误，这时候，CPU 就会产生一个错误码并输出至指令的 *ERRID* 参数中。下表列出了这些错误码及其描述。

错误码	**描 述**
0	正常
1	MINF 值或 MAXF 值大于最高允许速度（200KHZ）
2	MINF 值小于最低允许速度（20HZ）
3	MINF 值大于 MAXF 值

5.16.3 EPHOME（回原点，适用于 EC200）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	EPHOME		
IL	EPHOME	EPHOME *AXIS*, *EXEC*, *HOME*, *NHOME*, *MODE*, *DIRC*, *MINF*, *MAXF*, *TIME*, *DONE*, *ERR*, *ERRID*	U

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0 或者 1）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
HOME	输入	BOOL	I 、Q、V、M、L、SM、RS、SR
NHOME	输入	BOOL	I 、Q、V、M、L、SM、RS、SR
MODE	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
DIRC	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
MINF	输入	DWORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

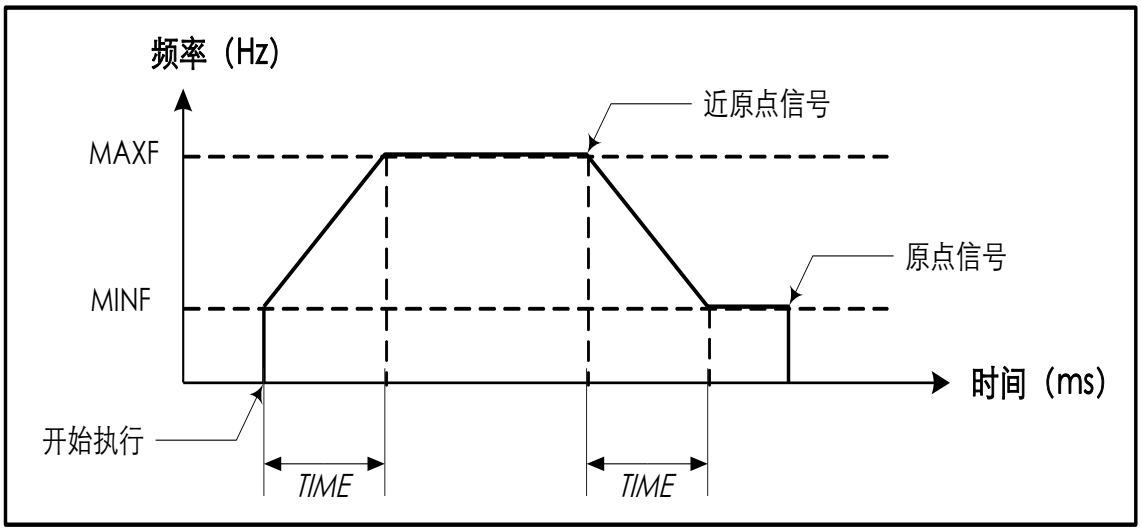
下表对各个参数的作用进行了详细的描述。

参数	描　述

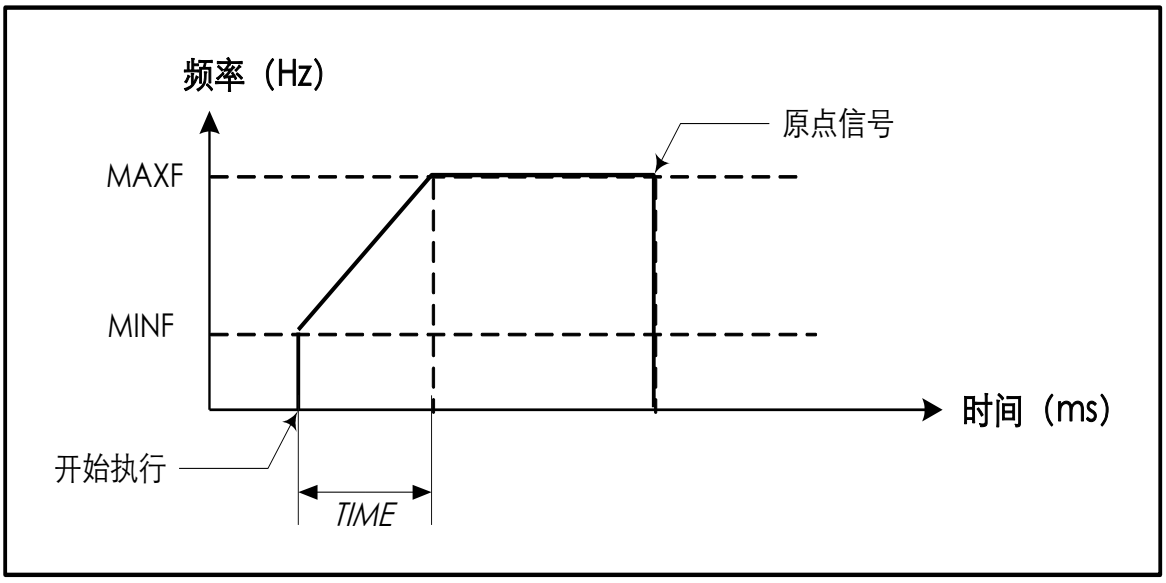
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PHOME 指令被触发执行。
HOME	原点输入信号。
NHOME	近原点输入信号。
MODE	回原点的控制方式： 0 表示利用近原点和原点输入信号进行控制；1 表示只利用原点输入信号进行控制
DIRC	电机的运转方向：0 表示正逻辑；1 表示负逻辑。 关于方向控制信号的输出请参阅 5.16.2.1 电机方向控制信号 中的描述。
MINF	输出脉冲的初始速度（即初始频率），单位：Hz。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 20Hz~200KHz。*MAXF* 必须大于等于 *MINF*。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 *MINF* 加速到 *MAXF* 所需的时间，减速时间是由 *MAXF* 减速到 *MINF* 所需的时间。注：加/减速时间必须是频率最小值对应周期的 20 倍以上。即，加/减速时间 $t \geq \frac{1000000}{F_{min}} \times 20 \times \frac{1}{1000}$，$F_{min}$ 单位为 Hz，t 单位为 ms。如果加减速时间小于最小频率周期的 20 倍，启动回原点指令时将跳过加减速段直接输出最高速度，在近原点信号触发回原点指令时也将跳过加减速段直接输出初始速度。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

EPHOME 指令利用近原点和原点输入信号进行返回原点的控制，参数 *MODE* 定义了控制方式：

① 若利用近原点和原点信号进行控制，则当检测到近原点信号时开始减速，当检测到原点信号时停止脉冲输出。时序图如下：



② 若只利用原点信号进行控制，则当检测到原点信号时停止脉冲输出。时序图如下：



EPHOME 指令执行时，若 *DIRC* 设定为正转，则当前值（SMD212/SMD242）将会增加，若 *DIRC* 设定为反转，则当前值（SMD212/SMD242）将会减少。

需要注意的是：当回原点运动完成后，当前值寄存器（SMD212/SM242）并不自动清零，用户需要根据实际要求自行改变当前值。

- **LD**

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

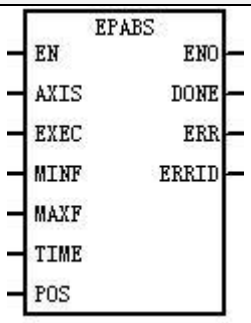
- **IL**

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

5.16.4 EPABS（绝对运动，适用于 EC200）

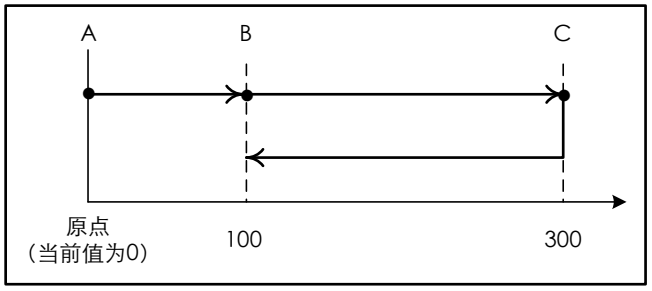
➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	EPABS		
IL	EPABS	EPABS *AXIS, EXEC, MINF, MAXF, TIME, POS, DONE, ERR, ERRID*	U

续

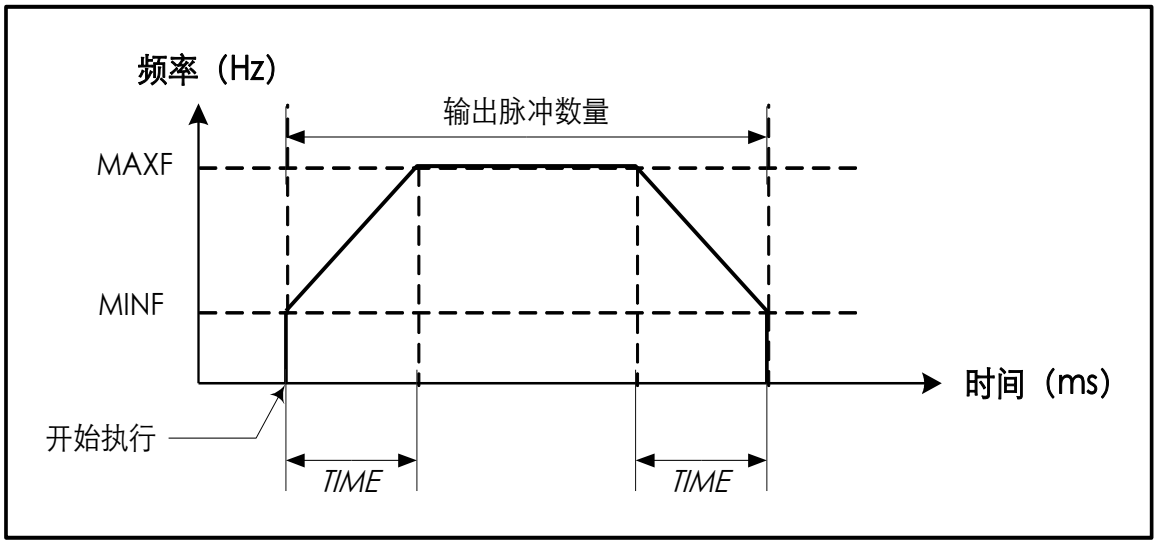
参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0 或者 1）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	WORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	DWORD	I、Q、M、V、L、SM、常量
POS	输入	DINT	I、Q、M、V、L、SM、HC、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PABS 指令被触发执行。
MINF	输出脉冲的初始速度（即初始频率），单位：Hz。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 20Hz~200KHz。*MAXF* 必须大于等于 *MINF*。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 *MINF* 加速到 *MAXF* 所需的时间，减速时间是由 *MAXF* 减速到 *MINF* 所需的时间。注：加/减速时间必须是频率最小值对应周期的 20 倍以上。即，加/减速时间 $t \geq \frac{1000000}{F_{\min}} \times 20 \times \frac{1}{1000}$，$F_{min}$ 单位为 Hz，t 单位为 ms。如果加减速时间小于最小频率周期的 20 倍，触发绝对运动指令时将跳过加减速段直接输出最高速度。
POS	目标值，也就是原点到要移动到的位置之间所需的脉冲个数。 如下图，若将物体由 A 处移到 B 处，则目标值应设定为“100”；若从 B 处移到 C 处，则目标值应设定为“300”；若从 C 处移到 B 处，则目标值设定为“100”。 
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

