前　言

上海步科自动化股份有限公司是致力于国产高品质 PLC 开发、营销、生产、服务的高科技企业。公司的创业者凭借在自动化领域十几年的实践经验和对 PLC 产品的深刻理解，利用我们所掌握的 PLC 软/硬件核心技术，开发与国际同步的高品质 PLC，以 100%自有知识产权、优良的品质、体贴到位的服务，创建适合国内用户的小型一体化 PLC 民族品牌，打破进口品牌垄断中国市场的局面，为民族自动化产业贡献一份力量。

自 2011 年以来，上海步科自动化股份有限公司推出了 Kinco-K 系列小型 PLC，包括 K5、K2 等产品。其中 K5 属于标准型产品，具有功能丰富、高性能、高可靠性、扩展性良好等特点。K2 属于经济单机型产品，在保证功能、性能、可靠性的前提下，优化硬件设计以降低成本，更提供了 USB 编程口、晶体管型 DIO 点（DI、DO 复用）等更贴近用户需求的功能，具有很高的性价比。自推向市场以来，K 系列 PLC 已经应用到环保机械、木工机械、纺织机械、建材机械、食品机械、中央空调以及小规模过程控制等领域，以其优良的性能价格比得到用户的一致认可。

为了方便用户的使用，我们编写了此手册，对 K 系列 PLC 的编程软件 KincoBuilder 进行了系统介绍。本手册内容详尽，详细描述了编程的基本概念、KincoBuilder 软件的界面功能、操作方法和指令集，其中穿插了大量的示例。另外，本手册对一些 IEC61131-3 的基本概念也进行了介绍，以照顾广大初学 PLC 的用户。

由于时间仓促，书中难免有疏漏不足之处，敬请广大用户指正，我们将不胜感激。

公司地址：深圳市南山区高新科技园北区朗山一路 6 号意中利工业园 1 号厂房 1-3 层
邮　　编：518057
电　　话：0755-26585555
传　　真：0755-26616372
电子邮箱：sales@kinco.cn
网　　址：www.kinco.cn

目　录

第一章 欢迎使用 KincoBuilder

1.1 KincoBuilder 简介

KincoBuilder 是步科公司 K 系列 PLC 的上位编程软件，编程环境符合 IEC61131-3 标准，是一套功能强大、使用方便、高效的开发系统。

IEC61131-3 是 IEC 为工业自动化编程制定的标准，是在吸收不同厂家编程语言风格、方言及适应未来软件技术发展要求制定的，独立于任何一家公司，适合不同领域、不同类编程人员习惯。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。
KincoBuilder 完全是自主研发，采用了符合 IEC61131-3 标准的方案，由于许多用户通过各种渠道已经掌握了大部分编程技巧，这就使得 KincoBuilder 简便、易学，使用起来将会得心应手。

KincoBuilder 软件具有如下特点：

- 符合 IEC61131-3 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 IEC61131-3 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、在线监视、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 本书常用名词术语

- 小型一体化可编程控制器

按国际通用分类原则，小型 PLC 一般指实际控制点数小于 128 点（并非设计控制点数）的 PLC，此类 PLC 通常采用一体化结构，即在 CPU 模块上集成一定数量的 I/O 以及输出电源、高速输入/输出等其他附件。

- CPU 本体

即 CPU 模块，是控制系统的核心。用户编写的应用程序经编程软件下载后存储于 CPU 模块中的永久性存储器中，在 CPU 运行软件的调度下执行用户程序的控制功能，运行软件同时还负责诊断各模块的工作状态和用户程序的执行错误。

- 扩展模块与扩展总线

扩展模块是用于扩展 CPU 本体功能的模块，分为扩展 I/O 模块（增加系统的输入/输出通道）和扩展功能模块（扩展 CPU 功能）。

扩展总线连接 CPU 模块和扩展模块的数据和信号通道，物理介质采用了 16 芯扁平电缆。在扩展总线中集成有数据总线、地址总线和扩展模块工作电源。

- KincoBuilder

Kinco-K 系列 PLC 的编程软件，符合 IEC61131-3 标准，目前支持 LD 和 IL 两种标准语言。利用 KincoBuilder 可以对 Kinco-K 系列 PLC 实现编程、调试等各种功能。

- CPU 运行软件

又称 CPU 板级固件（firmware），存储于 CPU 本体的 Flash 存储器中，在 CPU 模块上电同时开始运行，管理、调度 CPU 的所有任务。用户编写的应用程序是在 CPU 运行软件的调度之下执行的。运行软件同时还负责诊断各模块的工作状态和用户程序的执行错误。

- 应用程序

又称用户程序、用户工程，是用户编写的完成特定控制功能的程序。应用程序下载后存储于 CPU 模块的永久性存储器中，CPU 上电运行时被读入 RAM 中执行。

- 主程序与程序扫描

在 CPU 中各种任务是被循环地连续执行的，这种循环执行任务的过程称为扫描。

主程序是应用程序的执行入口。在每个扫描周期内主程序都会被执行一次。应用程序中只能有唯一的主程序，但可以从主程序中调用多个子程序。

- I/O 映像区

物理 I/O 点的状态数据在 CPU 中的存储区域，分为输入映像区和输出映像区。

为保证在 CPU 一次扫描中数据的一致性及提高程序执行速度，在每次扫描中 CPU 首先将物理输入点的状态读入到输入映像区，应用程序执行过程中只对 I/O 映像区进行访问，在扫描结束时再将输出映像区中的数据发送到物理输出通道。本文中提到的 I/O 地址，如 I0.0、Q1.0、AIW0、AQW0 等，均是 I/O 映像区中的地址。

- 数据保持与数据备份

数据保持是指在 CPU 断电后 RAM 中的数据保持为断电瞬间的状态并供 CPU 在下一次上电的时候使用。数据保持靠一次性电池，在常温下，保持时间约为 3 年，用户可以自行更换电池。

数据备份是指用户通过指令将数据写往永久存储器中进行保存。注意：永久性存储器均有使用寿命，E^2PROM 允许写 100 万次，FRAM 允许写 100 亿次。

第二章　使用 KincoBuilder 快速入门

本章对 KincoBuilder 的安装以及运行环境进行了详细的描述。另外，也描述了如何连接 PLC 与计算机。最后，举例说明了开发一个工程的常见步骤。这一章的目的只是帮助用户快速入门，关于 KincoBuilder 各种功能的详细描述请参见后续章节。

2.1 系统需求

2.1.1 硬件需求

- CPU 主频 1GHz 或更高， 硬盘 20M 以上空间，内存 512M 以上
- 键盘、鼠标、串行通信口（com 口）或者 USB 口（某些系列需另外使用 USB/RS232 转换器）
- 256 色或更高，分辨率 1024*768 或更高

2.1.2 软件需求

Windows XP(32bit), Windows Vista(32bit), Windows7(32/64bit), Windows8 (32/64bit), Windows 8.1(32/64bit)。

有些用户在 Windows7 或更高版本的操作系统下运行 Kincobuilder 可能会遇到问题，下面列出了常见的问题和解决办法。

- 【PC 机通信设置】对话框中的【COM 口】列表为空

KincoBuilder 通过读取注册表中的硬件信息来获取本机可用的 COM 口，在早期的版本中， 必须赋予 KincoBuilder 以管理员身份来运行的权限，否则就会显示一个空的串口列表。

`最新的版本中，增加了一个判断分支，当没有权限读取到 windows 的串口列表时，直接列出了 COM1-COM9 的列表，用户可以自己去看 windows 设备管理器中自己的计算机的串口号，并在 kincobuilder 中直接选择使用。

- 在某些计算机上无法运行 KincoBuilder

请尝试设置以兼容 Windows XP 的模式来运行 KincoBuilder。设置方法如下：

1) 在桌面或【开始】菜单中的“KincoBuilder V1.5.x.x”快捷方式中单击右键，然后在弹出菜单中执行【属性】命令。
2) 在【属性】对话框中，进入【兼容性】页面进行设置，如图 2-1。

图 2-1　Windows“兼容性”设置

图 2-2 “以同步方式打开串口”

- 使用某些 USB 转 RS232 转换器时与 PLC 通信经常失败

这是由该转换器驱动程序的兼容性引起的。目前发现的几个失败事例以 64 位版本的 Win7 版本居多。

在 Kincobuilder 中，执行【工具】→【**软件设置…**】菜单命令，打开“软件设置”对话框，选中“以同步方式打开串口”，然后单击“确定”按钮退出即可。如图 2-2。

设置完成后，Kincobuilder 将以同步方式操作串口，在大多数情况下能够解决这个问题。

2.2 KincoBuilder 界面总体介绍

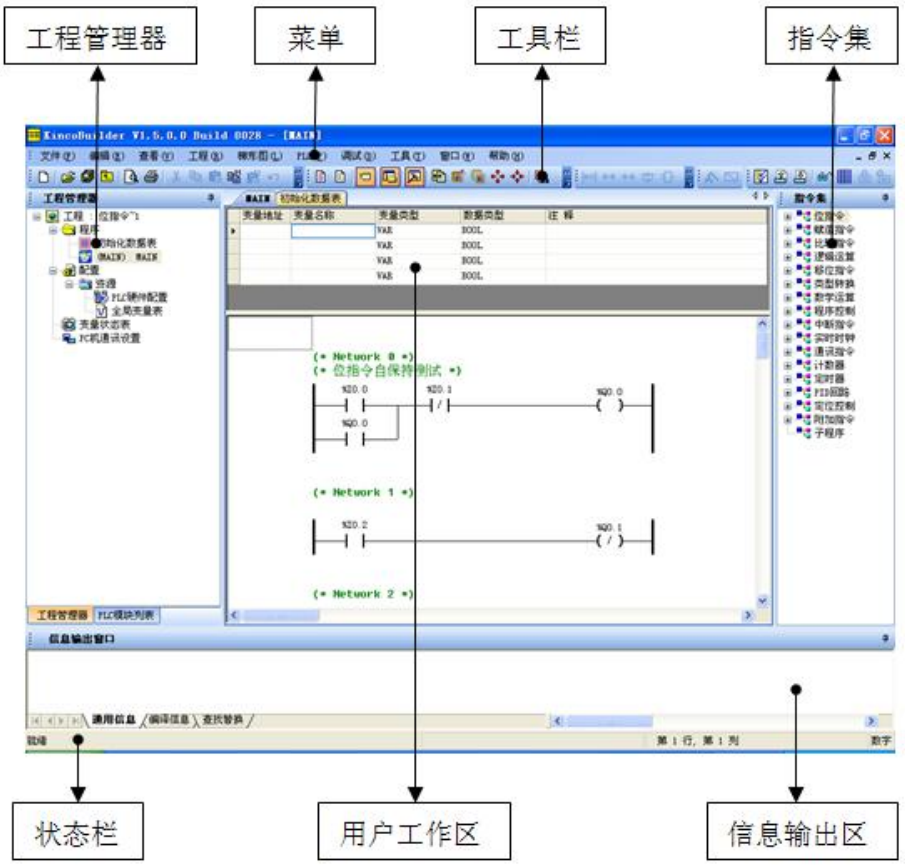


图 2-3 KincoBuilder 界面

- 菜　　单：菜单中包含了 KincoBuilder 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、变量状态表、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来。
- 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。图 2-7 中显示的是编辑器，

包括区域上部的变量定义表格和区域下部的程序编辑器（又分为 LD 编辑器、IL 编辑器）。

- 指　令　集：以树状列出了 Kinco-K 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 KincoBuilder 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。

2.3 KincoBuilder 中应用程序的组织

2.3.1 工程的组织结构

在 KincoBuilder 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。

程序	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。
	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。
	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。
配置	资源	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。
		全局变量表（可选）	用户在此声明工程中需要的全局变量。

表 2-1 工程的组织结构

2.3.2 工程的存储目录

在建立工程时 KincoBuilder 软件将会要求用户输入工程的存放路径，然后就会生成一个空工程文件（扩展名为.kpr）并存放于此路径下。另外，在此路径下将会自动建立一个与工程名称相同的子目录，用于存放该工程所有的程序文件、变量文件以及其它的一些临时文件等。

例如，用户若选择在 c:\temp 目录下建立一个名为 project 的工程，则工程文件的路径是 c:\temp\project.kpr，其它文件存放在 c:\temp\project 目录中。

2.3.3 导入工程和导出工程

KincoBuilder 在【文件】菜单中提供了【导入工程…】和【导出工程…】命令以便于用户对工程进行备份以及管理。

- 【导出工程…】

将当前打开的工程所涉及到的所有文件压缩至一个文件（扩展名.zip）中。在压缩文件中，所有的文件名、相对路径以及工程的相关设置均保持不变。例如，工程名称为 myproj1，工程文件为 myproj1.kpr，则压缩文件中的工程文件仍旧为 myproj1.kpr，工程名称仍旧为 myproj1。

执行【导出工程…】后弹出“导出工程…”对话框，如下图所示。选择好路径并输入备份文件名后，单击【保存】按钮即可。

图 2-4　导出工程

- 【导入工程…】

导入一个已存在的工程备份文件（扩展名.zip）并将其打开。

执行该命令后，首先将弹出“导入工程…”对话框，如下图。

图 2-5　导入工程：选择文件

选择好备份文件后并单击【打开】按钮，将弹出如下对话框，选择解压之后工程文件的存放目录。选择好路径后，单击【确定】按钮即可将工程文件解压到选定的目录下并将其打开。

图 2-6　导入工程：选择目录

2.4　CPU 执行用户程序

CPU 循环地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

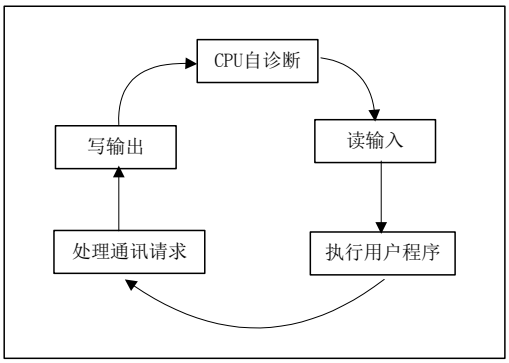


图 2-7　扫描周期

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。

Kinco-K 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。

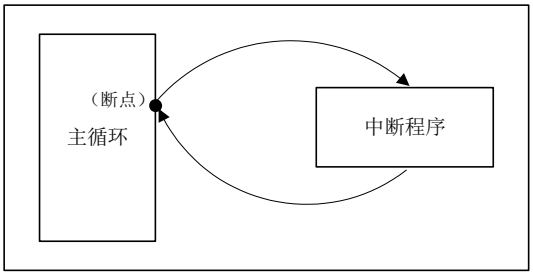


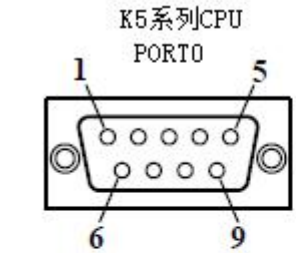
图 2-8　中断服务程序的执行

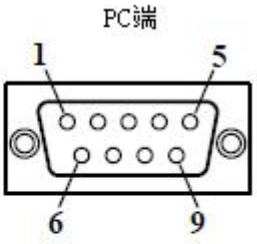
2.5　如何连接计算机与 PLC

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。此处介绍的是在如何开始建立计算机与 CPU 之间的通讯。

1)　使用适当的编程电缆连接好计算机和 CPU。

A:K5 系列 PLC 使用如下编程电缆连接计算机的串行通讯口和 CPU 的 PORT0 口(RS232)（PC 端无串行通讯口的需另外使用 USB/RS232 转换器）





孔号	定义	描述
5	GND	信号地
3	TxD	发送数据
2	RxD	接收数据

孔号	定义	描述
5	GND	信号地
3	TxD	发送数据
2	RxD	接收数据

⚠ 由于 RS232 的电气标准不支持带电插拔，因此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。

B:K2 系列 PLC 直接使用 Micro USB 编程电缆（常用作于安卓手机数据线）连接计算机的 USB 口和 CPU 的 USB 口（USB2.0）

在电脑上，K2 编程口作为一个虚拟串口，首次使用时必须先安装驱动程序。当安装完成最新版本的 Kincobuilder 编程软件后，驱动程序存放在 Kincobuilder 编程软件安装目录下面的 \Drivers 目录下面，目前支持 Windows XP、Windows 7 和 Windows 8 系统。如下图示例：

当第一次使用编程数据线连接 K2 和电脑时，Windows 系统会自动检测到新硬件并提示安装驱动程序，此时用户根据自己的 Windows 版本选择相应目录下的驱动程序即可。

- **在 Windows 7 系统下，无法安装驱动程序怎么办？**

这是因为使用的是精简过的 Ghost 的 Win7 系统，系统缺少一些文件，从而导致了无法安装虚拟串口。此时可有两种办法解决

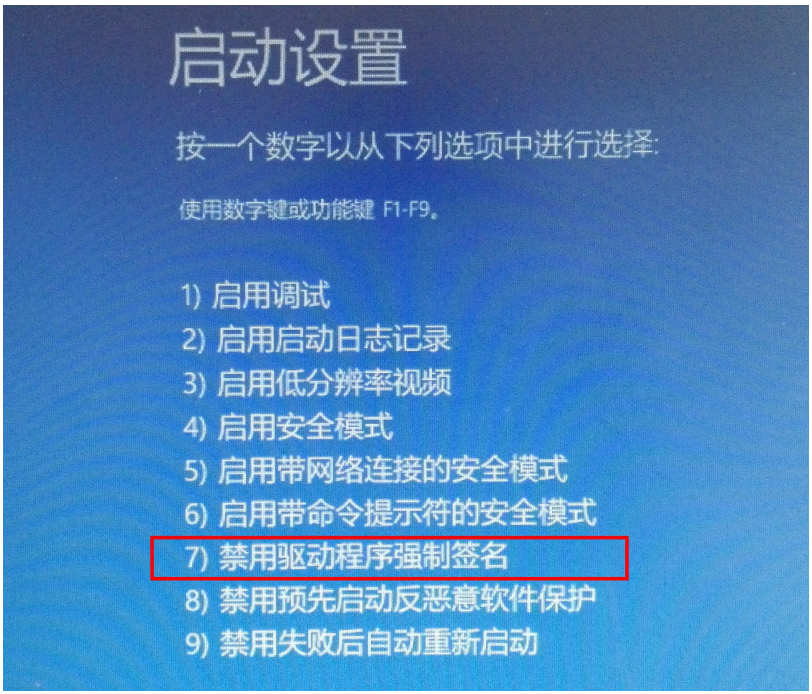
1：直接联网搜索驱动程序，该操作根据网络带宽会有一定的耗时

2：在 默认安装目录 C:\Program Files\Kinco\KincoBuilder V1.8.0.0\Drivers\Win7 下，存放着一个补丁包，用户可在该目录下根据文档说明安装该补丁包

- **在 Windows 8 系统下，如何安装驱动程序？**

Windows 8 及以上版本要求第三方的 inf 文件中必须包含经过 WHQL（Windows 实验室）认证的数字签名信息，否则就禁止继续安装。

解决办法是:执行【高级启动】中的【立即启动】命令，然后在后续的启动设置界面中选择【禁用驱动程序强制签名】并重启电脑，如下图。这样就能暂时禁用 Win8 的驱动程序强制签名检查功能，随后可以继续安装驱动程序，并在弹出的安全提示对话框中单击选择【始终安装此驱动程序软件】即可。



2) 启动 KincoBuilder，新建一个工程或者打开一个已经存在的工程。

3) 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：

   a) 在工程管理器中，双击**【PC 机通讯设置】**节点，或者在**【PC 机通讯设置】**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】→【PC 机通讯设置…】**菜单命令，均可进入“PC 机通讯设置”对话框，如图 2-13。

图 2-9 “PC 机通讯设置”对话框

b) 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，KincoBuilder 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 KincoBuilder 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。

若用户使用的是 USB 转 RS232 的转换器，那么在安装好转换器驱动程序第一次使用时，如果不能自动检测到串口参数，用户就需要重新启动计算机以便新驱动程序生效。

- 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！

4) 现在设置好了计算机串口的通讯参数，可以接着对所连 PLC 进行编程、调试了。

2.6 如何修改 CPU 的串行通讯参数？

用户连接好计算机和 CPU 之后就可以使用 KincoBuilder 来检查或者修改该 CPU 的通讯参数了。

1) 首先，使用如下任一方法进入硬件配置窗口：

- 双击工程管理器中【**资源**】组下的【**PLC 硬件配置**】节点；
- 在【**PLC 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。

在硬件配置窗口的上半部分，以表格形式列出了工程中用到的 PLC 模块，我们称之为模块列表。PLC 模块列表表达了一个真实的 PLC 控制系统：各个模块在表格中的排列次序应当与实际模块在扩展总线上的连接次序一致。

在硬件配置窗口的下半部分，是 PLC 模块列表中选中模块的所有配置参数，我们称之为模块参数配置窗口。

2) 鼠标单击并选中模块列表中的 CPU 模块，然后在下部的模块参数配置窗口中选择【通讯设置】页面，如下图，用户可以在此对其通讯参数进行设置。

图 2-10 串行口通讯参数配置页面

3) 通讯参数配置完成后，必须将当前工程下载至 CPU 中，然后新的通讯参数才会生效。

2.7 举例：建立一个工程的常见步骤

假定用户已准备好所需的硬件。

为了帮助用户迅速了解和掌握 Kinco-K5，下面我们将一步步地举例说明如何创建、调试一个具体的工程。请注意这些只是常见步骤的描述，若用户需要详细地了解某一方面的具体功能，请参阅后续章节的相关部分。

假定我们要编写如下工程：

- 工程名称：project
- 硬件：一个 Kinco-K506-24AT CPU 模块

- 实现功能：依次启动 Q0.0 --- Q0.7，并且循环执行这个过程。

为了保证良好的程序结构，我们在工程中使用了两个 POU：一个名称为“Demo”的子程序，目的是实现要求的控制功能；一个名称为“Main”的主程序，这是 CPU 的主循环任务，在主程序中将调用“Demo”子程序。

1) 启动 KincoBuilder。

2) 若需要的话，使用如下方式来设置 KincoBuilder 软件中用到的一些默认值。

执行**【工具】→【软件设置…】**菜单命令，进入“软件设置”对话框。在这个对话框中，用户可以对一些软件默认值进行设置，比如**【默认编程语言】**、**【默认 CPU 型号】**等。单击**【确认】**按钮后，设置的默认值将生效并由 KincoBuilder 自动保存，不需要每次都进行设置。

在此我们设置**【默认编程语言】**是“梯形图 LD”。

3) 使用如下任一方法建立一个新工程：

- 执行**【文件】->【新建工程..】**菜单命令；
- 单击工具栏上的图标。

然后就会出现“新建工程”对话框。用户在此为新工程命名并选择存放路径。路径、工程名称确定以后，单击**【保存】**按钮，创建新工程。本例中我们选择 D:\temp 作为工程存放目录，工程名称为“project”。

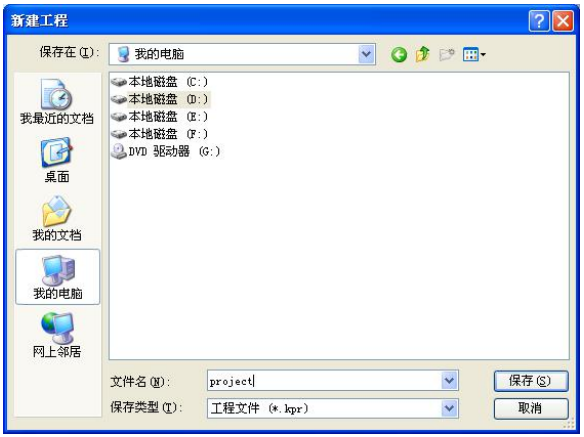


图 2-11 “新建工程”对话框

4) 进行 PLC 硬件配置

当用户新建一个工程时，KincoBuilder 将自动加入一个用户在“软件设置”对话框中指定的**【默认 CPU 型号】**的 CPU 模块。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。

请参阅 2.6 如何修改 CPU 的串行通讯参数 来了解进行硬件配置的具体步骤。

5) 关于初始化数据表

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。

注意：“初始化数据表”、硬件配置的“数据保持”以及用户程序中使用指令永久保存的内存区域均会在上电之后进入主循环之前进行恢复或者赋值，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、恢复用户使用指令永久保存的数据。

6) 根据实际的功能需求来编写程序

用户编程可以使用两种语言：IL 或者 LD。用户可以随时执行**【工程】→【IL 语言（指令表）】**或者**【工程】→【LD 语言（梯形图）】**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。

下面我们将针对本示例来编写一个主程序“Main”和一个子程序“Demo”。

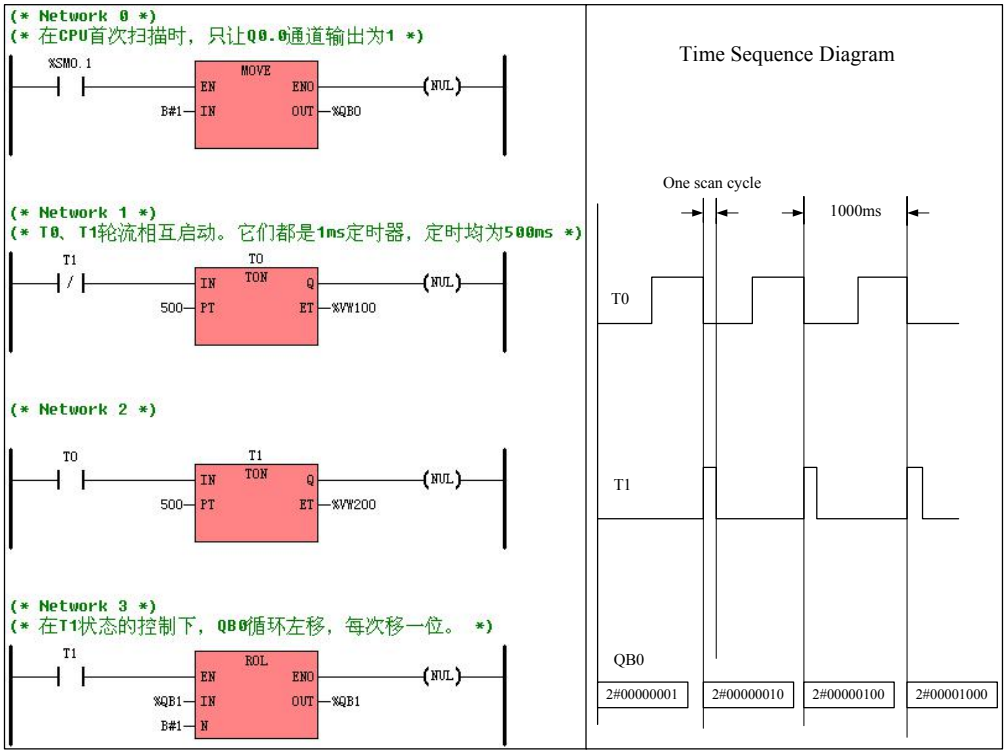
① 主程序

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”。

② 由于在主程序中只需要调用子程序“Demo”，但该子程序尚未建立，所以现在我们先不管主程序了，继续建立子程序“Demo”。

- 先使用如下任一方法建立一个新的子程序：
- 执行**【工程】→【新建子程序】**菜单命令；
- 单击工具栏上的  图标；
- 在工程管理器中的**【程序】**组节点上单击右键，执行弹出菜单中的**【新建子程序】**命令。

然后就会建立起一个新的子程序，其名字默认是“SBR_0”。现在可以先输入如下图所示的程序，图中右侧部分是时序图。

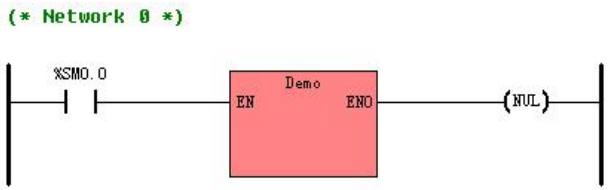


程序写完后，关闭该子程序，然后在工程管理器【**程序**】组中的【（SBR00）SBR_0】节点上单击鼠标右键将会弹出菜单，执行【**重命名**】命令将名字改为“Demo”，或者执行【**属性**...】命令，在该程序的“属性”对话框中进行修改也可以。

③ 重新修改主程序

现在我们编完了子程序“Demo”，需要重新返回到主程序中加入对它的调用指令。

切换到主程序的编辑窗口，输入以下程序：



7） 编译工程

编写完工程后，就可以对它进行编译了。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，以免有意外发生。

用户可以使用如下任一方法来启动编译过程：

- 执行【PLC】→【**编译当前工程**】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部信息输出窗口中的“编译信息”页面中。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。

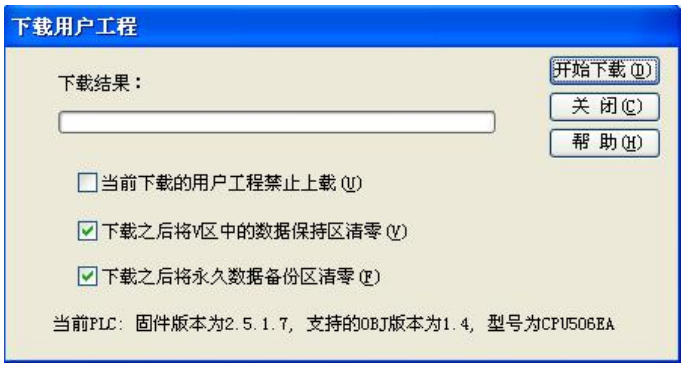
8) 下载工程

编译成功后，就可以将当前工程下载至 PLC 中了。

补充一下：若需要指定 PC 机串口的通讯参数，请参阅 2.5 如何连接计算机与 PLC

- 用户可以使用如下任一方法来打开“下载用户工程”对话框：
- 执行【PLC】→【**下载**..】菜单命令；
- 单击工具栏上的图标；
- 使用快捷键 **F8**。