EPABS 指令采用绝对式定位，根据当前值与目标值 *POS* 之间的差值来输出脉冲。当前值与目标值之间的差值就是输出脉冲的数量。

EPABS 指令执行的时序图如下：



若方向使能控制位（SM201.3/SM231.3）被设置为 0，那么 EPABS 指令将在相应的方向输出通道（Q0.2/Q0.3）输出电机的方向控制信号：当目标值>当前值时，输出正转信号，此时当前值（SMD212/SMD242）将会增加；当目标值<当前值时，输出反转信号，此时当前值（SMD212/SMD242）将会减少。

- **LD**

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

- **IL**

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

5.16.5 EPREL（相对运动，适用于 EC200）

➢ 指令及其操作数说明

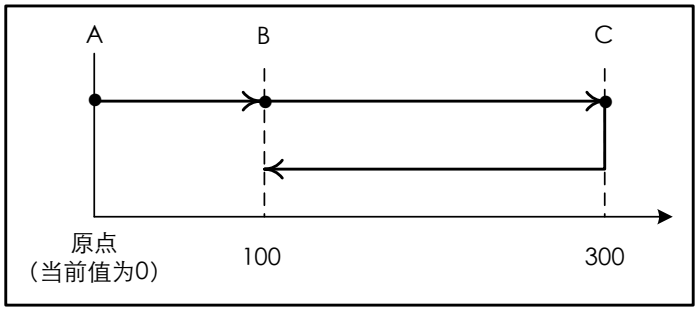
	名称	指令格式	影响 CR 值
LD	EPREL	EPREL EN ENO AXIS DONE EXEC ERR MINF ERRID MAXF TIME DIST	

IL	EPREL	EPREL *AXIS, EXEC, MINF, MAXF, TIME, DIST, DONE, ERR, ERRID*	U

续

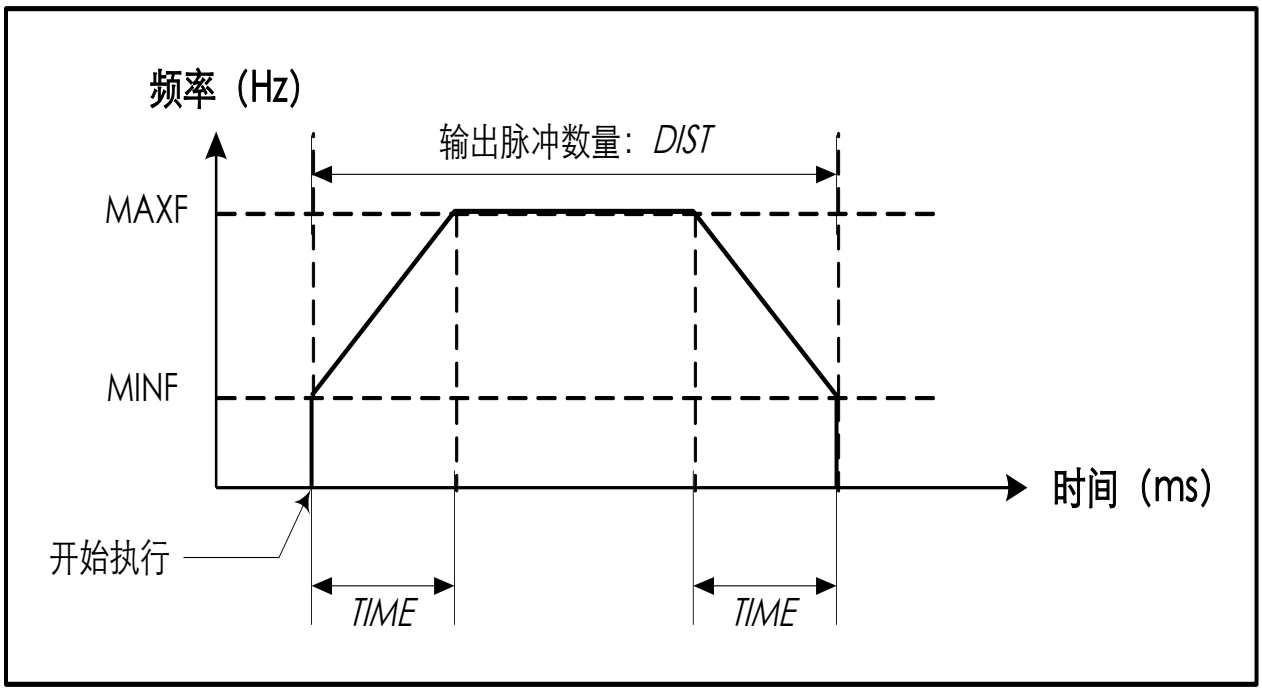
参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0 或者 1）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	DWORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
DIST	输入	DINT	I、Q、M、V、L、SM、HC、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PREL 指令被触发执行。
MINF	输出脉冲的初始速度（即初始频率），单位：Hz。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 20Hz~200KHz。*MAXF* 必须大于等于 *MINF*。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 *MINF* 加速到 *MAXF* 所需的时间，减速时间是由 *MAXF* 减速到 *MINF* 所需的时间。注：加/减速时间必须是频率最小值对应周期的 20 倍以上。即，加/减速时间 $t \geq \frac{1000000}{F_{\min}} \times 20 \times \frac{1}{1000}$，$F_{min}$ 单位为 Hz，t 单位为 ms。如果加减速时间小于最小频率周期的 20 倍，触发相对运动指令时将跳过加减速段直接输出最高速度。
DIST	移动量，也就是当前位置到要移动到的位置之间所需的脉冲个数。 如下图，若将物体由 A 处移到 B 处，则移动量应设定为“100”；若从 B 处移到 C 处，则移动量应设定为“200”；若从 C 处移到 B 处，则移动量设定为“-200”。 
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。

ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

EPREL 指令采用相对式定位，输出脉冲的数量就是移动量 *DIST*。EPREL 指令执行的时序图如下：



若方向使能控制位（SM201.3/SM231.3）被设置为 0，那么 EPREL 指令将在相应的方向输出通道（Q0.2/Q0.3）输出电机的方向控制信号：当移动量为正数时，输出正转信号，此时当前值（SMD212/SMD242）将会增加；当移动量为负数时，输出反转信号，此时当前值（SMD212/SMD242）将会减少。

- **LD**

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

- **IL**

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

5.16.6 EPJOG（点动，适用于 EC200）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值

LD	EPJOG		
IL	EPJOG	EPJOG *AXIS, EXEC, MINF, DIRC, DONE, ERR, ERRID*	U

续

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0 或者 1）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	DWORD	I、Q、M、V、L、SM、常量
DIRC	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1。
EXEC	若 *EXEC* 为 1，则持续输出脉冲；若为 0，则停止输出。
MINF	输出脉冲的频率，单位：Hz。允许范围是 20Hz~200KHz。
DIRC	电机的运转方向：0 表示正逻辑；1 表示负逻辑。 关于方向控制信号的输出请参阅 5.16.2.1 电机方向控制信号 中的描述。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

若 *EXEC* 输入为 1，则 EPJOG 指令从指定的通道 *AXIS* 持续输出脉冲串，频率为 *MINF*。若 *EXEC* 输入为 0，则立即停止输出。

EPJOG 指令执行时，若 *DIRC* 设定为正转，则当前值（SMD212/SMD242）将会增加，若 *DIRC* 设定为反转，则当前值（SMD212/SMD242）将会减少。

- **LD**

  当 *EN* 为 1 时，若 *EXEC* 输入为 1，则该指令被执行。

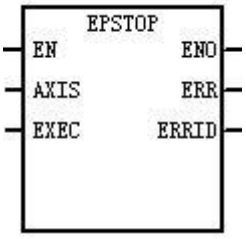
- **IL**

  当 CR 值为 1 时，若 *EXEC* 输入为 1，则该指令被执行。

  该指令的执行不影响 CR 值。

5.16.7 EPSTOP（急停，适用于 EC200）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	EPSTOP		
IL	EPSTOP	EPSTOP *AXIS, EXEC, ERR, ERRID*	U

参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0 或者 1）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

下表对各个参数的作用进行了详细的描述。

参数	描　述
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1。
EXEC	若检测到 *EXEC* 的上升沿跳变，则 PSTOP 指令被触发执行。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

EPSTOP 指令用于立即停止 *AXIS* 通道的脉冲输出，从而使当前的运动紧急停止，同时将急停标志位（SM201.7/SM231.7）置位。用户需要使用程序将急停标志位清 0，否则 CPU 不会再执行任何定位控制指令。