➢ “下载用户工程”对话框：



✧ 【当前下载的用户工程禁止上载】

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。

✧ 【下载之后将 V 区中的数据保持区清零】

若选中此项，则在本次下载之后，数据保持区域 V、C 区中的数据将全部清零。

若不选中此项，则在本次下载之后，数据保持区域 V、C 区中的数据将保持不变。

✧ 【下载之后将永久数据备份区清零】

若选中此项，则在本次下载之后，永久数据备份区中的数据将全部清零。

若不选中此项，则在本次下载之后，永久数据备份区中的数据将保持不变。

在“下载用户工程”对话框中设置完成后，单击**【开始下载】**按钮即可将工程下载至 PLC 中。现在可以将 CPU 的运行开关放至“RUN”位置，看看程序的运行情况吧。

9) 程序调试

用户可以使用 KincoBuilder 提供的在线监视、强制等功能对程序进行调试。

➢ 在线监视

在线监视有两种模式：

- 在变量状态表中进行监视。用户可以在表中输入想要监视的变量进行监视。
- 在程序中进行监视。用户可以观察到当前程序的运行情况。在线监视时不允许修改程序。

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

用户可以使用如下任一钟方法进入在线监视状态：

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

➢ 强制

用户可以使用强制功能来强制修改 PLC 中 I、Q、M、V、AI 或者 AQ 区内变量的值，其中 I、Q、M 和 V 区中的变量可以进行位、字节、字或者双字方式强制，AI、AQ 区中的变量可以进行字强制。

KincoBuilder 允许同时强制最多 32 个变量。立即指令不允许强制。

用户可以通过如下任一种方式来执行强制功能：

- 在变量状态表中进行强制。用户可以在表中输入所要监测的变量及其强制值，然后执行菜单命令（或者右键菜单命令）**【强制】**、**【全部强制】**等实现相应的功能。
- 进入在线监视状态之后,直接在程序中对用到的变量进行强制。

  开关量：在触点、线圈上单击右键，执行弹出菜单中的**【强制为 0】**、**【强制为 1】**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。

在同一时刻，一个变量或许会存在如下取值的可能：外部的输入信号（I、AI）或者用户程序中指令执行的结果（Q、AQ、M、V），用户指定的强制值。因此规定了如下强制值生效的原则：

✓ 对于 M 区、V 区中的变量，强制值与指令执行结果处于同一优先级：用户在强制时，强制值将会生效；但强制值在一个扫描周期中只会生效一次，之后指令执行结果将会生效。

- ✓ 对于 I 区、AI 区中的变量，强制值优先于外部信号输入值。若用户指定了强制值，则强制值将会优先在用户程序中生效。
- ✓ 对于 Q 区、AQ 区中的变量，在程序执行过程中以指令执行结果优先，但在扫描周期最后的输出任务中将会以强制值优先。

用户可以执行**【取消强制】**菜单命令来取消对某个变量的强制状态，或者执行**【全部取消强制】**命令来取消全部变量的强制状态。

当 CPU 冷启动（重新上电）后，所有变量的强制状态都会被取消。

上面以摘要的形式描述了用户创建一个新工程以及进行调试等常用的步骤，为初次使用的用户提供了一个指南。关于各个部分更详细的描述请参见后续的相关章节。

第三章　PLC 编程基础

本章详细描述了对 Kinco K 系列 PLC 编程的基础知识，同时也介绍了 IEC61131-3 标准中的一些基本概念，这些概念对于用户使用任何一种 IEC61131-3 软件都是非常有用的。本章的目的是帮助用户开始编程的学习和实践，达到“知其然并知其所以然”的程度。

在初次阅读时，并不需要用户对每个环节都理解得非常透彻，但建议用户采取“边阅读，边实验”的方式，这样会有助于对本章内容的理解。同时建议用户阅读后续章节时也采用这种方式。

3.1　程序组织单元（POU，Programme Organization Unit）

IEC61131-3 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，IEC61131-3 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 IEC61131-3 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。

3.2 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 IEC61131-3 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 K 系列 PLC 支持的基本数据类型如下表所示。

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； $1.18*10^{-38}$ ~ $3.40*10^{38}$， 0， $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	0.0

表 3-1 Kinco-K 系列支持的基本数据类型

PLC 中的实数类型遵循 ANSI/IEEE754-1985 标准，也就是 C 语言中的 float（单精度）类型。

- **关于 REAL 型数据的舍入误差**

实数的二进制表示是不可能很精确的。因为 REAL 数据在存储时占用 4 个字节的空间，能够表示的有效数字最多是 6-7 位，在有效位之外的数字将被舍入，因此会产生舍入误差。

所谓的有效数字，是从一个数据最左边第一个不是 0 的数字起，到最后一位数字止，这中间所有的数字都叫做这个数的有效数字。比如，3.3、0.33 的有效数字位数都是 2 位，3、30、0.330 的有效数字都是 3 位。

根据上面的描述，我们举个例子，1.23456 在 K 系列中是能够精确表示的，而 1.2345678 就不能精确表示。

- **关于实数“0.0”**

由于存在舍入误差，“0.0”在 PLC 中可能不会精确表示出来。因此，我们根据实数能够表示的最大有效数字位数，做如下规定：凡是位于[-0.000001, 0.000001]范围内的实数，PLC 都会作为“0.0”进行处理。

- **关于实数的比较**

当使用比较指令（GT、GE、EQ、NE、LT、LE）时，我们需要注意：由于存在舍入误差，因此两个实数不可能进行精确的比较。根据前面对“0.0”的描述，当两个实数的绝对值的差值位于[-0.000001, 0.000001]之内，PLC 就认为这两个实数是相等的，否则才是不相等的。

3.3 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。

3.3.1 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 KincoBuilder 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 IEC61131-3 明确规定。

3.3.2 标识符的使用

下面列出了在 KincoBuilder 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

3.4 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 Kinco-K 系列支持的各种类型常量的定义及示例。

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		W8#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	$1.18*10^{-38}$ ~ $3.40*10^{38}$， 0， $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	1.0，-243.456
	指数形式：xEy x：带小数点的十进制数字 y：整数。		-2.3E-23

表 3-2 常量的定义

提示：

(1) 在 IEC61131-3 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#16#7FFF＝32767，I#16#8000＝-32768 等。

3.5 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 IEC61131-3 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。

3.5.1 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 3.3.1 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 IEC61131-3 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 Kinco-K 系列支持的标准变量类型。

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。

表 3-3 Kinco-K 系列支持的变量类型

3.5.2 在 KincoBuilder 中声明变量

在 KincoBuilder 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。

3.5.3 变量的检验

在编辑、编译程序时，KincoBuilder 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 IEC61131-3 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，KincoBuilder 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。

3.6 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 Kinco-K 系列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。

3.6.1 内存区域类型及其特性

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
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，不允许掉电保持
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

表 3-4　Kinco-K5 的内存区域分配

3.6.2 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。

3.6.2.1 直接地址表示格式

IEC61131-3 规定，所有的直接地址都必须以“%”开始。用户在编程时输入直接地址的时候，可以不输入“%”，Kincobuilder 可以自动加上，比如，若用户输入“I0.0”，那么 Kincobuilder 可以自动转换为“%I0.0”。

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	%IBx
	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	%IWx
	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	%IDx
	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	%QBx
	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	%QWx
	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	%QDx
	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	%MBx
	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	%MWx
	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	%MDx
	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	%VBx
	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	%VWx
	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	%SMBx
	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	%SMWx
	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	%SMDx
	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	%LBx
	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	%LWx
	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	%LDx
	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	%AIWx
	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	%AQWx
	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**

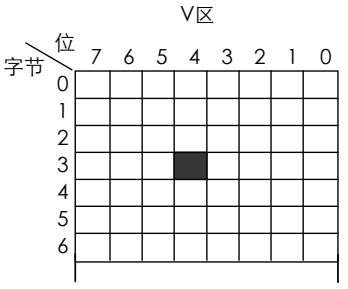
字寻址	格式	%HCx
	描述	x：高速计数器的编号。
	数据类型	DINT
	示例	%HC0　%HC1

3.6.2.2 直接地址与内存单元之间的映射

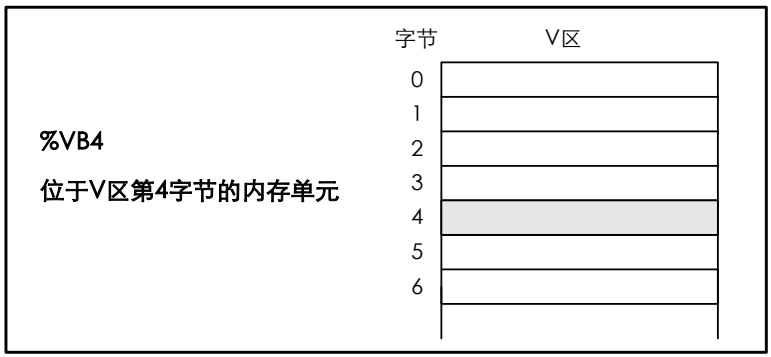
每一个合法的直接地址都对应于 CPU 中的一个内存单元。在程序中对直接地址的操作就是对其对应的内存单元进行操作。下面将以 V 区为例图解直接地址与内存单元之间的映射关系。

- 位地址：

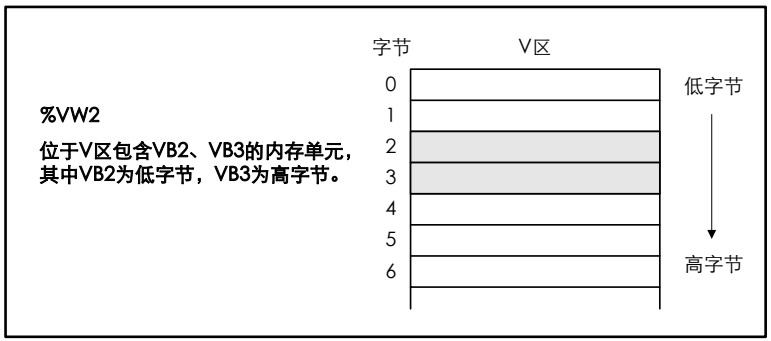
%V3.4
位于V区第3字节的第4位的内存单元



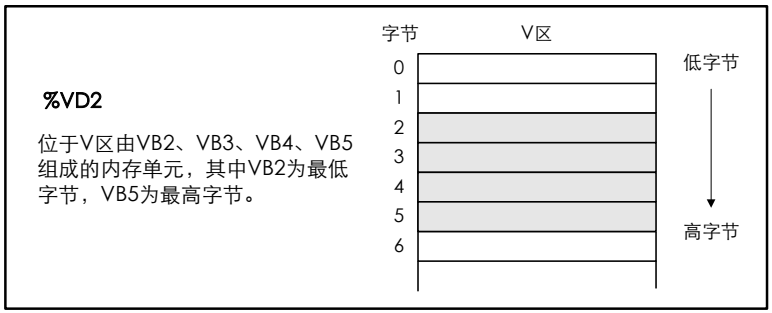
• 字节地址：



• 字地址：



• 双字地址：



3.6.3 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 Kinco-K 系列中，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。

3.6.3.1 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 *)
```

3.6.3.2 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    &VB0, %VD200      (* 建立指针 VD200，其中存放着 VB0 的物理地址 *)
MOVE    *VD200, %VB10     (* 该指令将 VB0 的值赋给 VB10。因为指针 VD200 指向 *)
                          (* 直接地址变量 VB0，所以*VD200 代表了 VB0。*)
```

3.6.3.3 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。

3.6.3.4 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。

Kinco-K 系列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    &VB4, *VD44
```

3.6.3.5 间接寻址示例

```
(* Network 0 *)
LD          %SM0.0
```

```
MOVE    &VW0, %VD200    (* 建立指针 VD200，其中存放着 VW0 的物理地址 *)
MOVE    *VD200, %VW50    (* 该指令将 VW0 的值赋给 VW50。因为指针 VD200 指向 *)
                         (* 直接地址变量 VW0，所以*VD200 代表了 VW0。*)
ADD      DI#2,  %VD200    (* 指针 VD200 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 VD200 指向了 VW2 *)
MOVE    *VD200, %VW52    (* 该指令将 VW2 的值赋给 VW52 *)
```

3.6.4 内存区域的地址范围

Kinco-K 系列有几种不同规格的 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。

		CPU504	CPU504EX	CPU506 、 CPU506EA 、 CPU508
I	长度（字节）	1	5	32
	位地址	%I0.0 --- %I0.7	%I0.0 --- %I4.7	%I0.0 --- %I31.7
	字节地址	%IB0、IB1	%IB0 --- %IB4	%IB0 --- %IB31
	字地址	%IW0	%IW0 --- %IW2	%IW0 ---% IW30
	双字地址	———	%ID0	%ID0 --- %ID28
Q	长度（字节）	1	5	32
	位地址	%Q0.0 --- %Q0.7	%Q0.0 --- %Q4.7	%Q31.0 --- %Q31.7
	字节地址	%QB0	%QB0 --- %QB4	%QB0 --- %QB31
	字地址	———	%QW0 --- %QW2	%QW0 --- %QW30
	双字地址	———	%QD0	%QD0 --- %QD28
AI	长度（字节）	0	16	64
	字地址	———	%AIW0 --- %AIW14	%AIW0 --- %AIW62
AQ	长度（字节）	0	16	64
	字地址	———	%AQW0 -- %AQW14	%AQW0 -- %AQW62
HC	长度（字节）	8		
	双字地址	%HC0，%HC1		
V	长度（字节）	4096		
	位地址	%V0.0 ---%V4095.7		
	字节地址	%VB0 --- %VB4095		
	字地址	%VW0 --- %VW4094		
	双字地址	%VD0 --- %VD4092 %VR0 --- %VR4092		
M	长度（字节）	1024		

	位地址	%M0.0 --- %M1023.7
	字节地址	%MB0 --- %MB1023
	字地址	%MW0 --- %MW1022
	双字地址	%MD0 --- %MD1020
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

表 3-5 Kinco-K5 内存区域的范围

		CPU205
I	长度（字节）	2
	位地址	%I0.0 --- %I0.5（%I0.0 --- %I1.1）
	字节地址	%IB0、IB1
	字地址	%IW0
	双字地址	———
Q	长度（字节）	2
	位地址	%Q0.0 --- %Q0.5（%Q0.0 --- %Q1.1）
	字节地址	%QB0、 QB1
	字地址	%QW0
	双字地址	———
AI	长度（字节）	0
	字地址	———
AQ	长度（字节）	0
	字地址	———
HC	长度（字节）	16
	双字地址	%HC0，%HC1,%HC2，%HC3
V	长度（字节）	4096
	位地址	%V0.0 ---%V4095.7
	字节地址	%VB0 --- %VB4095

	字地址	%VW0 --- %VW4094
	双字地址	%VD0 --- %VD4092 %VR0 --- %VR4092
M	长度（字节）	1024
	位地址	%M0.0 --- %M1023.7
	字节地址	%MB0 --- %MB1023
	字地址	%MW0 --- %MW1022
	双字地址	%MD0 --- %MD1020
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

表 3-5-1 Kinco-K2 内存区域的范围

3.6.5 关于功能块以及功能块实例

3.6.5.1 IEC61131-3 中定义的标准功能块

在 IEC61131-3 中定义了如下标准的功能块：

- 定时器：TP --- 脉冲定时器；TON --- 接通延时定时器；TOF --- 断开延时定时器。
- 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。

3.6.5.2 FB 的实例化

在 IEC61131-3 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。

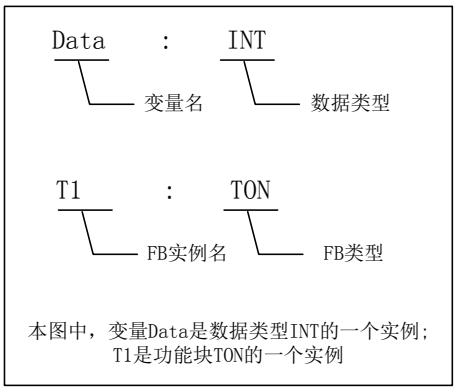


图 3-1　实例化举例

3.6.5.3　FB 实例存储区

Kinco-K 系列在内存中为每一种 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，KincoBuilder 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 Kinco-KK 系列的 FB 实例存储区域。

T	
描述	定时器区，可在该区域内分配 TON、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	不允许掉电保持，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

表 3-6　FB 实例的存储区

3.6.6　FB 实例的命名及使用

FB 的实例遵循“先声明，后使用”的原则。

为了方便用户，在 KincoBuilder 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	Tx
	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。 Tx 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	Cx
	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。 Cx 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	RSx
	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	SRx
	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

3.6.7 FB 实例存储区的范围

Kinco-K 系列 CPU 的内存中为每一种 FB 类型都分配了一个存储区域。当定义了某种 FB 的实例时，KincoBuilder 会自动在该 FB 类型对应的存储区域内为每个实例分配一个独立的存储单元。

下表描述了 Kinco-K 系列 CPU 为每种 FB 分配的实例存储区域。

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	32
	范围	SR0 --- SR31

表 3-7 FB 实例存储区的分配

第四章 使用 KincoBuilder … 基本功能

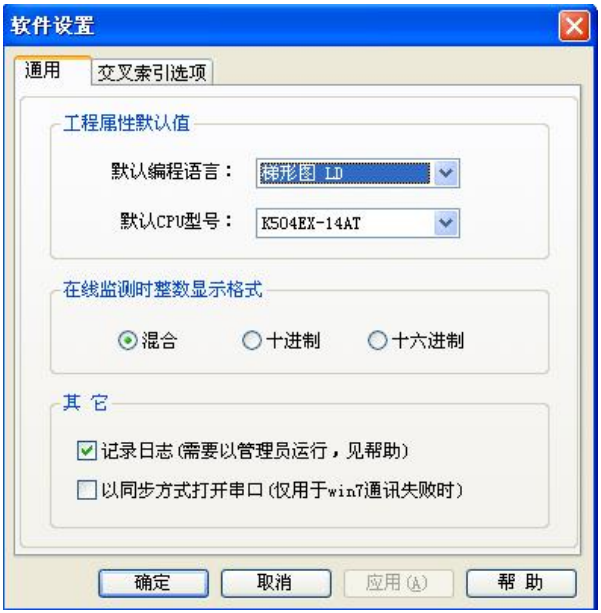
本章对 KincoBuilder 软件各部分的功能、使用进行了详细的描述，用户在掌握上一章介绍的基本概念的基础上，通过阅读本章可以快速认识并理解 KincoBuilder 的具体功能以及操作。

LD 编辑器和 IL 编辑器的使用将涉及到 IEC61131-3 标准中的许多语法，在本章中未作介绍，相关内容以及语法将在下一章节进行详细描述。

4.1 KincoBuilder 软件设置

用户在正式使用 KincoBuilder 之前需要对一些软件默认值进行设置，设置结果将保存在当前计算机中，以后 KincoBuilder 在运行时将自动读取并使用这些设置值。执行【工具】→【软件设置…】菜单命令，将弹出“软件设置”对话框。

① 【通用】页面



➢ 工程属性默认值

此处允许对 KincoBuilder 工程的一些属性进行设置。

- **默认编程语言**：选择程序刚创建时的默认编程语言，IL 或者 LD。
- **默认 CPU 型号**：选择工程刚创建时其硬件配置中默认使用的 CPU 型号。

➢ **在线监视时整数显示格式**

选择在线监视时在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：所有整型数据都以十六进制格式显示。

➢ 其它

- **记录日志**

若选中此项，那么 KincoBuilder 运行时会在安装目录下新建立一个 KincoPlcLog 子目录，在这个目录中，每天都会生成一个日志文件，日志中记录了 KincoBuilder 所有的操作，便于发生问题时进行跟踪分析。若操作系统是 Win7 或者 Win8，那么必须以管理员权限运行 Kincobuilder 才会记录日志。日志只记录当天的操作，旧记录会被自动删除。

- **以同步方式打开串口**

有些用户在 Windows7 或更高版本的操作系统下使用某些 USB 转 RS232 转换器时,可能会出现与 PLC 通信失败的问题。通常这是由该转换器驱动程序的兼容性引起的。

若出现这种问题，那么选中“以同步方式打开串口”，然后单击“确定”按钮退出。以后 Kincobuilder 将以同步方式操作串口，在大多数情况下能够解决这个问题。

② 【交叉索引选项】页面

编译后，在交叉索引表中可以列表显示出用户工程中用到的变量及其被使用的位置。

Kincobuilder 默认显示用到的所有变量。

在这个选项页面中，可以选择仅显示工程中用到的某个【内存区】及【数据类型】的变量，便于用户查询。比如上图中的选项，就指定了在交叉列表中只显示用户工程用到的 M 区的位变量。

4.2 浮动窗口

工程管理器、指令集窗口、信息输出窗口、PLC 模块列表窗口均被设计成浮动窗口，它们可以停靠在 KincoBuilder 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。

4.3 PLC 硬件配置

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。

当用户新建一个工程时，KincoBuilder 将自动加入一个默认的 CPU 模块，它的型号是由用户在“软件设置”对话框中指定的**【默认 CPU 型号】**。用户可以按实际需求自行修改所有的配置信息。

硬件配置信息必须下载到 CPU 中以后才会生效。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，并将检测到的实际信息与储存的配置信息进行比较，若用户配置的模块类型与实际连接的模块类型不一致，PLC 就会进入 STOP 状态，否则就进入正常运行状态。

硬件配置窗口的样式如下图。从图中可以看出，窗口可分为两部分：

图 4-2 硬件配置窗口

➢ 模块列表

在硬件配置窗口的上半部分，以表格形式列出了工程中用到的 PLC 模块，我们称之为模块列表。模块列表表达了一个真实的 PLC 控制系统：各个模块在表格中的排列次序应当与实际模块在扩展总线上的连接次序一致。

➢ 模块参数配置窗口

在硬件配置窗口的下半部分，是 PLC 模块列表中选中模块的所有配置参数，我们称之为模块参数配置窗口。

4.3.1 如何进入硬件配置窗口？

有如下两种方式可以进入硬件配置窗口：

- 双击工程管理器中【**资源**】组下的【**PLC 硬件配置**】节点；
- 在【**PLC 硬件配置**】节点上单击鼠标右键，然后执行弹出菜单的【**打开…**】命令。

4.3.2 在不同工程中复制和粘贴硬件配置信息

在 Kincobuilder 中，允许用户在不同工程中复制、粘贴【硬件配置】信息。注意，这个功能操作的是各种配置信息，比如 CANOpen 配置、通信口配置等，不会复制、粘贴具体的 CPU 型号，因此，在不同的 CPU 型号之间也可以进行操作。另外，待操作的各个工程必须分别使用 KincoBuilder 打开之后才可以进行操作。此功能与梯形图的复制粘贴，指令表的复制粘贴都不冲突，可以同时用。

用户希望移植或者继承旧工程中的 CANOpen 网络配置、通信口配置等信息，那么这个复制、粘贴【硬件配置】功能将会非常有用。

该功能的使用方法有如下 2 种：

- 在【编辑】菜单中执行【复制硬件配置】、【粘贴硬件配置】命令即可。
- 在【工程管理器】树中的【PLC 硬件配置】节点上单击右键，然后执行【复制硬件配置】、【粘贴硬件配置】命令

4.3.3 添加、删除模块

➢ 添加模块

用户可以按照如下步骤来添加一个模块：

① 鼠标单击模块列表中将要加入模块的位置，将焦点置于该行。
若该行已有模块存在，则必须首先删除已存在的模块，然后才能够加入新的模块。

② 在右侧的【PLC 模块列表】窗口中双击要加入的模块即可。
第 1 行（序号为 1 的行）只能加入 CPU 模块，其余各行只能加入扩展 I/O、功能模块。

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。

➢ 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**】命令也可。

4.3.4 配置模块参数

Kinco-KK 系列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 KincoBuilder 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。当然，用户也可以在模块列表中使用键盘上的 Up 和 Down 箭头键来改变选中模块的位置。

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！

4.3.4.1 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 4-3　【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的参数。如下图。

图 4-4 【通讯设置】参数配置页面

➢ PORT0（部分经济型系列 PLC 不具备此通讯口）

- **站号** ：为 PORT0 指定一个唯一的站号。该站号是作为 Modbus RTU 从站时的站号。
- **波特率** ：选择波特率：1200、2400、4800、9600、19200、38400、57600、115200。
- **奇偶校验**：选择奇偶校验方式：无校验、奇校验、偶校验。
- **数据位** ：选择数据位：8。
- **停止位** ：选择停止位：1。

➢ PORT1 和 PORT2（部分经济型系列 PLC 不具备此通讯口）

PORT1 和 PORT2 是 RS485 接口。

- **作为 Modbus 主站**：若选中此项，那么该 PORT 口就作为 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。
- **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。
- **波特率** ：K5 系列可选择波特率：1200、2400、4800、9600、19200。
  K2 系列可选择波特率：1200、2400、4800、9600、19200、38400、57600、115200。
- **奇偶校验**：选择奇偶校验方式：无校验、奇校验、偶校验。
- **数据位** ：选择数据位：8。
- **停止位** ：选择停止位：1。

当主站发出一个命令后，在下述情况下会产生通讯错误：

- 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；
- 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。当 CPU 掉电时，数据保持区域中的数据将由后备电池供电来得到保护，以供再次上电时使用。在常温下保持的时间不低于 3 个月。

注意：“初始化数据表”、“数据保持”以及“数据备份”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、恢复用户使用指令永久保存的数据。

数据保持页面如下图。

图 4-5 【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** ：指定被保护的数据区所在的内存区，有 V 区、C 区两个选项。
  对于计数器（C 区），只有其当前计数值可以保持。
- **起始地址**：指定该保持区域的起始字节地址或者计数器编号。

• **长度**　　：指定该保持区域的长度，单位：字节或者是计数器的个数。

如图 4-5 中，区域 1（VB0 - VB9）、区域 2（VB100 - VB109）、区域 3（C0 - C9）和区域 4（C20 - C29）中的数据在 CPU 掉电时将会保持。

④ **【本体 AI/AO】**页面

K506EA-30AT 本体集成了 4 路 AI，各通道的映像区地址分别固定为 AIW0、AIW2、AIW4 和 AIW6。每通道的采样转换速度约为 30 次/秒。

另外，本体还集成了 2 路 AO，各通道的映像区地址分别固定为 AQW0、AQW2。每通道的转换速度约为 30 次/秒。

这些集成的 AI、AO 点需要在【PLC 硬件配置】硬件配置中来选择各通道的信号形式和滤波方式等参数，配置界面如下：

⑤ **【CANOpen 主站】**页面（部分经济型 CPU 不具备此功能）

见附录 E。

⑥ **【其他】**页面

1. 将 VB3648-4095 永久保持不勾选时， 自动永久保持的区域与 K3 兼容， 因为自动保存的数据会影响下次 PLC 上电时运行的逻辑。

如果用户不是移植的 K3 PLC 的用户程序， 可以根据自己的需要来勾选这个选项， 如果用户需要比 K3 更多 V 区被永久保存， 则可以选中。

2. 将整个用户工程备份到 PLC 的永久存储中。

用户新建立的工程， 默认都会勾选此项， 这个功能会将用户编写的 Kincobuilder 工程使用 zip 压缩的方式， 嵌入到下载的 OBJ 中， 在用户上传工程时， 会从这个 zip 文件解压出用户工程。

- 相比其他的上载功能，这种上载方式可以保留用户的注释和程序中的各种中文名称。
- 这个功能会占用 EEPROM 存储， 当用户的程序大到无法压缩到 EEPROM 中时， 会自动忽略此选项，变成只能上载用户工程而不能上载用户的注释等信息。
- 这个选项会加长用户下载工程的时间。
- 如果你不需要上载 PLC 工程， 或者你是在调试你的工程， 可以把这个选项去掉。

4.3.4.2 DI 模块的参数配置

DI 模块的参数配置非常简单，如下图所示。

图 4-6 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 4-6，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。

4.3.4.3 DO 模块的参数配置

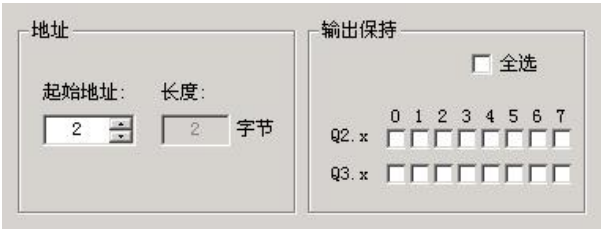


图 4-7 DO 模块参数配置

➢ 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 4-7，该模块具有 16 个 DO 通道，模块的起始地址是%QB2，它的通道的地址是 %Q2.0－%Q3.7。

➢ 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。

4.3.4.4 AI 模块的参数配置

图 4-8 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 4-8，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。

➢ 通道设置

- **信号形式**：为各个通道选择输入信号的形式，比如 4-20mA、1-5V 等。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“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点的软件滤波器。

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。

4.3.4.5 AO 模块的参数配置

图 4-9　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 4-9，模块的起始地址被指定为%AQW0，该模块具有 2 个 AO 通道，它的 2 个通道的地址依次是%AQW0、%AQW2。

➢ 通道设置

- **信号形式**：为各个通道选择所输出信号形式，比如 4-20mA、1-5V 等。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〔1,5〕V，停机时希望输出 1V，则应当在此处输入 1000。

4.4 初始化数据表

在初始化数据表中用户可以为 V 区中 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	%VB0	B#1	B#2		
2	%VW10	2	3	4	
3	%VD100	DI#100	DI#200	DI#2000	DW#2456
4	%VD3840	3.45			
5					

图 4-10 初始化数据表

注意：“初始化数据表”、“数据保持”以及“数据备份”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、恢复用户使用指令永久保存的数据。

4.4.1 如何进入初始化数据表？

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【**程序**】组下的【**初始化数据表**】节点；
- 在【**初始化数据表**】节点上单击鼠标右键，然后执行弹出的【**打开**...】菜单命令。

4.4.2 在表格单元中输入数据

表格单元得到焦点则会自动进入编辑状态，或者鼠标单击某表格单元也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。

4.4.3 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：

① 在〔起始地址〕列中输入一个直接地址；

② 在〔初始值〕列中输入一个或者多个数值。若输入多个数值，则 KincoBuilder 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。

如图 4-10，第 1 行的含义是为%VB0、%VB1 分别赋初始值 B#1、B#2；第 2 行的含义是为%VW10、%VW12、%VW14 分别赋初始值 2、3、4；第 3 行的含义是为%VD100、%VD104、%VD108、%VD112 分别赋初始值 DI#100、DI#200、DI#2000、DW#2456。

4.4.4 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。

4.5 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 IEC61131-3 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请参见 3.3.1 标识符的定义 部分。

关于全局变量的更多信息请参见 3.5 变量 。

在本书中，“全局变量表”通常就指这个页面。该页面样式如下图所示。


VAR_GLOBAL

	全局变量名	地 址	数据类型	注 释
1	MI_Guzhang1	%M0.2	BOOL	1#泵故障
2	MI_Jianxiu1	%M0.5	BOOL	1#泵检修
3	MI_Yingji1	%M1.0	BOOL	1#消防应急
4	MQ_JY1	%M2.0	BOOL	1# 泵降压运行
5	MQ_QY1	%M2.4	BOOL	1#泵全压运行
6	T_JQ	%VW4	INT	降压后的延时
7	T_Beng	%VW6	INT	泵之间的启动延时
8				

全局变量 / 功能块实例

图 4-11 **【全局变量】**页面

- **【功能块实例】**页面

在 3.6.6 FB 实例的命名及使用 中提到过，为了方便用户的使用，功能块实例的定义是由 KincoBuilder 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。页面如下图


VAR_GLOBAL

	FB实例名称	FB类型	FB实例定义位置
1	T5	TON	SBR_运行
2	T6	TON	SBR_运行
3	T7	TON	SBR_运行
4	T8	TON	SBR_运行
5	T9	TON	SBR_运行

全局变量 / 功能块实例

图 4-12 **【功能块实例】**页面

4.5.1 如何进入全局变量表？

有如下三种方式可以进入全局变量表窗口：

- 双击工程管理器中**【资源】**组下的**【全局变量表】**节点；
- 在**【全局变量表】**节点上单击鼠标右键，然后执行弹出的**【打开...】**菜单命令。
- 执行**【工程】→【全局变量表】**菜单命令。

4.5.2 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：

① 进入全局变量表窗口并切换至**【全局变量】**页面；
② 在〔全局变量名〕列中输入一个变量名称并确认；
③ 在〔地址〕列中输入一个直接内存地址并确认；
④ 在〔数据类型〕列中选择一个为变量选择一个数据类型并确认；
⑤（可选）在〔注释〕列中为这个全局变量输入注释。

如图 4-11 的第 1 行的含义是：为地址“%M0.2”定义一个符号名称为“MI_Guzhang1”，其数据类型为“BOOL”。在程序中，“%M0.2”与“MI_Guzhang1”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。

4.6 交叉索引表

交叉索引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。使用交叉索引表便于用户对当前的工程进行统计、分析。

交叉索引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。

交叉索引表如下图所示：

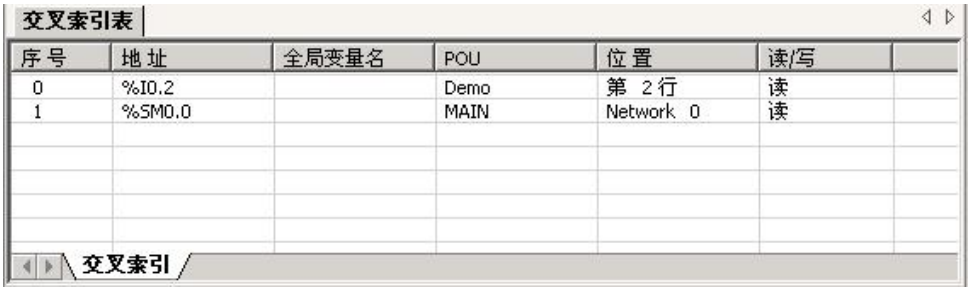


图 4-13　交叉索引表

- 〔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。
- 〔POU〕　　：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〕上被进行了读或者是写操作。

如图 4-13，表格中第一行的意思是：在本工程 *Demo* 程序的 *第 2 行* 用到了一次 *%I0.2*，此处对*%I0.2* 进行的是 *读* 操作。

4.6.1　如何进入交叉索引表？

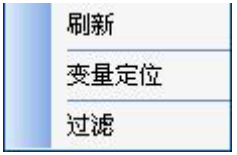
有如下三种方式可以进入交叉索引表窗口：

- 执行【**工程**】→【**交叉索引表**】菜单命令；
- 鼠标单击工具栏上的  图标。
- 使用 Alt+C 快捷键。

4.6.2　在交叉索引表中进行操作

➢　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**：刷新显示交叉索引数据。
- **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**：弹出【软件设置】对话框中的【交叉索引选项】页面，用户可以选择仅显示工程中用到的某个内存区且数据类型相符的变量。

4.7　变量状态表

用户可以利用变量状态表来对 PLC 中的的内存变量进行在线监视和强制。在变量状态表中，

提供了内存监视表和变量状态表两个页面。

➢ 内存监视表

用于监测 PLC 的任意内存地址。

内存监视功能支持 K 系列所有 PLC，但 K5，K2 系列支持监视所有内存， K3 系列只支持监视 I，Q，AI，AQ，M，V 区。

内存监视表样式如下图：

表中共分如下：

- 〔内存地址〕：输入被监测的内存区域的起始地址。
- 〔监视长度〕：输入从“内存地址”开始要监测的数据个数，最大允许个数为 150。
  表格中每行最多显示 3 个数据，若输入的长度超过 3，那么软件会自动分行显示。
- 〔显示格式〕：选择内存值的显示格式，包括十进制、16 进制。
- 〔内存值〕 ：显示所监视的内存值。每行最多显示 3 列，其中第一列是起始内存的值。
  若监视的是位内存，那么只显示 TRUE、FALSE，而“显示格式”就无效了。

➢ 变量状态表

变量状态表用于对当前工程中用到的任意地址变量进行在线监视和强制。

变量状态表的样式如下图所示。

变量状态表

	地 址	全局变量名	显示格式	当前值	强制值
1	%M0.0	MI_Gaoshuiwei	BOOL	2#0	
2	%M0.1	MI_Dishuiwei	BOOL	2#0	
3	%VW4	T_JQ	无符号整数	W#2000	
4	%VW6	T_Beng	有符号整数	300	
5					
6					

图 4-14 变量状态表

表中共分如下五列：

- 〔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、有符号数、无符号数、二进制、16 进制等。
- 〔当前值〕　：在线监视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〕　：在线监视时可以在此输入本行〔地址〕将要被强制为的值。

注意：为了提高效率，在变量状态表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，KincoBuilder 虽然不提示错误，但当前值、强制值均不会起作用。

4.7.1 如何进入变量状态表？

有如下三种方式可以进入变量状态表窗口：

- 双击工程管理器中的**【变量状态表】**节点；
- 在**【变量状态表】**节点上单击鼠标右键，然后执行弹出的**【打开...】**菜单命令；
- 执行**【调试】→【变量状态表】**菜单命令。

4.7.2 监视变量的值

用户可以按如下步骤来利用变量状态表监视用户程序中用到的变量的值：

① 在〔地址〕列中输入要监视的直接内存地址；

② 使用如下方法之一进入在线监视状态：

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

③ 用户可以在任何时候改变监视值的〔显示格式〕。

4.7.3 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。

K5 允许同时强制最多 32 个变量。立即指令不允许强制。

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。

4.7.4 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变量状态表中。

4.7.5 强制、取消强制

在变量状态表中，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：

① 在〔地址〕列中输入要强制的直接内存地址。

② （可选）选择数值的〔显示格式〕。当然显示格式也可以随时修改。

③ 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，KincoBuilder 会按照〔显示格式〕对数值的格式自动进行调整。

④ 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：

⑤ 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。

变量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。

4.8 密码保护

Kinco-K 系列的 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。

K 系列密码长度允许 8-14 个字符，可以由数字、字母、下划线组成，字母区分大小写。

若用户将 PLC 保护等级设置为“等级 3：最高保护”并设置密码， 或者启用了“禁止上载”功能，PLC 存储的用户程序将全字节进行加密，以密文的形式进行存储，以防止被用直接读硬件芯片的方法破解。

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。密码输入错误次数受到限制，当密码输入错误到达限定次数后，必须重启 PLC 才能继续执行带密码的操作，以避免暴力破解。

4.8.1 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。设置密码后，PLC 存储的用户程序将全将全字节进行加密，以密文的形式进行存储，以防止被用直接读硬件的方法破解。

4.8.2 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。如下图：

图 4-15 “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，则让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。

  密码允许 8-14 个字符，可以由数字、字母、下划线组成，字母区分大小写。

  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，而且 KincoBuilder 将会有对话框弹出来提示修改的结果。

4.8.3 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。

第五章 使用 KincoBuilder 编写用户程序

本章对 KincoBuilder 中 LD 和 IL 编辑器的功能、使用进行了详细的描述，同时也阐述了 IEC61131-3 标准中关于 LD、IL 语言的相关语法、规定。通过阅读本章，用户可以轻松地使用 KincoBuilder 编写出符合 IEC 标准的应用程序。

针对 PLC 应用程序的编写，IEC61131-3 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）。

KincoBuilder 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【工程】→【IL 语言（指令表）】或者【工程】→【LD 语言（梯形图）】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。

5.1 IL 编程

5.1.1 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、吸收世界上各著名 PLC 厂商的指令表语言的基础上而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。

5.1.2 IL 的语法规定

5.1.2.1 IL 语句格式

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）**
  使用标号的目的就在于实现程序的跳转。标号的命名格式如同变量。
- **操作符/功能/功能块**
  即 PLC 指令。
- **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，其中也包含了相应操作数的描述。
  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，嵌套的注释编译器将认为是错误。

IL 语句举例如下：