- **LD**

  当 *EN* 为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

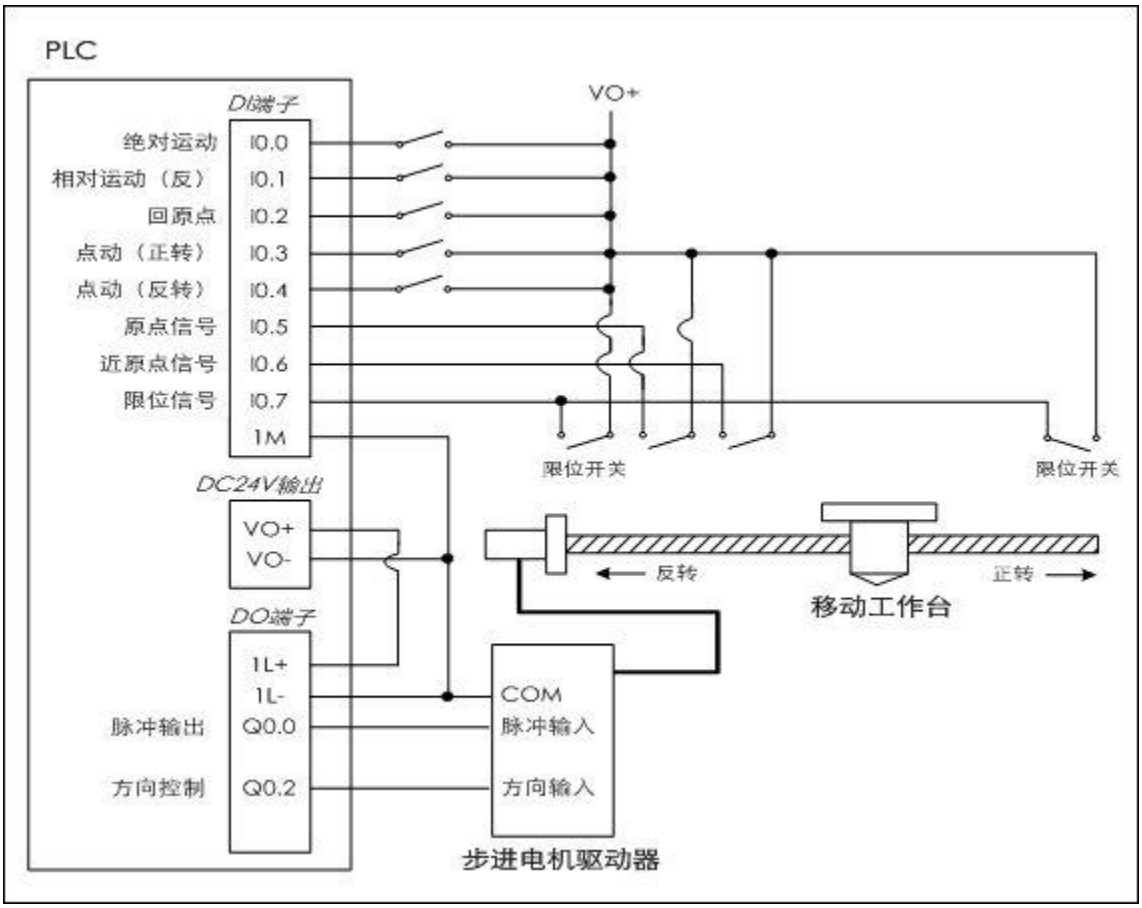
- **IL**

  当 CR 值为 1 时，若检测到 *EXEC* 输入端的上升沿，则该指令被触发执行。

  该指令的执行不影响 CR 值。

5.16.8 定位控制指令示例

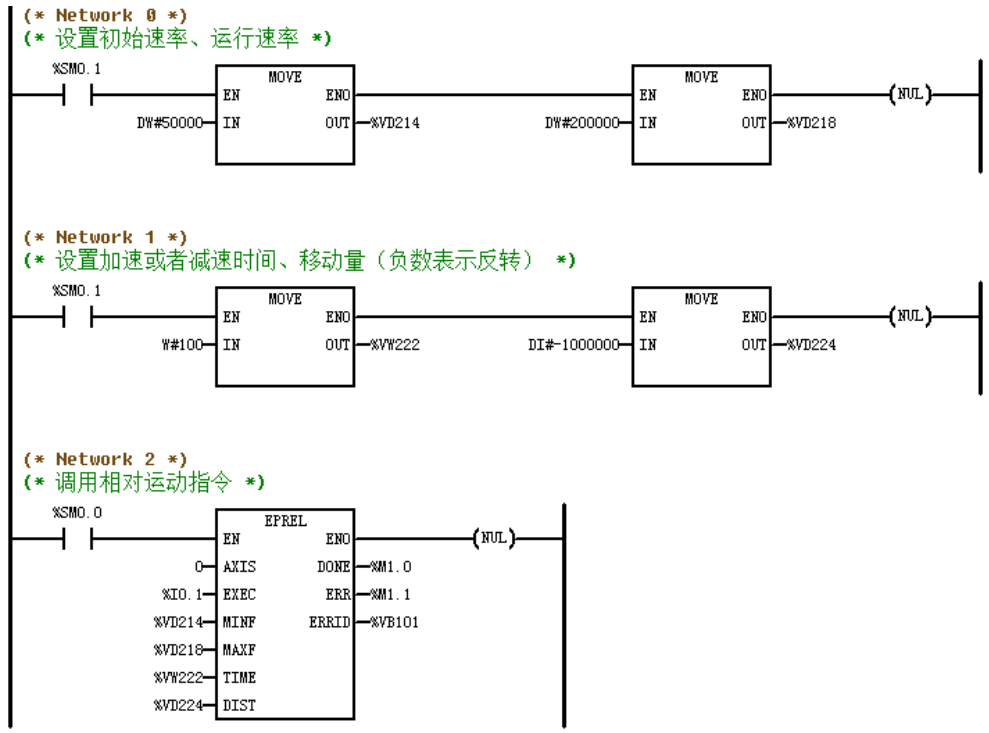
- 接线示意图



下面我们将依据这个系统来举例描述 EPREL、EPABS、EPHOME、EPJOG 和 EPSTOP 指令的使用方式。

相对运动（反转）：I0.1 所接的“相对运动（反）”信号用于启动相对运动（反转）。

LD

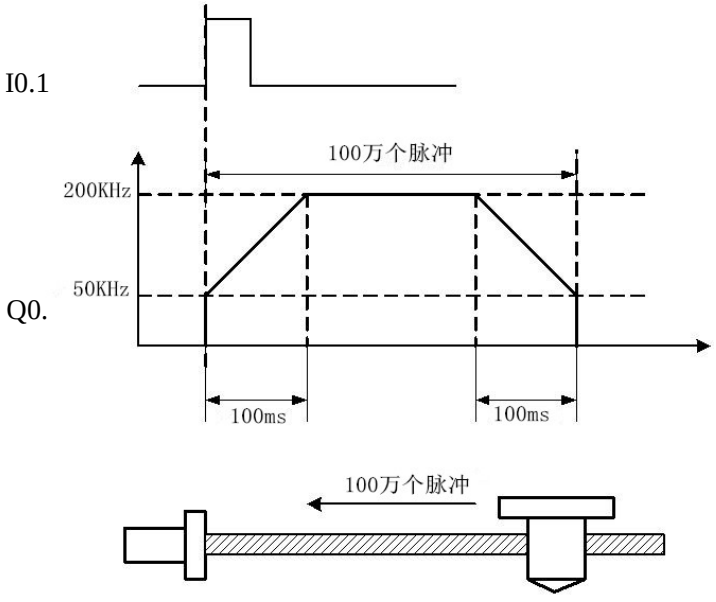


IL

```
(* Network 0 *)
(*设置初始速率、运行速率*)
LD        %SM0.1
MOVE      DW#50000, %VD214
MOVE      DW#200000, %VD218
(* Network 1 *)
(*设置加速或者减速时间、移动量（负数表示反转）*)
LD        %SM0.1
MOVE      W#100, %VW222
MOVE      DI#-1000000, %VD224
(* Network 2 *)
(*调用相对运动指令*)
LD        %SM0.0
EPREL     0, %I0.1, %VD214, %VD218, %VW222, %VD224, %M1.0, %M1.1, %VB101
```

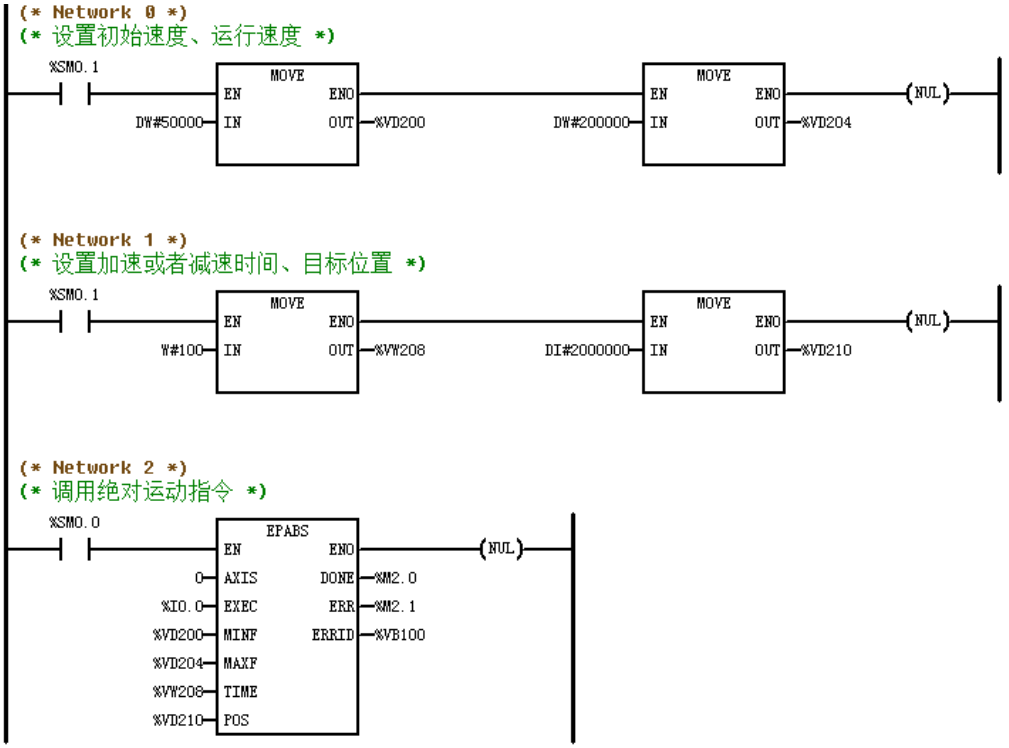
描述

若检测到 I0.1 的上升沿，则在 Q0.0 输出脉冲。Q0.2 输出为 1 表示反转。



绝对运动：I0.0 所接的“绝对运动”信号用于启动绝对运动。

LD

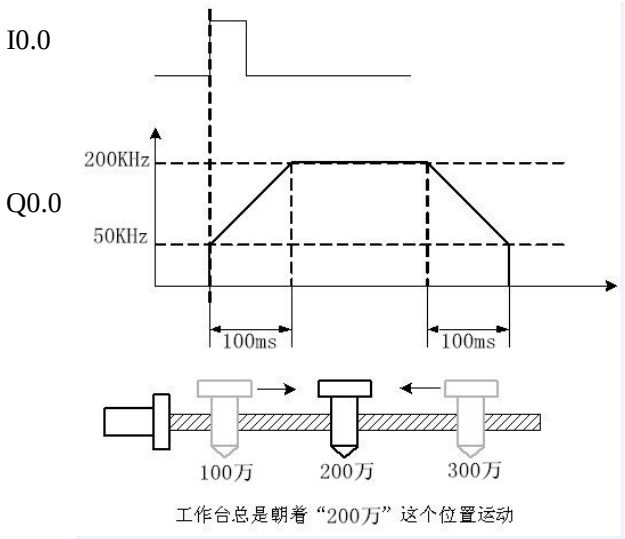


IL

```
(* Network 0 *)
(*设置初始速度、运行速度*)
LD        %SM0.1
MOVE      DW#50000, %VD200
MOVE      DW#200000, %VD204
(* Network 1 *)
(*设置加速或者减速时间、目标位置*)
LD        %SM0.1
MOVE      W#100, %VW208
MOVE      DI#2000000, %VD210
(* Network 2 *)
(*调用绝对运动指令*)
LD        %SM0.0
EPABS     0, %I0.0, %VD200, %VD204, %VW208, %VD210, %M2.0, %M2.1, %VB100
```