```
(* NETWORK 0 *)
Run:                (* 标号，供跳转时使用 *)
LD      %I1.0
TP      T2, 168     (* 若 I1.0 为 true, 启动定时器 T2，T2 被声明为 TP 型 *)
```

5.1.2.2 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 KincoBuilder 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。

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

表 5-1　各操作符执行之后对于 CR 的影响

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。

5.1.2.3　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这两部分组成。在 KincoBuilder 中，关于网络中的格式有如下规定：

- 在一个网络中可以只有一个语句标号。例如：

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

- 在一个网络中可以只有程序语句。

  在 5.1.2.2 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  %M3.5  (* 以 LD 指令开始 *)
ST  %Q2.3  (* 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 *)
MRun:     (* 以语句标号开始 *)
LD %M3.5
ST %Q2.3  (* 以允许的指令结束 *)
```

5.1.3 KincoBuilder 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。

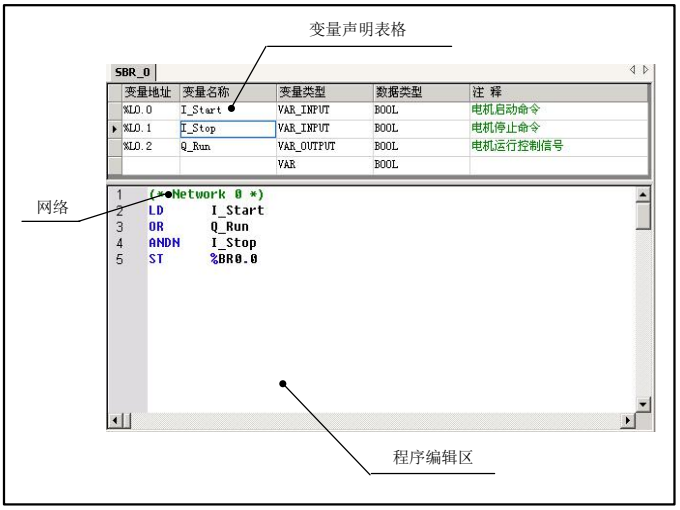


图 5-1 IL 编辑器

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。

5.1.3.1 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【**加入新网络**】弹出菜单命令。

➢ 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。

复制或者箭切完毕后，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：

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：

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。

5.1.3.2 IL 程序示例

```
(* NETWORK 0 *)
LDN     %M0.0
TON     T0, 1000        (* 依靠 T1 的输出来启动 T0，定时为 1000×1ms *)
ST      %M0.1
(* NETWORK 1 *)
LD      %M0.1
TON     T1, 1000        (* 依靠 T0 的输出来启动 T1，定时为 1000×1ms *)
ST      %M0.0
(* NETWORK 2 *)
LD      %M0.1
ST      %Q0.0           (* 在 Q0.0 输出方波 *)
```

5.1.4 IL 程序转换为 LD 程序

执行【工程】→【LD 语言（梯形图）】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。

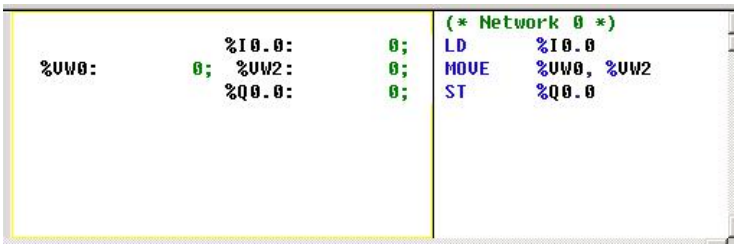
5.1.5 调试和监视程序

5.1.5.1 在线监视 IL 程序

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视状态。

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

在线监视状态下，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。



注意：在线监视状态下不允许对程序进行编辑。

5.1.5.2 强制指定变量

在前面 4.7.3 关于强制功能 中对 KINCO-K 系列的强制功能进行了详细的描述。

在线监视 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

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 3.4 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。

5.2 LD 编程

5.2.1 LD 的背景

LD（梯形图）语言是 IEC61131-3 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。

5.2.2 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。

5.2.3 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。

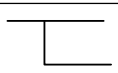
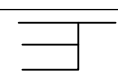
图形对象	名 称	描 述
	水平连接	可以将其理解为一根理想的导线。 水平连接将线上任一点左侧的值传递到右侧。
	垂直连接 （含有水平连接）	垂直连接将它左侧所有水平连接的值首先进行逻辑“或”运算，然后再将运算结果传递到它右侧所有的水平连接上。
		
		

表 5-2 LD 中的连接

➢ 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。

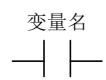
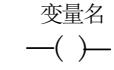
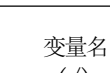
图形对象	名 称	描 述
	常开接点	若变量值为 TRUE，则触点闭合，它左侧连接的值被传递到右侧连接；否则触点断开，它右侧连接的值为 FALSE。
	常闭接点	若变量的值为 TRUE，则触点断开，它右侧连接的值为 FALSE；否则触点闭合，它左侧连接的值被传递到右侧连接。

表 5-3 LD 中的接点

➢ 线圈

线圈是用于给布尔型变量赋值的图形元素。

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

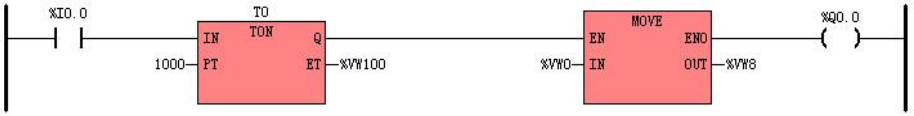
表 5-4　LD 中的线圈

➢　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。

下图是一个调用功能和功能块的示例。



5.2.4 KincoBuilder 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。

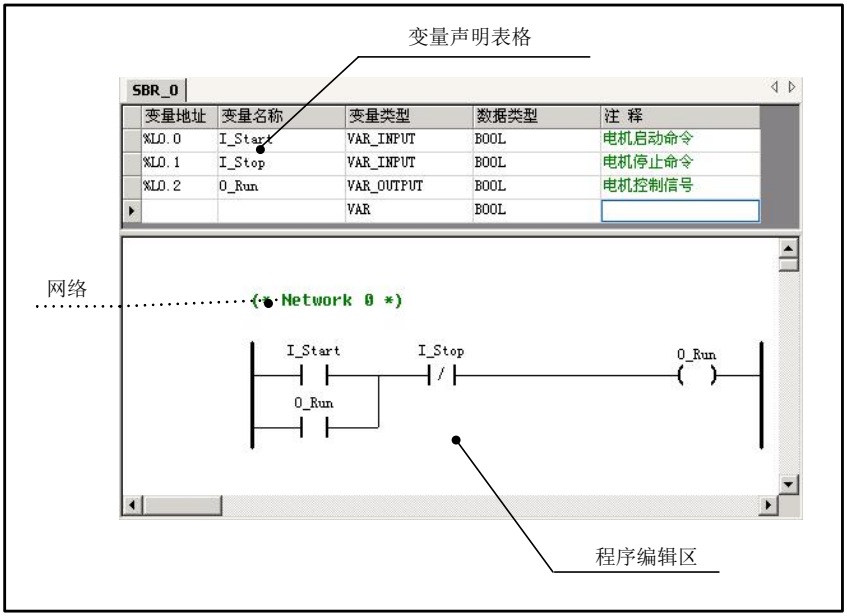


图 5-2 LD 编辑器

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。

5.2.4.1 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：

- 一个程序（主程序，子程序或者中断程序）中最多允许 200 个网络。
- 对可添加的字程序或者中断程序数量无限制，但所有程序网络中用到的元件总数量不超过 4096。
- 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。

5.2.4.2 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

① 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。

② 使用如下任一方法在程序中加入一个新网络：

- 执行【**梯形图**】→【**新网络**】菜单命令。
- 单击工具栏上的图标。
- 使用 Ctrl+W 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。

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

① 在期望添加的位置上标记一个元件作为基准。

② 使用如下任一方法在程序中加入一个串联触点：

- 执行【**梯形图**】→【**左触点**】或者【**右触点**】菜单命令。
- 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。
- 鼠标双击窗口右侧“**LD 指令集**”中所需要的触点指令。

➢ 添加一个并联触点

① 在期望添加的位置上标记一个触点元件作为基准。

② 使用如下任一方法在程序中加入一个并联触点：

- 执行【**梯形图**】→【**并联触点**】菜单命令。
- 单击工具栏上的图标。
- 使用 Ctrl+D 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

① 在期望添加的位置上标记一个线圈元件作为基准。

② 使用如下任一方法在程序中加入一个并联线圈：

- 执行【**梯形图**】→【**并联线圈**】菜单命令。
- 单击工具栏上的图标。
- 使用 Ctrl+D 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。
- 鼠标双击窗口右侧“**LD 指令集**”中所需要的线圈指令。

➢ 添加一个串联的功能/功能块

① 在期望添加的位置上标记一个元件作为基准。

② 使用如下任一方法在程序中加入一个串联的功能/功能块：

- 执行【**梯形图**】→【**功能/功能块**】菜单命令；或者单击工具栏上的图标；或者使用 Ctrl+B 快捷键；或者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：

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。

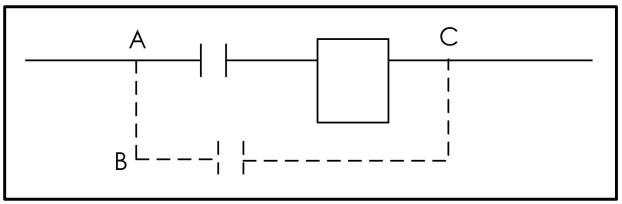
- 鼠标双击窗口右侧“**LD 指令集**”中所需要的功能、功能块指令或者子程序。

➢ 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：

① 执行【**梯形图**】→【**并联支路**】菜单命令；或者单击工具栏上的图标；或者使用 **Ctrl+H** 快捷键；或者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。

② 使用鼠标继续按如下步骤来画一个并联支路：



1) 在期望的并联起始位置处（如上图中A）单击；
2) 向上或者向下移动鼠标至任意位置（如B）并单击；
3) 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。

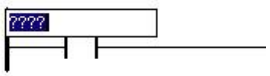
注：上面提到的三个位置并不需要太精确，用户在“大约”的位置处单击即可

➢ 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。



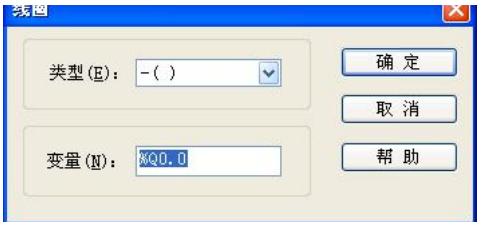
用户直接在此输入框内输入期望的常数、直接地址、全局变量或者局部变量名称，然后按 Enter 键确认即可。LD 编辑器会自动对用户输入的直接地址进行格式化，因此用户在输入时可以不输入百分号（%）。在输入时，也可以按 ESC 键取消当前的修改。

- 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，然后按照上面一个方法进行输入即可。
- 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

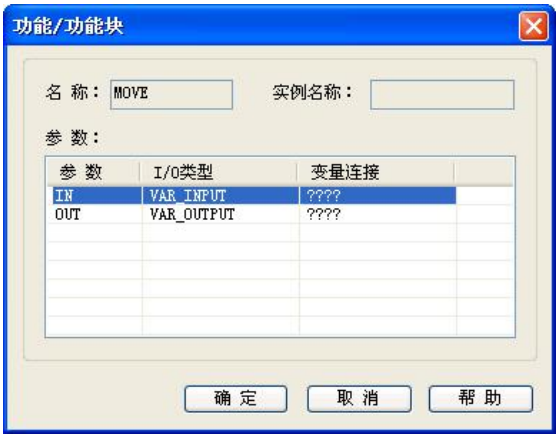
用户可以修改该元件的【**类型**】、连接的【变量】，然后单击【**确定**】按钮即可。

✓ 线圈的属性对话框



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



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。

KincoBuilder 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、数据类型错误、内存区域类型错误等。

所有输入完成后，单击【**确定**】按钮即可。

➢ 删除一个元件

① 标记期望删除的元件。

② 使用如下任一方法删除标记的元件：

- 执行【**梯形图**】→【**删除元件**】菜单命令。
- 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

① 在期望删除的网络中标记任何一个元件。

② 使用如下任一方法删除标记元件所在的网络：

- 执行【**梯形图**】→【**删除网络**】菜单命令。
- 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。

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。

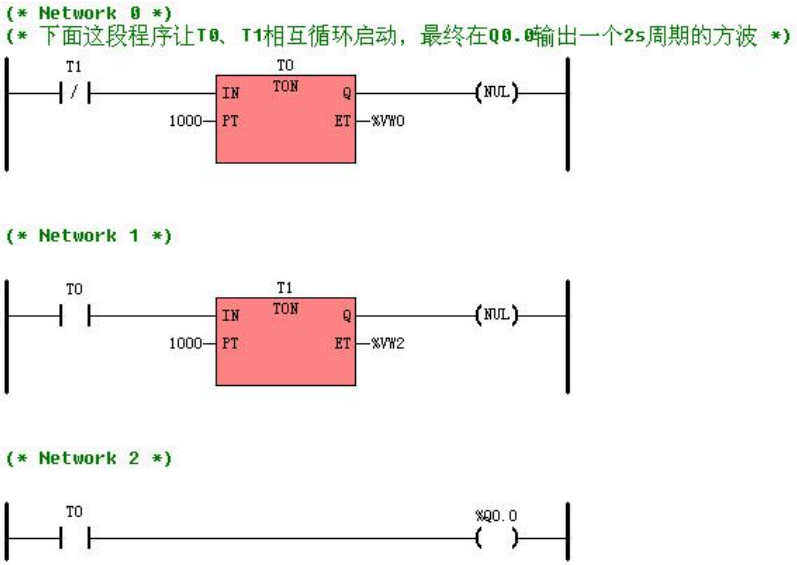
使用 **Delete** 键或者执行【**删除**】命令，可以删除选中的内容。

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。

复制或者箭切完毕后，在想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。

5.2.4.3 LD 程序示例



5.2.5 监视和调试程序

5.2.5.1 在线监视 LD 程序

注意：在线监视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视状态：

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

在线监视状态下，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。

下图是一个在线监视的程序示例：

注：黄色的锁代表强制值

5.2.5.2 强制指定变量

在前面 4.7.3 关于强制功能 那一节中对 KINCO-K 系列的强制功能进行了详细的描述。

在线监视 LD 程序时，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：

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 3.4 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。

第六章 K 系列 PLC 指令集

K 系列 PLC 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。

6.1 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了 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 5.1.2.2 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 5.1.2.2 中规定的“分组缩写”符号。

6.2 位指令

6.2.1 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			
	常闭触点			
IL	LD	LD *bit*	C	☑ K5 ☑ K2
	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。

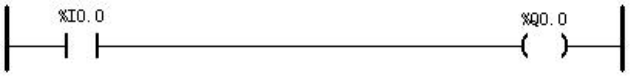
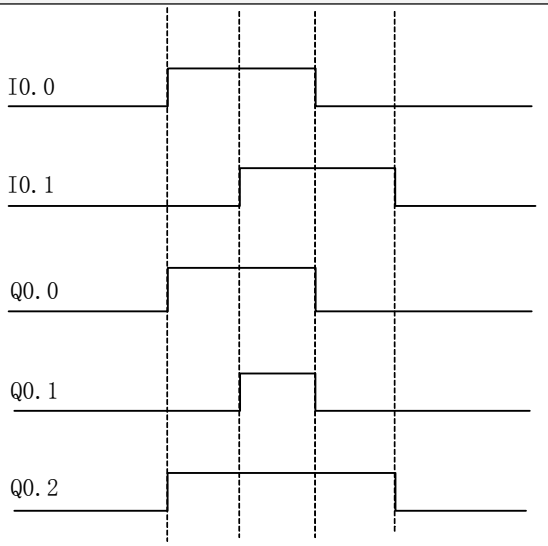
常闭触点由 *LDN*、*ANDN*、*ORN* 指令提供。

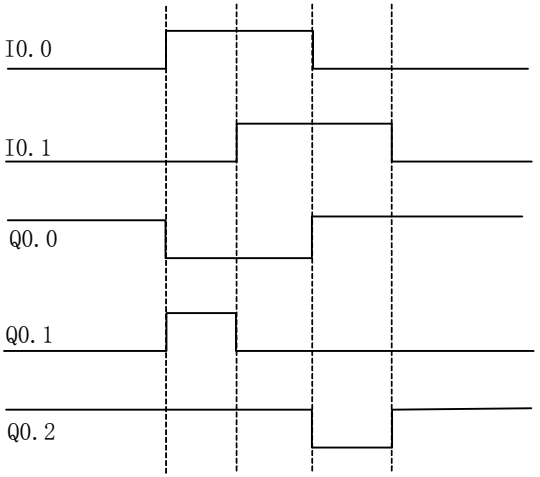
LDN 指令将 *bit* 值取反，然后将结果装载于 CR 中并作为新的 CR 值；

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 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。

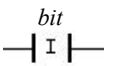
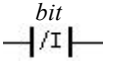
➢ 指令使用举例

LD	IL
	LD %I0.0 ST %Q0.0
	LD %I0.0 AND %I0.1 ST %Q0.1
	LD %I0.0 OR %I0.1 ST %Q0.2
时序图	
	

LD	IL
	LDN %I0.0 ST %Q0.0
	LD %I0.0 ANDN %I0.1 ST %Q0.1
	LD %I0.0 ORN %I0.1 ST %Q0.2
时序图	
	

6.2.2 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			☑ K5 ☑ K2
	立即常闭触点			
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。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。

立即触点指令只能用于 CPU 本体的 DI 点。在用户工程的【PLC 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。

- 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。

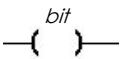
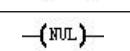
LDNI 指令将 *bit* 的物理输入通道的实际状态值取反，然后将结果装载于 CR 中并作为新的 CR 值；

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。

6.2.3 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			☑ K5 ☑ K2
	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

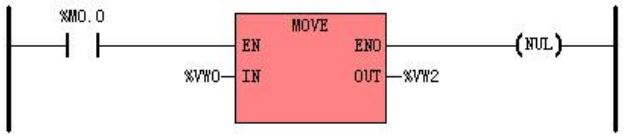
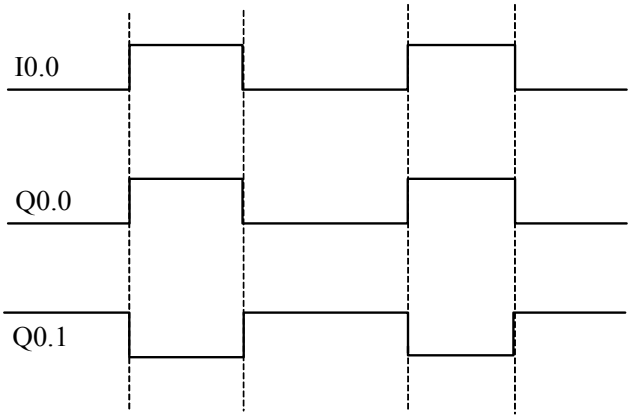
  线圈由 *ST* 指令提供，取反线圈由 STN 指令提供。

  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

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>LD %I0.0
ST %Q0.0
STN %Q0.1</td></tr>
<tr><td> 
</td><td>LD %M0.0
MOVE %VW0, %VW2</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.2.4 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	*bit* —(I)—		☑ K5 ☑ K2
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。