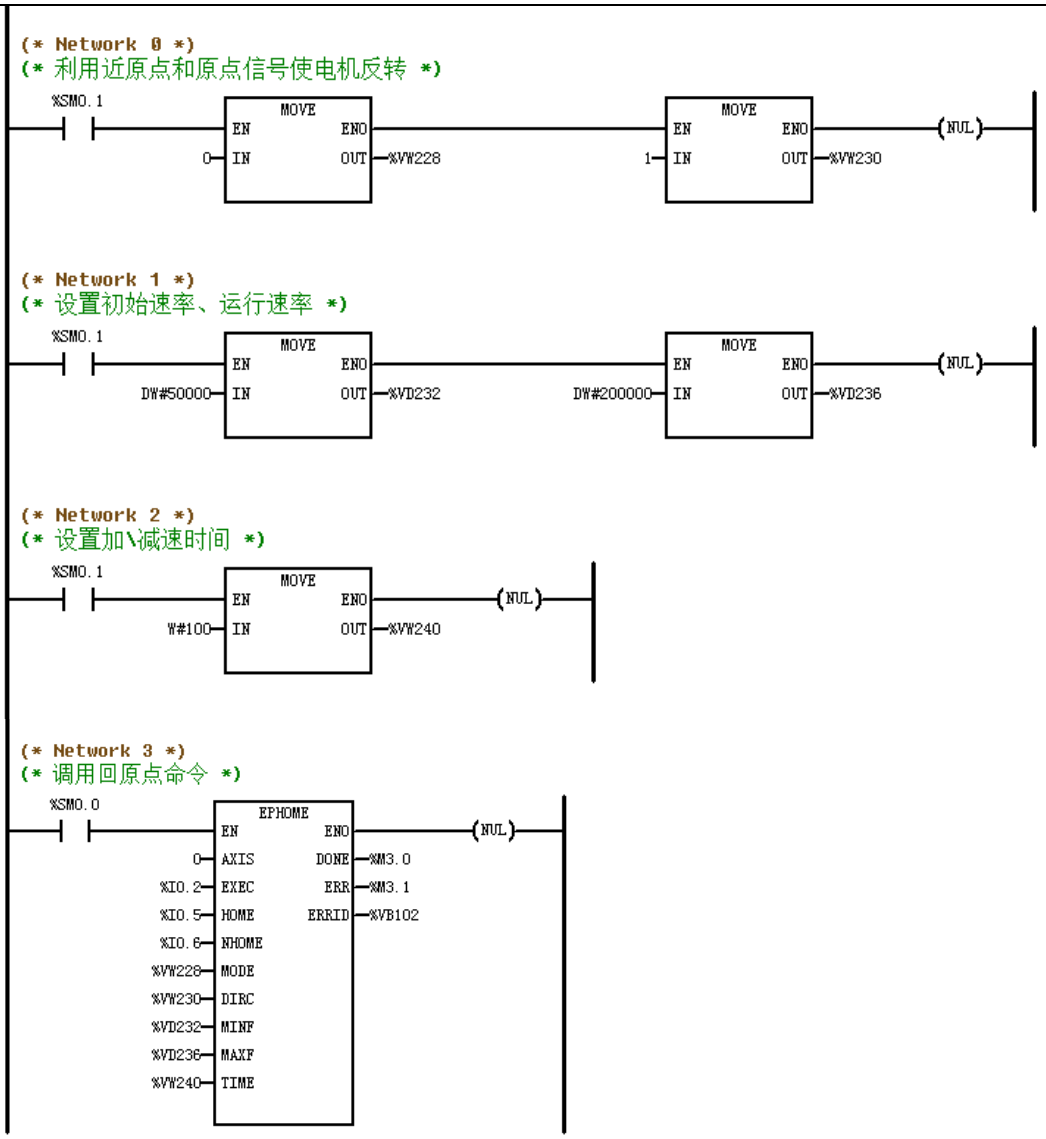
描述

若检测到 I0.0 的上升沿，则在 Q0.0 输出脉冲。Q0.2 输出为 0 表示正转，输出为 1 表示反转。



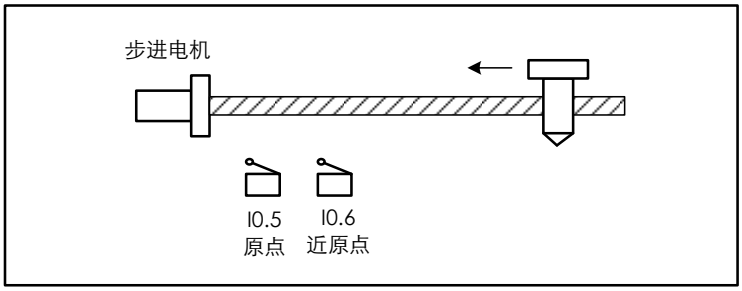
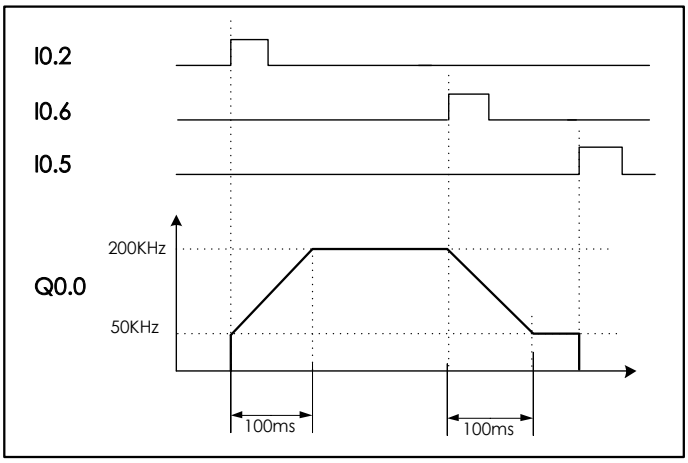
回原点：I0.2 所接的“回原点”信号用于启动回原点运动。需要注意的是：当回原点运动完成后，当前值寄存器（SMD212/SM242）并不自动清零，用户需要根据实际要求自行改变当前值。

LD



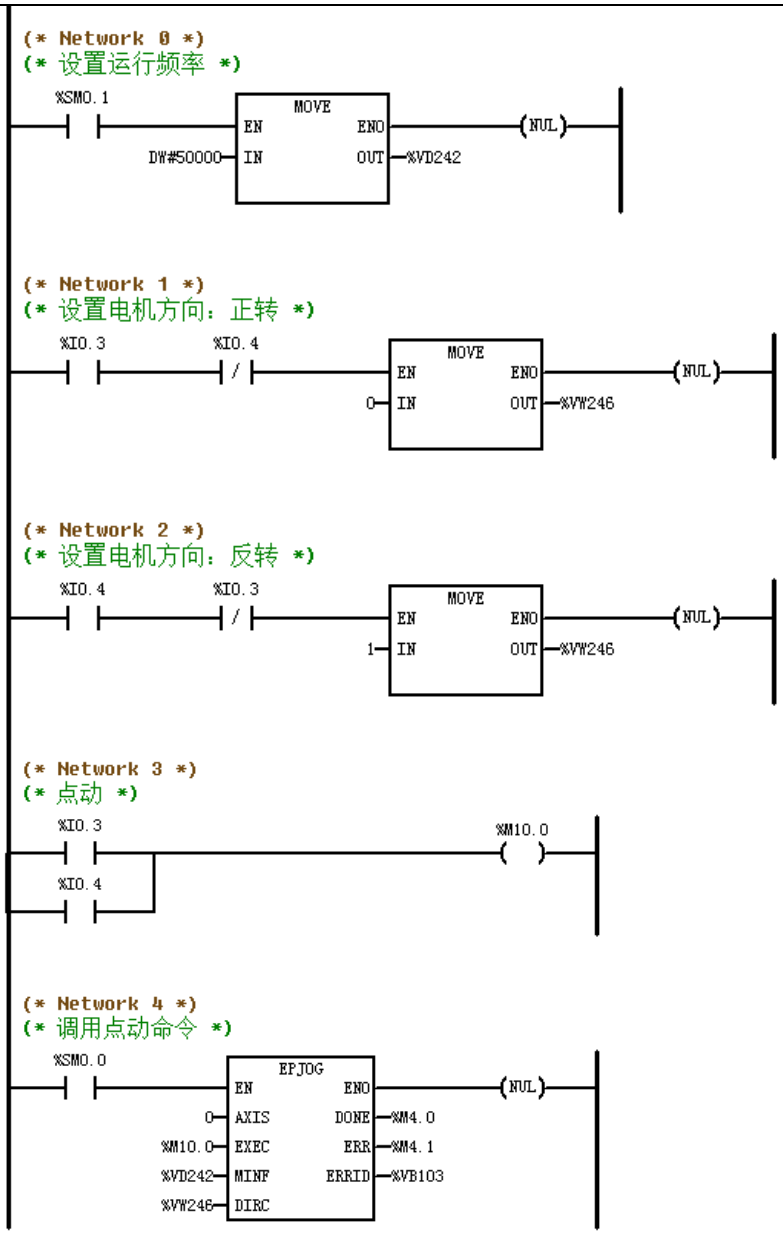
IL

```
(* Network 0 *)
(*利用近原点和原点信号使电机反转*)
LD        %SM0.1
MOVE      0, %VW228
MOVE      1, %VW230
(* Network 1 *)
(*设置初始速率、运行速率*)
LD        %SM0.1
MOVE      DW#50000, %VD232
MOVE      DW#200000, %VD236
(* Network 2 *)
(*设置加\减速时间*)
LD        %SM0.1
MOVE      W#100, %VW240
(* Network 3 *)
(*调用回原点命令*)
LD        %SM0.0
EPHOME
0, %I0.2, %I0.5, %I0.6, %VW228, %VW230, %VD232, %VD236, %VW240, %M3.0, %M3.1, %VB102
```

描述	假定当前工作台处于如下状态：  若检测到 I0.2 的上升沿，则在 Q0.0 输出脉冲。Q0.2 输出为 1 表示反转。 

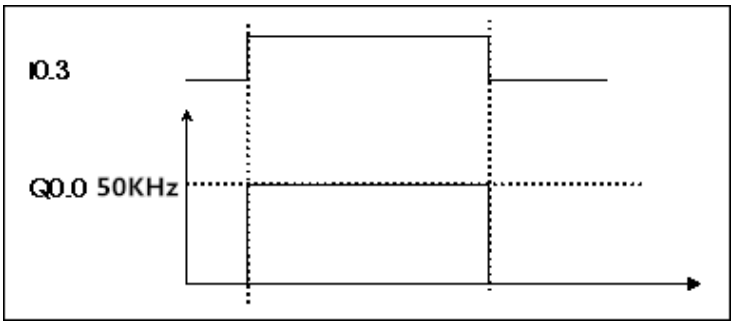
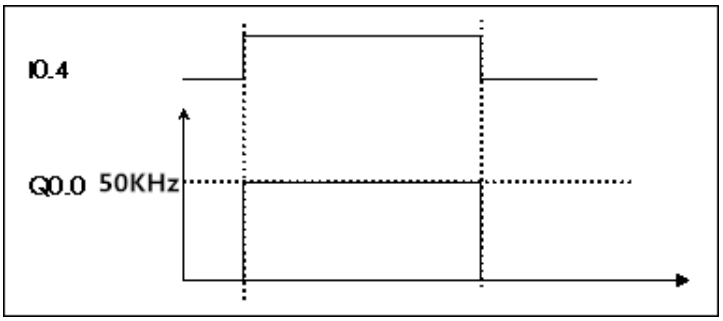
点动：I0.3 所接的“点动（正转）”信号用于点动（电机正转）。I0.4 所接的“点动（反转）”信号用于点动（电机反转）。若 I0.3 和 I0.4 同时接通，则采用上一次的电机转向。

LD

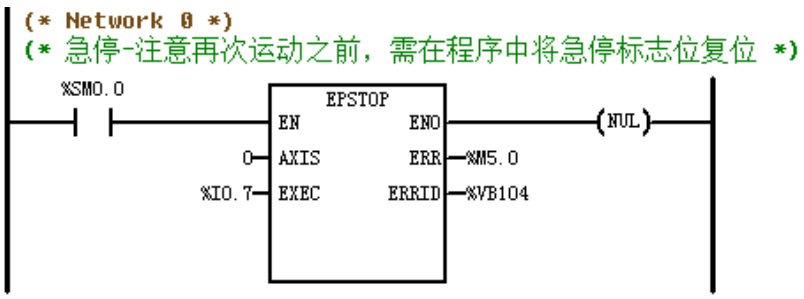


IL

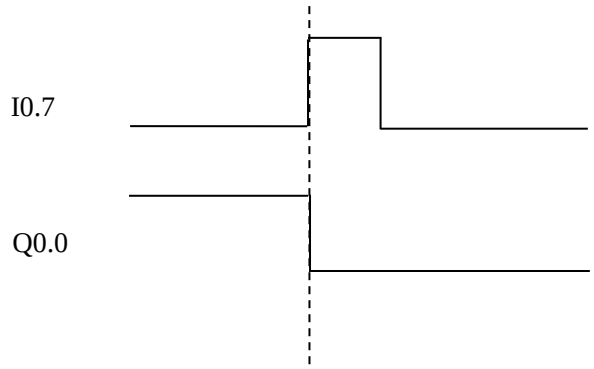
```
(* Network 0 *)
(*设置运行频率*)
LD      %SM0.1
MOVE    DW#50000, %VD242
(* Network 1 *)
(*设置电机方向：正转*)
LD      %I0.3
ANDN    %I0.4
MOVE    0, %VW246
(* Network 2 *)
(*设置电机方向：反转*)
LD      %I0.4
ANDN    %I0.3
MOVE    1, %VW246
(* Network 3 *)
(*点动*)
LD      %I0.3
OR      %I0.4
ST      %M10.0
(* Network 4 *)
(*调用点动命令*)
LD      %SM0.0
EPJOG   0, %M10.0, %VD242, %VW246, %M4.0, %M4.1, %VB103
```