6.2.5 置位与复位

➢ 指令及其操作数说明

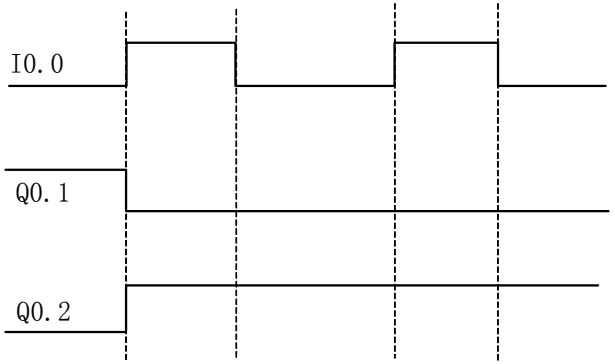
	名 称	指令格式	影响 CR 值	
LD	复位	*bit* —(R)—		☑ K5 ☑ K2
	置位	*bit* —(S)—		
IL	复位	R *bit*	U	
	置位	S *bit*		

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。

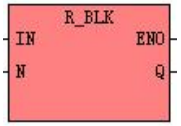
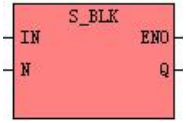
- IL
  复位指令的作用是：若 CR 值为 1，则 *bit* 值被置为 0，否则 *bit* 值保持不变。
  置位指令的作用是：若 CR 值为 1，则 *bit* 值被置为 1，否则 *bit* 值保持不变。
  R、S 指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>LD %I0.0
R %Q0.1
S %Q0.2</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2">假定执行该指令之前Q0.1和Q0.2的值分别为1和0
 
</td></tr>
</table>

6.2.6 块置位与块复位

➢ 指令及其操作数说明

	名 称	指令格式	影响 CR 值	
LD	块复位			☑ K5 ☑ K2
	块置位			
IL	块复位	R_BLK *N, Q*	U	
	块置位	S_BLK *N, Q*		

操作数	输入/输出	数据类型	允许的内存区
IN	输入	BOOL	能流
N	输入	INT	L、M、V、常量
Q	输出	BOOL	Q、V、M、SM、L

- LD

R_BLK 的作用：若 IN 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 0，否则这些位保持不变。

S_BLK 的作用：若 IN 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 1，否则这些位保持不变。

- IL

R_BLK 的作用：若 CR 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 0，否则这些位保持不变。

S_BLK 的作用：若 CR 值为 1，则从指定的位地址 *Q* 开始的连续 *N* 个位都被置为 1，否则这些位保持不变。

R_BLK、S_BLK 指令的执行不影响 CR 值。

参数 N 的最大数量为 1024。

参数 Q 为一个可变长度的内存块的起始地址，注意整个内存块都不允许落在非法内存区域，否则结果不可预期。

6.2.7 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)—		☑ K5 ☑ K2
	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

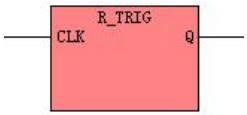
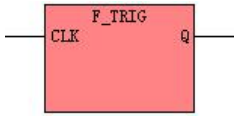
立即复位指令的作用是：若 CR 值为 1，则 *bit* 对应的输出映像寄存器和物理输出点均被置为 0，否则这两者的值保持不变。

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。

6.2.8 边沿检测

➢ 指令及其参数说明

	名 称	指 令	影响 CR 值	
LD	上升沿检测			☑ K5 ☑ K2
	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。

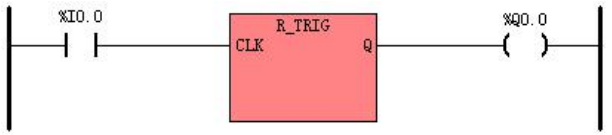
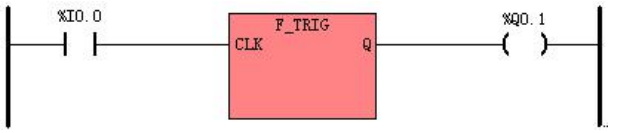
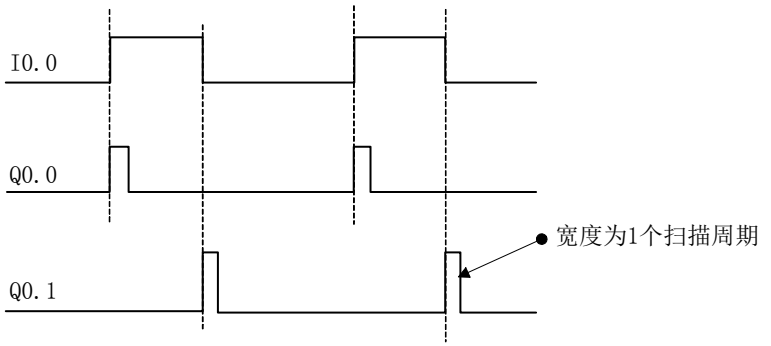
F_TRIG 用于检测 *CLK* 输入的下降沿跳变：如果 *CLK* 值产生了由 1 到 0 的跳变，则 *Q*输出为 1 并保持一个扫描周期，然后 *Q*将输出为 0。

- 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。

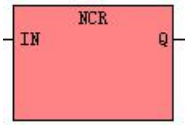
F_TRIG 用于检测当前点 CR 值的下降沿跳变：如果当前点的 CR 值产生了由 1 到 0 的跳变，则立即将 CR 值设置为 1，否则将 CR 值设置为 0。

➢ 指令使用举例

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>LD %I0.0
R_TRIG
ST %Q0.0</td></tr>
<tr><td> 
</td><td>LD %I0.0
F_TRIG
ST %Q0.1</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.2.9 NCR（取反）

➢ 指令及其参数说明

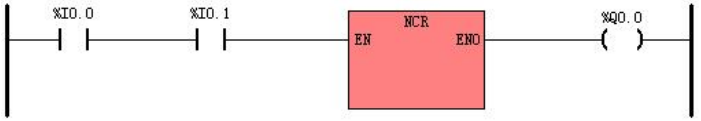
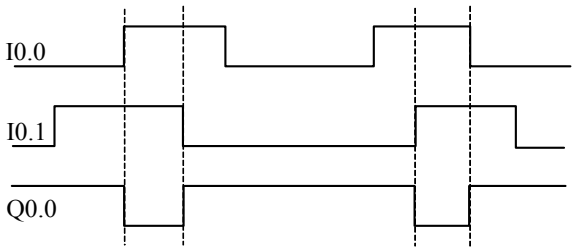
	名 称	指 令	影响 CR 值	
LD	取反			☑ K5 ☑ K2
IL	取反	NCR	P	

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
	LD %I0.0 AND %I0.1 NCR ST %Q0.0
时序图	
	

6.2.10 双稳态触发器

双稳态触发器是 IEC61131-3 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 3.6.5 关于功能块以及功能块实例 中的描述。

6.2.10.1 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	*SRx* S1 SR Q1 R		☑ K5 ☑ K2
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 触发器的置位端输入是 *S1*，复位端输入是 *R*。其真值表如下：

S1	*R*	输出 *Q1* 及 *SRx* 状态值
0	0	保持前一状态
0	1	0
1	0	1
1	1	1

6.2.10.2 RS（复位优先触发器）

➢ 指令及其操作数说明

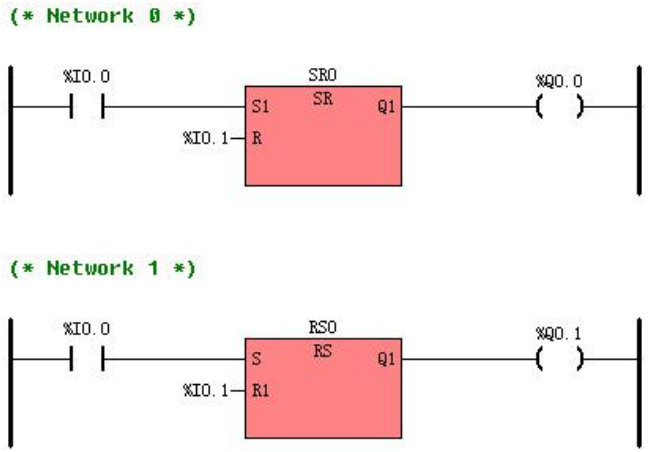
	名称	指令格式	影响 CR 值	
LD	复位优先触发器	*RSx* 		☑ K5 ☑ K2
IL	复位优先触发器	LD *S* RS *RSx*, *R1*	P	

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

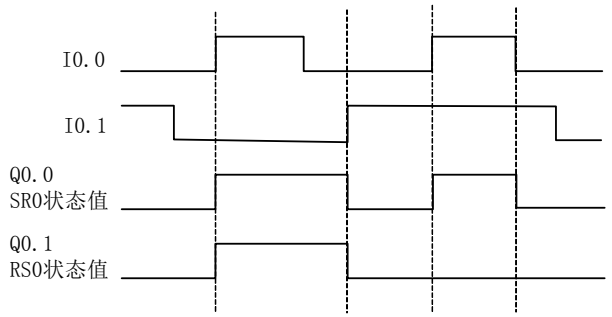
RS 触发器的置位端输入是 *S*，复位端输入是 *R1*。RS 触发器的真值表如下：

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

6.2.10.3 RS、SR 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>


</td><td>
(* Network 0 *)
LD %I0.0
SR SR0, %I0.1
ST %Q0.0

(* Network 1 *)
LD %I0.0
RS RS0, %I0.1
ST %Q0.1
</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2">


</td></tr>
</table>

6.2.11 ALT（反转）

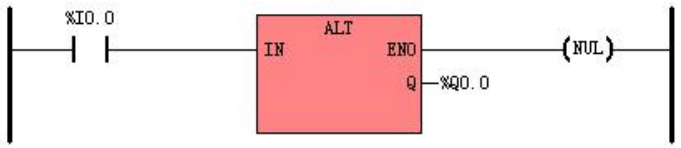
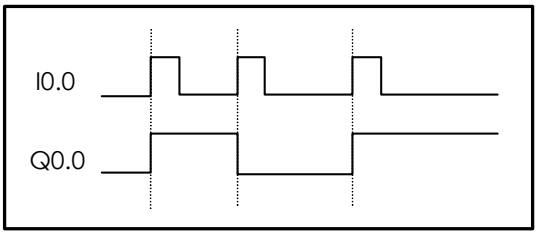
➢ 指令及其参数说明

	名 称	指 令	影响 CR 值	
LD	取反			☑ K5 ☑ K2
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
	LD %I0.0 ALT %Q0.0
时序图	
	

6.2.12 NOP（空操作）

➢ 指令及其参数说明

	名 称	指 令	影响 CR 值	
LD	空操作			☑ K5 ☑ K2
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。

PLC 实测数据如下：

新建一个空的用户工程，在此空工程中调用 12 个 NOP 指令，在每个 NOP 指令中，输入参数 N = 32767，下载此工程到 PLC，测试 PLC 扫描的时间为约 37ms，即 393204 个 NOP 执行时间为 37ms，37ms == 37000000ns，因此**单 NOP 平均执行时间约为 94ns，但实际中因为指令的调用开销，单 NOP 的实际时间可能会稍大于 94ns。NOP 不能作为精确的时间指令。**

6.2.13 括号修饰符

➢ 指令及其参数说明

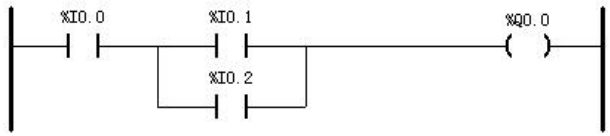
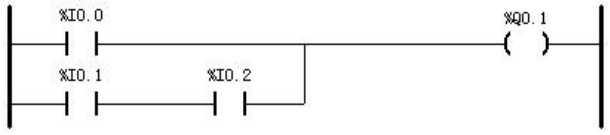
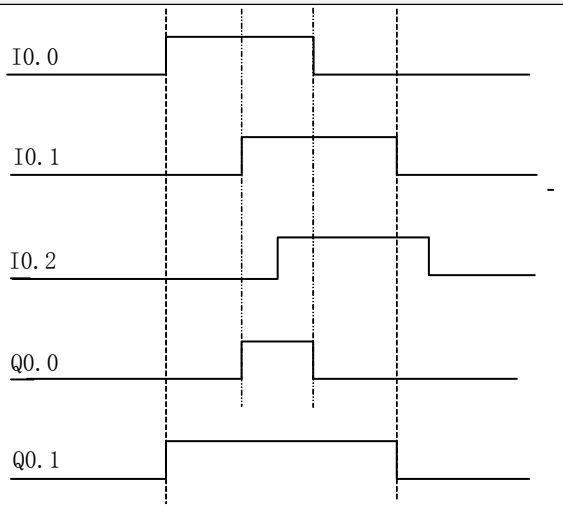
	名 称	指令格式	影响 CR 值	
IL	AND(	AND(	U	☑ K5 ☑ K2
	OR(	OR(		
	)	)	P	

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 IEC61131-3 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。

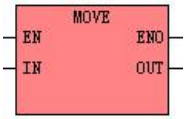
➢ 指令使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>LD %I0.0
AND(
LD %I0.1
OR %I0.2
)
ST %Q0.0</td></tr>
<tr><td> 
</td><td>LD %I0.0
OR(
LD %I0.1
AND %I0.2
)
ST %Q0.1</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.3 赋值指令

6.3.1 MOVE（赋值）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	MOVE			☑ K5 ☑ K2
IL	MOVE	MOVE IN, OUT	U	

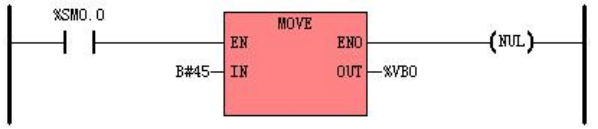
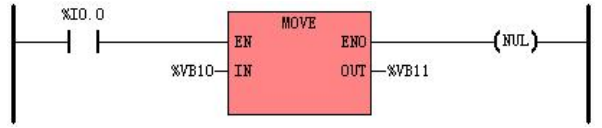
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE、WORD、DWORD、INT、DINT、REAL	I、Q、M、V、L、SM、AI、AQ、T、C、HC、常量、指针
OUT	输出	BYTE、WORD、DWORD、INT、DINT、REAL	Q、M、V、L、SM、AQ、指针

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*。

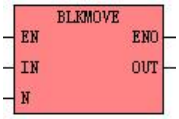
- IL
  如果 CR 值为 1，则该指令将输入变量 *IN* 的值赋给输出变量 *OUT*。
  该指令的执行对 CR 值无影响。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>SM0.0 固定为 1，因此 MOVE 指令总是执行：BYTE 类型常量 B#45 被赋给 VB0。</td></tr>
<tr><td> 
</td><td>I0.0 为 0：不执行 MOVE 指令。
I0.0 为 1：VB10 的值被赋给 VB11。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
MOVE B#45, %VB0 (* VB0 被赋值为 B#45 *)</td></tr>
<tr><td></td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
MOVE %VB10, %VB11 (* 若 CR 为 1：VB10 的值被赋赋给 VB11 *)
(* 若 CR 为 0：不执行 MOVE 指令，VB11 的值不变 *)</td></tr>
</table>

6.3.2 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			☑ K5 ☑ K2
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