描述	若 I0.3 为 1，则 Q0.0 持续输出脉冲串。若 I0.3 为 0，则 Q0.0 立即停止输出。Q0.2 输出为 0 表示电机正转。  若 I0.4 为 1，则 Q0.0 持续输出脉冲串。若 I0.4 为 0，则 Q0.0 立即停止输出。Q0.2 输出为 1 表示电机反转。 

急停：在丝杠两端有两个限位开关，这两个开关并联接入 I0.7 作为急停信号

LD	IL
 	(* Network 0 *) (*急停-注意再次运动之前，需在程序中将急停标志位复位*) LD %SM0.0 EPSTOP 0, %I0.7, %M5.0, %VB104

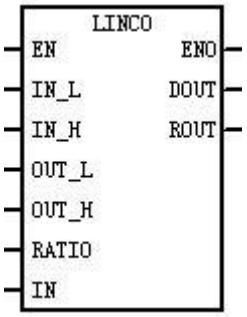
举例结果如下：若检测到 I0.7 的上升沿，则 Q0.0 立即停止输出。



5.17 附加指令

5.17.1 LINCO（线性变换）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	LINCO		
IL	LINCO	LINCO *IN_L, IN_H, OUT_L, OUT_H, RATIO, IN, DOUT, ROUT*	U

续

参数	输入/输出	数据类型	允许的内存区
IN_L	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
IN_H	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ、常量
OUT_L	输入	REAL	V、L、常量
OUT_H	输入	REAL	V、L、常量
RATIO	输入	REAL	常量
IN	输入	INT	I、Q、V、M、L、SM、T、C、AI、AQ
DOUT	输出	DINT	Q、M、V、L、SM
ROUT	输入	REAL	V、L

注意：参数 *IN_L*,、*IN_H*、*OUT_L* 和 *OUT_H* 必须全为常量或者全为变量。

LINCO 指令将输入 *IN* 按照指定的线性关系进行计算，并将计算结果乘以系数 *RATIO* 后赋给 *ROUT*，然后将 *ROUT* 取整（舍弃小数）后赋给 *DOUT*。所用的线性关系如下：输入下限 *IN_L* 和输出下限 *OUT_L*、输入上限 *IN_H* 和输出上限 *OUT_H*，将这些参数按照“两点定一直线”的方法进行计算，从而得到其中的线性关系。

LINCO 指令的作用可以用如下公式描述：

$$ROUT = RATIO \times (k \times IN + b)$$

$$DOUT = \mathrm{TRUNC}(ROUT)$$

其中 $k = \dfrac{OUT\_H - OUT\_L}{IN\_H - IN\_L}$ ； $b = OUT\_L - k \times IN\_L$ 。

- **LD**

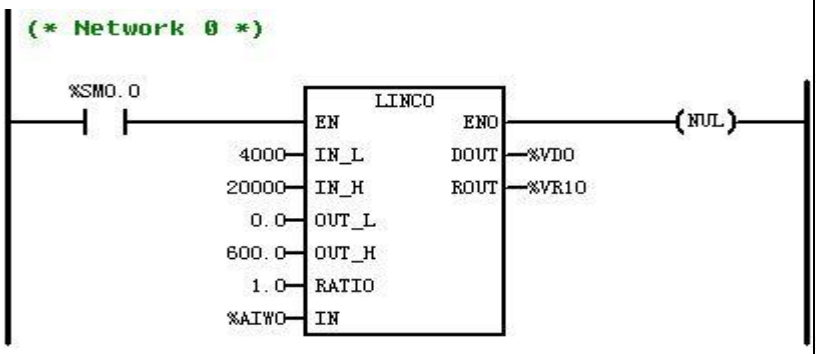
如果 *EN* 为 1，则该指令被执行。

- **IL**

如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

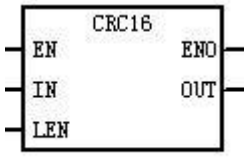
指令使用举例：

假定现场一个温度变送器的测量范围是 0∽600℃，信号输出范围是 4∽20mA。变送器的输出信号接至 PLC 的 AIW0 通道。要求 PLC 计算出实际的温度值。

LD	IL
	(* Network 0 *) LD %SM0.0 LINCO 4000, 20000, 0.0, 600.0, 1.0, %AIW0, %VD0, %VR10

5.17.2 CRC16（16 位 CRC 校验码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值
LD	CRC16		
IL	CRC16	CRC16 *IN, OUT, LEN*	U

参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM
LEN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

该指令用于计算指定数据的 16 位 CRC 校验码。被校验的数据存放在从 *IN* 开始连续 *LEN* 个字节长度的区域内。计算结果存放在从 *OUT* 开始连续 2 个字节长度的区域内，其中，*OUT* 存放着 CRC 码的高字节，*OUT* 的下一个字节存放着 CRC 码的低字节。

- **LD**

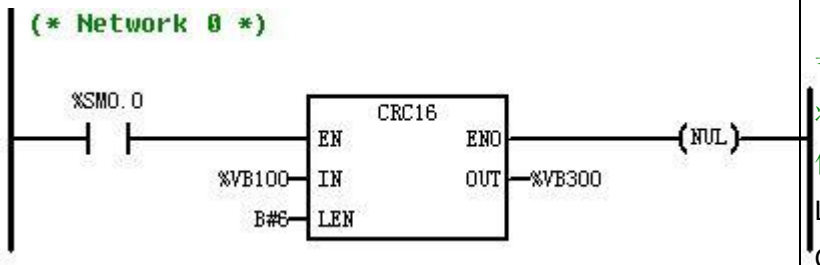
若 *EN* 值为 1，则该指令被执行，否则不执行。

- **IL**

  如果 CR 值为 1，则该指令被执行，否则不执行。

  该指令的执行不影响 CR 值。

指令使用举例：

LD	IL
	(* Network 0 *) （*将 VB100 开始连续 6 个字节的数据进行 CRC 校验，并将校验码高字节写入 VB300，低字节写入 VB301） LD %SM0.0 CRC16 %VB100, %VB300, B#6

附录 A　使用 Modbus RTU 协议通讯

Modbus RTU 是一种主从通讯协议。在默认的情况下，EC100/EC200 的 CPU 采用 Modbus RTU 协议进行通讯并作为从站。可通过【**硬件配置**】中的【**通讯设置**】设置为 Modbus 主站。

1、PLC 内存区说明

1.1　可访问的内存区

Modbus RTU 主站可访问作为从站的 EC100/EC200 内存区域分类如下：

类型	**Modbus 功能码**	对应的 PLC 内存区域
DO（开关量输出，0XXXX）	01，05，15	Q 区，M 区
DI（开关量输入，1XXXX）	02	I 区，M 区
AO（模拟量输出，4XXXX）	03，06，16	AQ 区，V 区
AI （模拟量输入，3XXXX）	04	AI 区，V 区

1.2　Modbus 寄存器与内存区域对照

*注：某些 Modbus RTU 主站的寄存器从 1 开始编号，此时将表中的数据直接加 1 即可。

- **适用于 EC102**

内存区域	范围	类型	对应的 Modbus 寄存器*
I	I0.0 --- I0.7	DI	0 --- 7
Q	Q0.0 --- Q0.5	DO	0 --- 5
M	M0.0 --- M63.7	DI/DO	320 -- 831
AI	----	AI	----
AQ	----	AO	----
V	VW0 ---VW6142	AI/AO	100 -- 3171

- **适用于 EC104**

内存区域	范围	类型	对应的 Modbus 寄存器*
I	I0.0 --- I1.5	DI	0 --- 13
Q	Q0.0 --- Q1.1	DO	0 --- 9
M	M0.0 --- M63.7	DI/DO	320 -- 831
AI	----	AI	----
AQ	----	AO	----
V	VW0 ---VW6142	AI/AO	100 -- 3171

➢ **适用于 EC106**

内存区域	范围	类型	对应的 Modbus 寄存器*
I	I0.0 --- I2.7	DI	0 --- 23
Q	Q0.0 --- Q1.7	DO	0 --- 15
M	M0.0 --- M63.7	DI/DO	320 -- 831
AI	----	AI	----
AQ	----	AO	----
V	VW0 ---VW6142	AI/AO	100 -- 3171

➢ **适用于 EC202、EC204、EC206**

内存区域	范围	类型	对应的 Modbus 寄存器*
I	I0.0 --- I31.7	DI	0 --- 255
Q	Q0.0 --- Q31.7	DO	0 --- 255
M	M0.0 --- M127.7	DI/DO	320 -- 1343
AI	AIW0 --- AIW62	AI	0 --- 31
AQ	AQW0 --- AQW62	AO	0 --- 31
V	VW0 ---VW10238	AI/AO	100 -- 5219

2、Modbus RTU 报文基本格式

报文中的 CRC 校验码是高字节在前，低字节在后。

不小于 3.5 个字符长度的报文间隔时间	目标站号	功能码	数据	CRC 校验码
	1 字节	1 字节	N 字节	2 字节

2.1 Modbus RTU 命令简介

下面对于各请求命令的“应答格式”的描述是从站正确的应答格式。若是异常应答，那么返回的应答帧中“功能码”部分变为如下数据：功能码的最高位置“1”后得到的数据。比如功能码为 0x01，若从站是异常的应答，则返回的功能码为 0x81。

2.1.1 功能码 01：读线圈（开关量输出）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	01	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	…	CRC
1 字节	01	1 字节	1 字节	1 字节	…	2 字节

2.1.2 功能码 02：读输入状态（开关量输入）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	02	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	返回数据字节 1	返回数据字节 2	…	CRC
1 字节	02	1 字节	1 字节	1 字节	…	2 字节

2.1.3 功能码 03：读保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	03	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	寄存器 1 高字节	寄存器 1 低字节	…	CRC
1 字节	03	1 字节	1 字节	1 字节	…	2 字节

2.1.4 功能码 04：读输入寄存器（模拟量输入）

请求格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	04	高字节	低字节	高字节	低字节	2 字节

正确应答格式：

站号	功能码	返回数据字节数	寄存器 1 高字节	寄存器 1 低字节	…	CRC
1 字节	04	1 字节	1 字节	1 字节	…	2 字节

2.1.5 功能码 05：写单线圈（开关量输出）

请求格式：

目标站号	功能码	线圈地址		强制值		CRC
1 字节	05	高字节	低字节	高字节	低字节	2 字节

注：强制值 ＝ 0xFF00，则置线圈为接通；强制值＝0x0000，则置线圈为断开。

应答格式：若设置成功，则原报文返回

2.1.6 功能码 06：写单保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	寄存器地址		强制值		CRC
1 字节	06	高字节	低字节	高字节	低字节	2 字节

应答格式：若设置成功，则原报文返回

2.1.7 功能码 15：写多线圈（开关量输出）

请求格式：

目标站号	功能码	起始地址		写入个数		强制值字节数	强制值第 1 字节	…	CRC
1 字节	15	高字节	低字节	高字节	低字节	1 字节	1 字节	…	2 字节

正确应答格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	15	高字节	低字节	高字节	低字节	2 字节

2.1.8 功能码 16：写多保持寄存器（模拟量输出）

请求格式：

目标站号	功能码	起始地址		写入个数		强制值字节数	强制值1高字节	强制值1低字节	…	CRC
1 字节	16	高字节	低字节	高字节	低字节	1 字节	1 字节	1 字节	…	2 字节

正确应答格式：

目标站号	功能码	起始地址		读取个数		CRC
1 字节	16	高字节	低字节	高字节	低字节	2 字节

2.2 Modbus 协议中的 CRC 校验算法

在 Modbus RTU 协议中，使用 CRC 作为帧的校验方式。下面是用 C 编写的两种 CRC 算法。

2.2.1 直接计算 CRC

```
/* 参   数：chData ——  const BYTE*，指向待校验数据存储区的首地址
                uNO   ——  待校验数据的字节个数
   返回值：WORD 型，计算出的 CRC 值。                    */
WORD   CalcCrc(const BYTE* chData, WORD uNo)
{
    WORD crc=0xFFFF;
    WORD wCrc;
    UCHAR i,j;
    for (i=0; i<uNo; i++)
    {
        crc ^= chData[i];
        for (j=0; j<8; j++)
        {
            if (crc & 1)
            {
                crc >>= 1;
                crc ^= 0xA001;
            }
            else
                crc >>= 1;
        }
    }

    wCrc=( (WORD)LOBYTE(crc) )<<8;
```

```
    wCrc=wCrc|( (WORD)HIBYTE(crc) );
    return (wCrc);
}
```

2.2.2 查表快速计算 CRC

```
/* 高字节 CRC 表 */
const UCHAR auchCRCHi[] =
{
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
        0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
        0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
        0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
        0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
        0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
        0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
} ;

/* 低字节 CRC 表 */
const UCHAR auchCRCLo[] =
{
```

```
            0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
            0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
            0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
            0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
            0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
            0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
            0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
            0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
            0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
            0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
            0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
            0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
            0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
            0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
            0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
            0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
            0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
            0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
            0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
            0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
            0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
            0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
            0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
            0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
            0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
            0x43, 0x83, 0x41, 0x81, 0x80, 0x40
};

/* 参   数：puchMsg   －－ const BYTE*，指向待校验数据存储区的首地址
            usDataLen －－ 待校验数据的字节个数
  返回值：WORD 型，计算出的 CRC 值。     */
WORD CKDNSerialCom：：CalCrcFast(const BYTE* puchMsg , WORD usDataLen)
{
    BYTE uchCRCHi = 0xFF ;   /* CRC 高字节初始化  */
    BYTE uchCRCLo = 0xFF ;   /* CRC 低字节初始化  */
    WORD uIndex ;            /* CRC 查表的索引*/

    while (usDataLen--)
    {
        uIndex = uchCRCHi ^ *puchMsg++ ;          /*  计算  CRC */
```

```
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];
        uchCRCLo = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
```

附录 B　特殊存储器 SM 区的定义

本附录详细描述了系统存储区 SM 区中的相关定义。系统存储区是用来辅助 EC100/EC200 实现特定的指令功能，用户也可以通过使用系统存储器来读取 PLC 的某些状态。

1、SMB0：系统状态

SM0.0--SM0.7 由 CPU 的运行软件进行赋值，不受用户程序控制，在用户程序中只能对这些位调用（只读）。详细的功能描述请参见下表。

位(只读)	描 述
SM0.0	总是为“1”。
SM0.1	PLC 进入 Run 状态后，该位将置 1 一个扫描周期的时间。 在 CPU 首次扫描时为“1”，之后清“0”。通常用于用户程序初始化。
SM0.2	周期为 10ms 的时钟脉冲，占空比为 50%。
SM0.3	周期为 100ms 的时钟脉冲，占空比为 50%。
SM0.4	周期为 1s 的时钟脉冲，占空比为 50%。
SM0.5	周期为 60s 的时钟脉冲，占空比为 50%。
SM0.6	该位为扫描时钟，每个扫描周期反转一次，如果这个扫描周期为 1，那么下一个扫描周期为 0。
SM0.7	保留

2、SMB1：指令执行状态

特殊存储器 SMB1 为 PLC 各种指令的执行状态。指令执行时对相应位置位或者复位。详细的功能描述请参见下表。

位(只读)	描 述
SM1.0	保留
SM1.1	保留
SM1.2	保留
SM1.3	指令发生除零时，该位置位。适用于 DIV、MOD 指令。
SM1.4	BCD 转整型、整型转 BCD，当源数据不符合指令要求时，会置位。
SM1.5	SQRT（平方根）指令，当源数据为负数时运算错误，置位。
SM1.6	ATH（ASCII 转 16 进制数）指令，如果输入值非十六进制 ASCII 数值，该位置位。
SM1.7	保留

3、SMB2：自由口接收字符

SMB2 存储的是自由口通讯接收字符。端口 0 和端口 1 的自由口通讯接收字符都会在该字节暂存，用户可以通过程序访问。新的接收数据会覆盖之前的接收数据。

4、SMB3：自由口奇偶校验错误

SMB3 用于自由口通讯，指示自由口通讯的奇偶校验是否错误。

如果 Port0 或者 Port1 通讯过程中出现了奇偶校验错误，则该字节为 1，无错误时为 0。

5、SMB4 到 SMB7：中断队列溢出标志

特殊寄存器 SMB4 到 SMB7 为中断队列溢出标志，如果中断队列中已经存在的中断，在没有处理之前，相同的中断的再次发生，则该中断溢出。

SM	描　述
SM4.0	中断事件 1 中断溢出：1=溢出
SM4.1	中断事件 2 中断溢出：1=溢出
SM4.2	中断事件 3 中断溢出：1=溢出
SM4.3	中断事件 4 中断溢出：1=溢出
SM4.4	——
SM4.5	——
SM4.6	——
SM4.7	——
SM5.0	中断事件 9 中断溢出：1=溢出
SM5.1	中断事件 10 中断溢出：1=溢出
SM5.2	中断事件 11 中断溢出：1=溢出
SM5.3	中断事件 12 中断溢出：1=溢出
SM5.4	——
SM5.5	——
SM5.6	中断事件 15 中断溢出：1=溢出
SM5.7	中断事件 16 中断溢出：1=溢出
SM6.0	中断事件 17 中断溢出：1=溢出
SM6.1	中断事件 18 中断溢出：1=溢出
SM6.2	中断事件 19 中断溢出：1=溢出
SM6.3	中断事件 20 中断溢出：1=溢出
SM6.4	中断事件 21 中断溢出：1=溢出
SM6.5	中断事件 22 中断溢出：1=溢出
SM6.6	中断事件 23 中断溢出：1=溢出

SM6.7	中断事件 24 中断溢出：1=溢出
SM7.0	中断事件 25 中断溢出：1=溢出
SM7.1	中断事件 26 中断溢出：1=溢出
SM7.2	中断事件 27 中断溢出：1=溢出
SM7.3	中断事件 28 中断溢出：1=溢出
SM7.4	中断事件 29 中断溢出：1=溢出
SM7.5	中断事件 30 中断溢出：1=溢出
SM7.6	中断事件 31 中断溢出：1=溢出
SM7.7	中断事件 32 中断溢出：1=溢出

6、SMB10：CPU 型号标识码

特殊寄存器 SMB10 为 PLC 的 CPU 型号，具体对应关系见下表。

CPU 型号	寄存器数值(16 进制)
EC102	0x12
EC104	0x14
EC106	0x16
EC202	0x22
EC204	0x24
EC206	0x26

7、SMW12：扩展模块在线标志

上电时，PLC 对扩展模块进行诊断，获取在线的扩展模块的 ID，具体的模块型号可访问 SMB281-SMB295 寄存器。1，表示在线；0，表示不在线。

注：下表中扩展模块编号为扩展模块在系统中的物理位置，PLC 主控后第一个扩展模块为扩展模块 1，范围依次为 1---15。

SM	描　述
SM12.0	扩展模块 1 在线标志。
SM12.1	扩展模块 2 在线标志。
SM12.2	扩展模块 3 在线标志。
SM12.3	扩展模块 4 在线标志。
SM12.4	扩展模块 5 在线标志。
SM12.5	扩展模块 6 在线标志。
SM12.6	扩展模块 7 在线标志。

SM12.7	扩展模块 8 在线标志。
SM13.0	扩展模块 9 在线标志。
SM13.1	扩展模块 10 在线标志。
SM13.2	扩展模块 11 在线标志。
SM13.3	扩展模块 12 在线标志。
SM13.4	扩展模块 13 在线标志。
SM13.5	扩展模块 14 在线标志。
SM13.6	扩展模块 15 在线标志。
SM13.7	保留

8、SMW14：扩展模块组态诊断信息

PLC 运行后，如果组态扩展模块在线，则相应位为 0。如果组态扩展模块不在线或者与实际系统中扩展模块不同，则相应位为 1，与此同时 PLC 的错误灯会常亮。

注：1 表示组态错误，0 表示组态正确。

SM	**描　述**
SM14.0	扩展模块 1 组态诊断。
SM14.1	扩展模块 2 组态诊断。
SM14.2	扩展模块 3 组态诊断。
SM14.3	扩展模块 4 组态诊断。
SM14.4	扩展模块 5 组态诊断。
SM14.5	扩展模块 6 组态诊断。
SM14.6	扩展模块 7 组态诊断。
SM14.7	扩展模块 8 组态诊断。
SM15.0	扩展模块 9 组态诊断。
SM15.1	扩展模块 10 组态诊断。
SM15.2	扩展模块 11 组态诊断。
SM15.3	扩展模块 12 组态诊断。
SM15.4	扩展模块 13 组态诊断。
SM15.5	扩展模块 14 组态诊断。
SM15.6	扩展模块 15 组态诊断。
SM15.7	保留

9、SMW18 和 SMW20：定时中断的周期

SMW18 用于存储定时中断 0（中断号为 3）的周期值，其数值范围为 1~65535，单位为 ms。若将 SMW18 设置为 0，则定时中断 0 被禁止。SMW18 的缺省值为 0。

SMW20 用于存储定时中断 1（中断号为 4）的周期值，其数值范围为 1~65535，单位为 ms。若将 SMW20 设置为 0，则定时中断 1 被禁止。SMW20 的缺省值为 0。

10、SMW22 至 SMW26：PLC 扫描周期时间

如下表所示，SMW22 至 SMW26 提供 PLC 程序扫描周期时间，上次扫描周期时间、最短扫描周期时间和最长扫描周期时间，时间单位为 100 微秒。不足 100us 记为 0，超过 6s 记为 60000。

系统寄存器地址	描述
SMW22	最后一次扫描周期时间
SMW24	自 PLC 进入运行状态后，最短的扫描周期时间
SMW26	自 PLC 进入运行状态后，最长的扫描周期时间

11、SMB28 和 SMB29：顶调电位器

SMB28 和 SMB29 用于存储顶调电位器的调节值。存储过程由 CPU 运行软件自动完成，无需用户程序干预。SMB28 和 SMB29 对用户来说是只读的。

SMB28 对应顶调电位器 0 的数值，SMB29 对应顶调电位器 1 的数值。

12、SMB30 和 SM130 为自由口通讯参数字节

SMB30 为端口 0 的自由口通讯参数，SMB130 为端口 1 的自由口通讯参数。

用户可对 SMB30 和 SMB130 进行读取和写入操作。

位（只读）		**描述**
Port 0	**Port 1**	
SMB30 的格式	SMB130 的格式	自由端口模式控制字节 MSB7—LSB0：xxpp dbbb
SM30.0 到 SM30.2	SM130.0 到 SM130.2	bbb: 自由端口波特率 000 = 1200，001 = 2400，010 = 4800， 011 = 9600，100 = 19200，101 = 38400， 110 = 57600，111 = 115200
SM30.3	SM130.3	d: 数据位数 0 = 8 位(1 位停止位) ； 1 = 7 位(2 位停止位)
SM30.4 和 SM30.5	SM130.4 和 SM130.5	pp: 奇偶校验位

		00 = 无校验，01 = 偶校验，10 = 奇校验
SM30.6 和 SM30.7	SM130.6 和 SM130.7	Xx:保留

13、SMB36 到 SMB65：HSC0、HSC1 和 HSC2 的寄存器

SMB36到SMB65用于监测和控制高速计数器 HSC 0 、HSC 1和HSC 2的操作，见下表。

SM	描　述
HSC0 状态寄存器	
SM36.0 ~SM36.4	保留
SM36.5	HSC0 当前计数方向：0＝减；1＝增
SM36.6	HSC0 当前计数值是否等于预置值：0＝否；1＝是
SM36.7	HSC0 当前计数值是否大于预置值：0＝否；1＝是
HSC0 控制寄存器	
SM37.0	HSC0 复位信号的有效电平：0＝高电平有效；1＝低电平有效
SM37.1	保留
SM37.2	HSC0 正交计数器速率：0＝1x 速率；1＝4x 速率
SM37.3	HSC0 计数方向：0＝减计数；1＝增计数
SM37.4	HSC0 是否写入计数方向：0＝否；1＝是
SM37.5	HSC0 是否写入新预置值：0＝否；1＝是
SM37.6	HSC0 是否写入新当前值：0＝否；1＝是
SM37.7	HSC0 是否允许计数：0＝禁止；1＝允许
SMD38	HSC0 当前值（也就是计数的起始值）
SMD42	HSC0 预置值
HSC1 状态寄存器	
SM46.0~SM46.4	保留
SM46.5	HSC1 当前计数方向：0＝减；1＝增
SM46.6	HSC1 当前计数值是否等于预置值：0＝否；1＝是
SM46.7	HSC1 当前计数值是否大于预置值：0＝否；1＝是
HSC1 控制寄存器	
SM47.0~SM47.2	保留
SM47.3	HSC1 计数方向：0＝减计数；1＝增计数
SM47.4	HSC1 是否写入计数方向：0＝否；1＝是
SM47.5	HSC1 是否写入新预置值：0＝否；1＝是
SM47.6	HSC1 是否写入新当前值：0＝否；1＝是
SM47.7	HSC1 是否允许计数：0＝禁止；1＝允许
SMD48	HSC1 当前值（也就是计数的起始值）

SMD52	HSC1 预置值
HSC2 状态寄存器	
SM56.0~SM56.4	保留
SM56.5	HSC2 当前计数方向：0＝减；1＝增
SM56.6	HSC2 当前计数值是否等于预置值：0＝否；1＝是
SM56.7	HSC2 当前计数值是否大于预置值：0＝否；1＝是
HSC2 控制寄存器	
SM57.0	HSC2 复位信号的有效电平：0＝高电平有效；1＝低电平有效
SM57.1	保留
SM57.2	HSC2 正交计数器速率：0＝1x 速率；1＝4x 速率
SM57.3	HSC2 计数方向：0＝减计数；1＝增计数
SM57.4	HSC2 是否写入计数方向：0＝否；1＝是
SM57.5	HSC2 是否写入新预置值：0＝否；1＝是
SM57.6	HSC2 是否写入新当前值：0＝否；1＝是
SM57.7	HSC2 是否允许计数：0＝禁止；1＝允许
SMD58	HSC2 当前值（也就是计数的起始值）
SMD62	HSC2 预置值

14、SMB66 到 SMB85：PTO/PWM 的寄存器

SMB66到SMB85用于监测和控制脉冲输出（PTO）和脉宽调制（PWM）功能。见下表。

SM	**描　　述**
PTO0/PWM0 状态寄存器	
SM66.0~SM66.3	保留
SM66.4	PTO0 多段输出是否由于加减速时间设置过短而终止：0＝否；1＝是
SM66.5	PTO0 多段输出是否由于存储起始位置不为奇数地址而终止：0＝否；1＝是
SM66.6	PTO0 单段排队输出是否已满：0＝否，1＝是
SM66.7	PTO0 是否空闲：0＝忙；1＝空闲
PTO0/PWM0 控制寄存器	
SM67.0	PWM0 是否更新周期值：0=否；1=是
SM67.1	PWM0 是否更新脉宽值：0=否；1=是
SM67.2	保留
SM67.3	PTO0/PWM0　时基：0=1μs；1=1ms
SM67.4	PWM0　更新方法：0=异步更新；1=同步更新
SM67.5	PTO0 操作方式：0=单段操作；1=多段操作

SM67.6	PTO0/PWM0 功能选择：0= PTO；1=PWM
SM67.7	PTO0/PWM0 允许或禁止此功能：0=禁止；1= 允许
SMW68	PTO0/PWM0 周期值，范围2~65535
SMW70	PWM0 脉宽值，范围0~65535
SMD72	PTO0 脉冲个数，范围1~4,294,967,295
PTO1/PWM1 状态寄存器	
SM76.0~SM76.3	保留
SM76.4	PTO1 多段输出是否由于加减速时间设置过短而终止：0＝否；1＝是
SM76.5	PTO1 多段输出是否由于存储起始位置不为奇数地址而终止：0＝否；1＝是
SM76.6	PTO1 单段排队输出是否已满：0＝否，1＝是
SM76.7	PTO1 是否空闲：0＝忙；1＝空闲
PTO1/PWM1 控制寄存器	
SM77.0	PWM1 是否更新周期值：0=否；1=是
SM77.1	PWM1 是否更新脉宽值：0=否；1=是
SM77.2	保留
SM77.3	PTO1/PWM1 时基：0=1μs；1=1ms
SM77.4	PWM1 更新方法：0=异步更新；1=同步更新
SM77.5	PTO1 操作方式：0=单段操作；1=多段操作
SM77.6	PTO1/PWM1 功能选择：0= PTO；1=PWM
SM77.7	PTO1/PWM1 允许或禁止此功能：0=禁止；1= 允许
SMW78	PTO1/PWM1 周期值，范围2~65535
SMW80	PWM1 脉宽值，范围0~65535
SMD82	PTO1 脉冲个数，范围1~4,294,967,295

15、SMB86 到 SMB95、SMB186 到 SMB195：自由通讯的寄存器

15.1 SMB86 和 SMB186：自由通讯接收状态字节

位（只读）		值	含 义
Port0	**Port1**		
SM86.0	SM186.0	1	终止接收：接收到的字符存在奇偶校验错误。
SM86.1	SM186.1	1	终止接收：达到最大接收字节数。
SM86.2	SM186.2	1	终止接收：接收一个字符超时。
SM86.3	SM186.3	1	终止接收：系统接收超时。
SM86.4	SM186.4	-	保留。
SM86.5	SM186.5	1	终止接收：接收到了用户定义的结束字符。

SM86.6	SM186.6	1	终止接收：参数设置错误，无起始条件接收或者无接收结束条件等。
SM86.7	SM186.7	1	终止接收：用户使用了禁止接收命令。

15.2 SMB87 和 SMB187：自由通讯接收控制字节

位		值	描 述
Port0	**Port1**		
SM87.0	SM187.0	-	保留。
SM87.1	SM187.1	0	当发送或者接收完成时禁止生成相应的中断。
		1	当发送或者接收完成时允许生成相应的中断。
SM87.2	SM187.2	0	忽略 SMW92/SMW192 中用户定义的接收字符超时值。
		1	使用 SMW92/SMW192 中用户定义的接收字符超时值。
SM87.3	SM187.3	-	保留。
SM87.4	SM187.4	0	忽略 SMW90/SMW190 中用户定义的接收准备时间检测。
		1	使用 SMW90/SMW190 中用户定义的接收准备时间检测。
SM87.5	SM187.5	0	忽略 SMB89/SMW189 中用户定义的接收结束字符。
		1	使用 SMB89/SMW189 中用户定义的接收结束字符。
SM87.6	SM187.6	0	忽略 SMB88/SMW188 中用户定义的接收开始字符。
		1	使用 SMB88/SMW188 中用户定义的接收开始字符。
SM87.7	SM187.7	0	禁止接收数据。此禁止条件优先于其它所有的接收控制字。
		1	允许接收数据。

15.3 其它控制寄存器

控制字（字节）		描 述
Port0	**Port1**	
SMB88	SMB188	用于存放用户定义的接收起始字符。 执行 RCV 指令后，CPU 收到了起始字符就开始进入有效接收状态，之前收到的数据都会被丢弃。CPU 将起始字符作为接收的第一个有效数据并保存在接收缓冲区。 如果要使本设置值生效，需要将 SM87.6/SM187.6 置为 1。
SMB89	SMB189	用于存放用户定义的接收结束字符。 CPU 将以该字符作为接收的最后一个字节。收到该字符后，无论其它的结束条件如何 CPU 都会立刻终止接收状态。 如果要使本设置值生效，需要将 SM87.5/SM187.5 置为 1。

SMW90	SMW190	用于存放用户定义的接收准备时间（范围为 0~65535ms）。 执行 RCV 指令后，经过了该时间值，CPU 将自动进入有效接收状态而不管是否收到起始字符，此后接收到的数据将被认为是有效数据。接收准备时间为 0 表示不需要接收准备，直接接收数据。 如果要使本设置值生效，需要将 SM87.4/SM187.4 置为 1。
SMW92	SMW192	用于存放用户定义的接收字符超时值（范围为 1~65535ms）。 执行 RCV 指令并进入有效接收状态后，若在此超时时间内没有接收到任何字符，CPU 就会结束接收状态。 如果要使本设置值生效，需要将 SM87.2/SM187.2 置为 1。 若该设置为 0，则 RCV 指令直接结束执行。
SMB94	SMB194	用于存放用户定义的每次接收字符数（1~255）。 CPU 只要接收到了此数量个有效字符，无论其它的结束条件如何，都会马上终止接收状态。本设置值总是有效。 若该值设置为 0，则 RCV 指令将直接退出。

16、SMB136 到 SMB145：HSC3 的寄存器

SMB136到SMB165用于监测和控制高速计数器HSC3、HSC4和HSC5的操作，见下表。

SM	**描　述**
HSC3 状态寄存器	
SM136.0~SM136.4	保留
SM136.5	HSC3 当前计数方向：0＝减；1＝增
SM136.6	HSC3 当前计数值是否等于预置值：0＝否；1＝是
SM136.7	HSC3 当前计数值是否大于预置值：0＝否；1＝是
HSC3 控制寄存器	
SM137.0~SM137.2	保留
SM137.3	HSC3 计数方向：0＝减计数；1＝增计数
SM137.4	HSC3 是否写入计数方向：0＝否；1＝是
SM137.5	HSC3 是否写入新预置值：0＝否；1＝是
SM137.6	HSC3 是否写入新当前值：0＝否；1＝是
SM137.7	HSC3 是否允许计数：0＝禁止；1＝允许
SMD138	HSC3 当前值（也就是计数的起始值）
SMD142	HSC3 预置值

17、SMB166 到 SMB179：包络表的寄存器

SMB166~SMB178 用来显示包络步的数量和V区中包络表的地址。

SM	**描　述**

SMB166	PTO0 正在进行的多段操作段数
SMB167	保留
SMW168	PTO0 包络表的起始位置（用相对于 VB0 的字节偏移来表示）， 仅用于PTO多段操作。
SMB170~SMB175	保留
SMB176	PTO1 正在进行的多段操作段数
SMB177	保留
SMW178	PTO1 包络表的起始位置（用相对于 VB0 的字节偏移来表示），仅用于 PTO 多段操作。

18、SMB201 到 SMB242：定位控制的寄存器

SMB201~SMB242 用来控制定位控制指令的急停和方向，显示定位运动的当前脉冲值。

SM	**描　述**
Q0.0定位控制寄存器	
SM201.7	Q0.0 急停标志位。若该位为 1，则不执行任何定位控制指令。 当PSTOP（急停）指令执行时，该位将自动置1。用户需要使用程序将该位清0。
SM201.4~SM201.6	保留
SM201.3	Q0.0 方向使能控制位。 1 --- 禁止使用指定的方向输出通道； 0 --- 允许使用指定的方向输出通道。
SM201.0~SM201.2	保留
Q0.0定位控制脉冲当前值寄存器	
SMD212	脉冲个数当前值
Q0.1定位控制寄存器	
SM231.7	Q0.1 急停标志位。若该位为 1，则不执行任何定位控制指令。 当PSTOP（急停）指令执行时，该位将自动置1。用户需要使用程序将该位清0。
SM231.4~SM231.6	保留
SM231.3	Q0.1 方向使能控制位。 1 --- 禁止使用指定的方向输出通道； 0 --- 允许使用指定的方向输出通道。
SM231.0~SM231.2	保留
Q0.1定位控制脉冲当前值寄存器	
SMD242	脉冲个数当前值

19、SMB281 到 SMB295：系统中扩展模块类型寄存器

SMB281到SMB295中存储系统中扩展模块的类型标识：

SM	描　述
SMB281	扩展模块1类型
SMB282	扩展模块2类型
SMB283	扩展模块3类型
SMB284	扩展模块4类型
SMB285	扩展模块5类型
SMB286	扩展模块6类型
SMB287	扩展模块7类型
SMB288	扩展模块8类型
SMB289	扩展模块9类型
SMB290	扩展模块10类型
SMB291	扩展模块11类型
SMB292	扩展模块12类型
SMB293	扩展模块13类型
SMB294	扩展模块14类型
SMB295	扩展模块15类型

模块类型 ID 与模块的对应关系：

扩展模块	模块类型ID
EC221_08DX	0x07
EC221_16DX	0x08
EC222_08DTS/EC222_08DTD	0x09
EC222_16DTS/EC222_16DTD	0x0A
EC222_08XR	0x0B
EC222_16XR	0x0C
EC223_08DT	0x0D
EC223_08DR	0x0E
EC223_16DT	0x0F
EC223_16DR	0x10
EC231_04IV	0x11
EC231_04TC	0x12
EC231_04RD	0x13
EC232_02IV	0x14
EC233_04IV	0x15

EC231_04IVM	0x16
EC231_08IV	0x17
EC231_08IVM	0x18
EC231_04RDM	0x19
EC231_08RD	0x1A
EC231_08RDM	0x1B
EC231_04TCM	0x1C
EC231_08TC	0x1D
EC231_08TCM	0x1E
EC231_02IVM	0x1F
EC232_04IV	0x20
EC232_04IVM	0x21
EC232_08IV	0x22
EC232_08IVM	0x23
EC233_04IVM	0x24
EC233_08IV	0x25
EC233_08IVM	0x26

附录 C　数据保持

EC100/EC200 提供了数据保持功能。可组态 V 存储区、M 存储区、C 存储区、T 存储区为数据保持区。

CPU 对数据保持区做以下操作：

1） 在 PLC 掉电或者停止运行时，将组态指定的数据保持区数据内存保存到数据保持存储区。

2） 在 PLC 上电时，PLC 先将内存中的 V 存储区、M 存储区、C 存储区、T 存储区清除，然后将保存到掉电存储区的数据复制到相应内存区中。

数据保持区可设范围：

PLC 类型	EC100	EC200
掉电保持区	V 区:　1KB M 区:　32B(256 位) C 区：　C0-C255 T 区：　T0-T255	V：2KB M：64B(512 位) C 区：　C0-C255 T 区：　T0-T255

注意：数据掉电保持区在 PLC 掉电后，采用电池供电。通常上电正常使用情况下，电池可使用 3 到 5 年。如果 PLC 断电重新上电后，发现设置为掉电保持的区域数据杂乱无章。通常是 PLC 的电池没电了，可寄回公司重新更换电池。

附录 D　EC100/EC200 的故障处理

1、CPU 本体的故障处理

(1)　如何将 CPU 恢复至出厂的初始状态

EC100/EC200 提供了以下方法将 CPU 恢复至出厂的初始状态：

- **“软”清除**

EC100/EC200 支持“软”清除的方式来清除 CPU 存储器中的数据，包括用户程序、配置数据、密码等，从而将 CPU 恢复成出厂的初始状态。“软”清除的步骤如下：

①　让 CPU 的编程口使用默认的串行通讯参数。

先将目标 CPU 断电，将其运行开关拨至“STOP”位置，然后对 CPU 重新上电，此时 CPU 编程口将使用默认的通讯参数：站号为 1，波特率 9600，无校验，8 位数据位，1 位停止位。

注意：在“软”清除完成之前不要改变运行开关的位置！

②　设置计算机 COM 口的通讯参数，准备与目标 CPU 进行通讯。在**【通讯设置】**中将计算机 COM 口的通讯参数设置为目标 CPU 默认的通讯参数。具体的设置方法请参阅 2.6 与计算机的连接 。

③　执行“软”清除命令。进入 EuraProg，执行**【PLC】→【清除…】**菜单命令，即可清除 CPU 存储器中的所有内容。执行完**【清除…】**命令后，CPU 将恢复成出厂状态，用户程序、配置数据、密码等等将全部被清除，但是时钟仍然保持清除之前的设置不变。“软”清除之后，编程口的通讯参数被配置为：PLC 站号为 1，波特率 9600，无校验，数据位 8 位，停止位 1 位。

(2)　**故障现象：**上电后，CPU 的 ERR 指示灯点亮，同时程序执行异常。

可能的原因和处理方法：

- *用户工程的**【硬件配置】**中配置的 CPU 类型与实际的 CPU 类型不一致。*

  在 EuraProg 中打开用户工程，将**【硬件配置】**中配置的 CPU 类型修改为目标 PLC 的实际类型，然后将修改后的工程下载至目标 PLC 中。

- *若用户工程中用到了间接寻址，则可能是指针指向了一个非法的变量地址。*

  注意指针的有效性是由用户程序自己保证的。用户需仔细检查程序，确保在运行时指针是指向有效的变量地址，然后将正确的程序下载至目标 PLC 中。

(3)　**故障现象：**上电后，CPU 的 RUN 指示灯闪烁（有时 STOP、ERR 灯也同时闪烁）。

可能的原因和处理方法：

- *若用户工程中用到了间接寻址，则可能是指针指向了一个非法的变量地址。*

  注意指针的有效性是由用户程序自己保证的。用户需仔细检查程序，确保在运行时指针是指向有效的变量地址，然后将正确的程序下载至目标 PLC 中。下载时可以让 CPU 的编程口采用默认通讯参数，具体方法请参照上面关于“软”清除方式的描述。

- *用户使用的编程软件 EuraProg 的版本与目标 CPU 板级程序的版本不匹配。*

  由于在高版本的 EuraProg 中增加的一些功能、指令是老版本的 CPU 所不支持的，因此若使用高版本的 EuraProg 为老版本的 CPU 编程、下载，就会导致 CPU 不能正确解析用户程序。从而使 CPU 的看门狗频繁复位，出现这个故障现象。

处理方法：用户可以与欧瑞传动电气股份有限公司联系将目标 CPU 升级至更高版本然后再使用。或者用户自己将目标 CPU 恢复成出厂的初始状态，然后使用合适版本的 EuraProg 为目标 CPU 编程、下载。请参阅本节中 如何将 CPU 恢复至出厂的初始状态

敬告用户：

感谢您选用我司产品，为保证您正确使用本产品及得到我司最佳售后服务，请认真阅读下述条款，并做好相关事宜。

只有具备一定的电气知识的操作人员才能够对本产品进行接线、上电操作；手册中示例程序仅供参考，不保证其实用性。

本公司致力于产品的不断改善和升级，手册提供资料如有变更，恕不另行通知，请自行访问本公司网站获取。

产品保修范围：按使用要求正常使用情况下，所产生的故障。

产品保修期限：本公司产品的保修期为自出厂之日起，十二个月以内。保修期实行长期技术服务。

非保修范围：任何违反使用要求的认为意外、自然灾害等原因导致的损坏，以及未经许可而擅自对产品拆卸、改装及修理的行为，视为自动放弃保修服务。

从中间商处购入产品：凡从经销代理商处购买产品的用户，在产品发生故障时，请与经销商、代理商联系。

免责条款：因下列原因造成的产品故障不在厂家 12 个月免费保修服务范围之内：

（1）、厂家不依照《产品手册》中所列程序进行正确的操作；

（2）、用户未经与厂家沟通自行修理产品或擅自改造产品；

（3）、因用户环境不良导致产品器件异常老化或引发故障；

（4）、因用户超过产品的标准范围使用产品；

（5）、由于地震、火灾、风水灾害、雷击、异常电压或其他自然灾害等不可抗力的原因造成的产品损坏；

（6）、因购买后由于人为摔落及运输导致硬件损坏。

责任：无论从合同、保修期、疏忽、民事侵权行为、严格的责任、或其他任何角度讲，EURA 和他的供货商及分销商都不承担以下由于设备所造成的特殊的、间接的、继发的损失责任。其中包括但不仅仅局限于利润和收入的损失，使用供货设备和相关设备的损失，资金的花费，代用设备的花费，工具费和服务费，停机时间的花费，延误，及购买者的客户或任何第三方的损失。另外，除非用户能够提供有力的证据，否则公司及它的供货商将不对某些指控如：因使用不合格原材料、错误设计、或不规范生产所引发的问题责任。

解释权归欧瑞传动电气股份有限公司。

如果您对 EURA 的产品还有疑问，请与 EURA 公司或其办事处联系。技术数据、信息、规范均为出

版时的最新资料，EURA 公司保留部事先通知而更改的权利，并对由此造成的损失不承担任何责任。

解释权归 EURA 公司。

欧瑞传动电气股份有限公司
EURA DRIVES ELECTRIC CO.,LTD
24小时服务热线： 4006-866-333
公司网址： www.euradrives.com