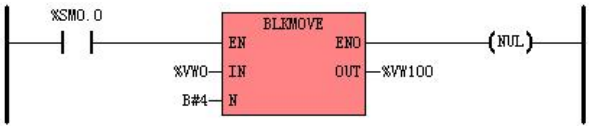
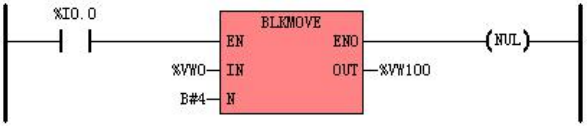
参数 *IN*、*OUT* 的数据类型必须一致。该指令用于将从地址 *IN* 开始的连续 *N* 个变量的值传送给从地址 *OUT* 开始的连续 *N* 个变量，这些变量的数据类型与 *IN*、*OUT* 一致。

- 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。

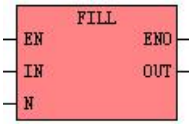
注意， IN、OUT 为一个可变长度的内存块的起始地址，整个内存块都不允许落在非法内存区域， 否则结果不可预期。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>SM0.0 恒为 1，因此 BLKMOVE 指令总是执行：VW0 至 VW6 中的数据被依次传送到 VW100 至 VW106 中。</td></tr>
<tr><td> 
</td><td>I0.0 为 0：不执行 BLKMOVE。
I0.0 为 1：VW0 至 VW6 中的数据被依次传送到 VW100 至 VW106 中。</td></tr>
<tr><td>IL</td><td colspan="2">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 *)</td></tr>
<tr><td></td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
BLKMOVE %VW0, %VW100, B#4 (* 若 CR 为 1:VW0 至 VW6 中的数据被传送到 VW100 至 VW106 中 *)
(* 若 CR 为 0：不执行 BLKMOVE 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">若 BLKMOVE 被执行，则结果举例如下：
<table>
<tr><td>VW0</td><td>VW2</td><td>VW4</td><td>VW6</td></tr>
<tr><td>0</td><td>10</td><td>20</td><td>30</td></tr>
<tr><td>VW100</td><td>VW102</td><td>VW104</td><td>VW106</td></tr>
<tr><td>0</td><td>10</td><td>20</td><td>30</td></tr>
</table></td></tr>
</table>

6.3.3 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			☑ K5 ☑ K2
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*。

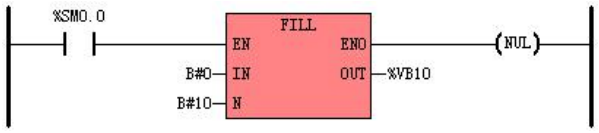
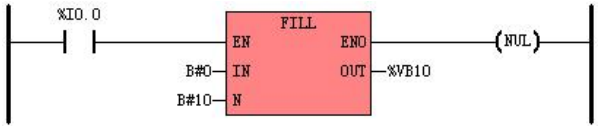
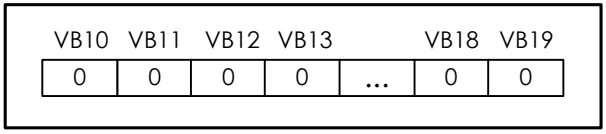
注意， *OUT* 为一个可变长度的内存块的起始地址，整个内存块都不可以落在非法内存区域，否则结果不可预期。

- 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。

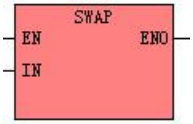
注意，OUT 参数为一个可变长度的块内存参数，整个块内存都不可以落在非法内存区域，否则结果不可预期。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>SM0.0 恒为 1，因此 FILL 指令总是执行：VB10 至 VB19 总计 10 个变量均被赋值为 B#0。</td></tr>
<tr><td> 
</td><td>I0.0 为 0：不执行 FILL 指令。
I0.0 为 1：VB10 至 VB19 总计 10 个变量均被赋值为 B#0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
FILL B#0, %VB10, B#10 (* VB10 至 VB19 总计 10 个变量均被赋值为 B#0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 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 *)</td></tr>
<tr><td>结果</td><td colspan="2">若 FILL 被执行，则结果如下：
 
</td></tr>
</table>

6.3.4 SWAP（交换）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SWAP			☑ K5 ☑ K2
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

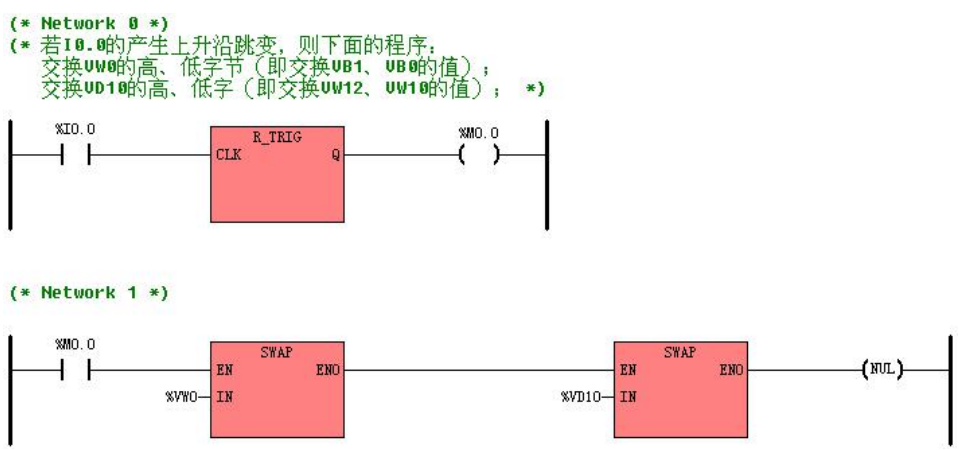
若参数 *IN*是 WORD 型，则该指令用于交换 *IN*的高字节与低字节；
若参数 *IN*是 DWORD 型，则该指令用于交换 *IN*的高字与低字。

- 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

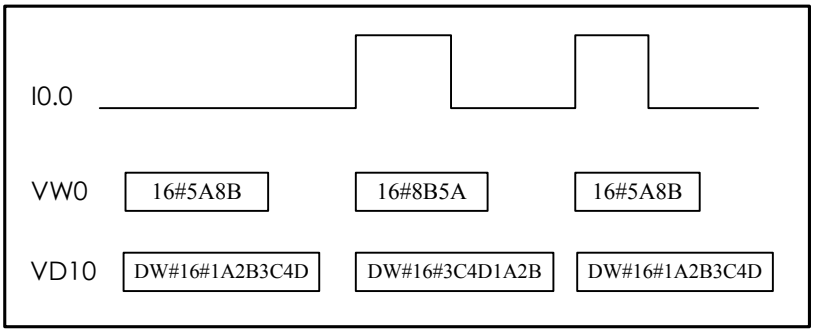


IL

```
(* Network 0 *)
LD      %I0.0
R_TRIG                  (* 若 I0.0 的产生上升沿跳变 *)
SWAP    %VW0            (* 则交换 VW0 的高、低字节 *)
SWAP    %VD10           (* 则交换 VD10 的高、低字 *)
```

结果

假定 VW0 的初始值是 W#16#5A8B，VD10 的初始值是 DW#16#1A2B3C4D。

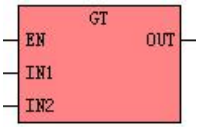


6.4 比较指令

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。若输入参数为实数时，则实数精度的定义见 3.2 数据类型 和 3.4 常量 。

6.4.1 GT（大于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	GT			☑ K5 ☑ K2
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。

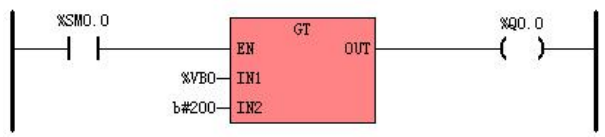
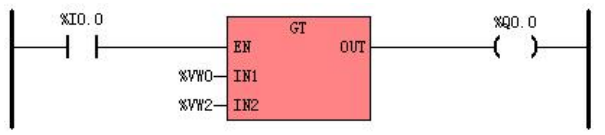
注意， 输入参数为实数时， 实数精度的定义见 3.2 数据类型 3.4 常量

- 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

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 GT 总是执行：若 VB0 的值大于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 GT，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值大于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
GT %VB0, b#200 (* 若 VB0 大于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
GT %VW0, %VW2 (* 若 CR 为 1：若 VW0 大于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *)
(* 若 CR 为 0：不进行 GT 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.2 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			☑ K5 ☑ K2
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

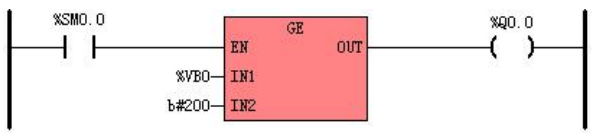
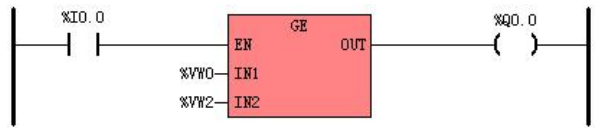
注意， 输入参数为实数时， 实数精度的定义见 3.2 数据类型 3.4 常量

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

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 GE 总是执行：若 VB0 的值大于等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 GE，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值大于等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
GE %VB0, b#200 (* 若 VB0 大于等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
GE %VW0, %VW2 (* 若 CR 为 1：则若 VW0 大于等于 VW2，则 CR 被置为 1，
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不进行 GE 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.3 EQ（等于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	EQ			☑ K5 ☑ K2
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

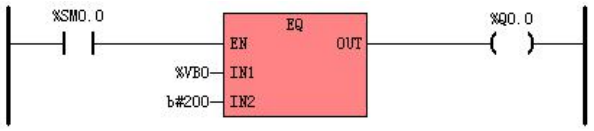
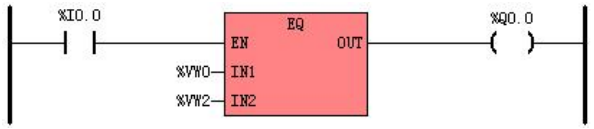
注意， 输入参数为实数时， 实数精度的定义见 3.2 数据类型 3.4 常量

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。

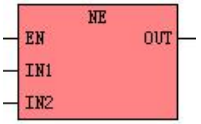
- IL
  如果 CR 值为 1，该指令被执行：若 *IN1* 等于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 EQ 总是执行：若 VB0 的值等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 EQ，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
EQ %VB0, b#200 (* 若 VB0 等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
EQ %VW0, %VW2 (* 若 CR 为 1：则若 VW0 等于 VW2，则 CR 被置为 1，否则 CR 被置为 0 *)
(* 若 CR 为 0：则不进行 EQ 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.4 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			☑ K5 ☑ K2
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

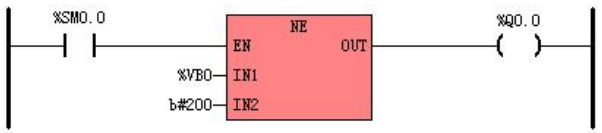
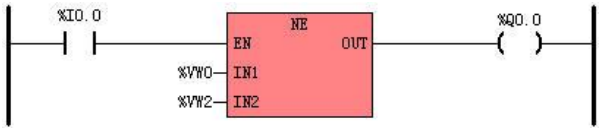
注意， 输入参数为实数时， 实数精度的定义见 3.2 数据类型 3.4 常量

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。

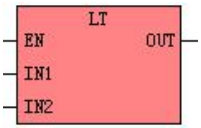
- IL
  若 CR 值为 1，该指令被执行：若 *IN1* 不等于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；
  若 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 NE 总是执行：若 VB0 的值不等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 NE，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值不等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
NE %VB0, b#200 (* 若 VB0 不等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
NE %VW0, %VW2 (* 若 CR 为 1：则若 VW0 不等于 VW2，则 CR 被置为 1 *)
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不进行 NE 比较，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.5 LT（小于）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	LT			☑ K5 ☑ K2
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

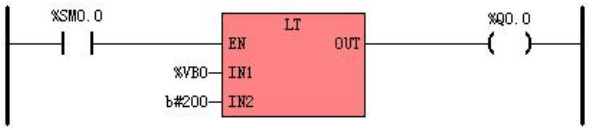
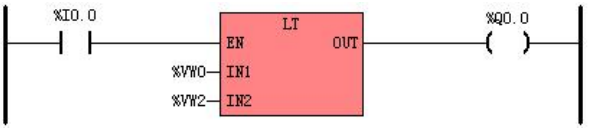
注意， 输入参数为实数时， 实数精度的定义见 3.2 数据类型 3.4 常量

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。

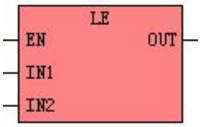
- IL
  如果 CR 值为 1，该指令被执行：若 *IN1* 小于 *IN2*，则将 CR 置为 1，否则将 CR 置为 0；
  如果 CR 值为 0，该指令不执行，CR 值保持为 0。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 LT 总是执行：若 VB0 的值小于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 LT，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值小于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
LT %VB0, b#200 (* 若 VB0 小于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td></td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
LT %VW0, %VW2 (* 若 CR 为 1：则若 VW0 小于 VW2，则 CR 被置为 1 *)
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不执行 LT，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.4.6 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			☑ K5 ☑ K2
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

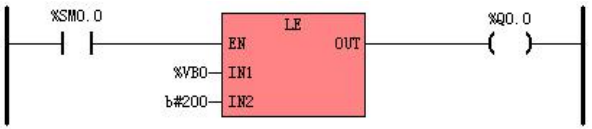
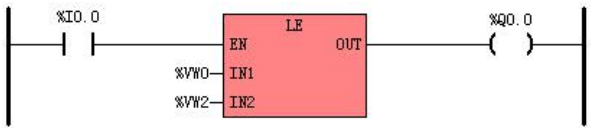
注意， 输入参数为实数时， 实数精度的定义见 3.2 数据类型 3.4 常量

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。

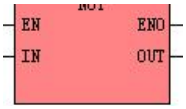
➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>由于 SM0.0 固定为 1，因此 LE 总是执行：若 VB0 的值小于等于 B#200，则 Q0.0 被置为 1，否则 Q0.0 就置为 0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 LE，Q0.0 被置为 0。
若 I0.0 为 1：若 VW0 的值小于等于 VW2 的值，则 Q0.0 被置为 1，否则 Q0.0 被置为 0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
LE %VB0, b#200 (* 若 VB0 小于等于 B#200，则 CR 被置为 1，否则 CR 被置为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
LE %VW0, %VW2 (* 若 CR 为 1：则若 VW0 小于等于 VW2，则 CR 被置为 1 *)
(* 否则 CR 被置为 0 *)
(* 若 CR 为 0：则不执行 LE，CR 保持为 0 *)
ST %Q0.0 (* 将 CR 值赋给 Q0.0 *)</td></tr>
</table>

6.5 逻辑运算

6.5.1 NOT（按位取反）

➢ 指令及其操作数说明

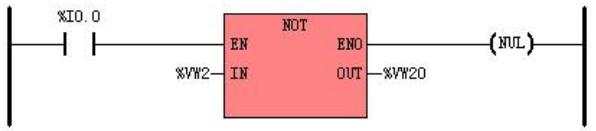
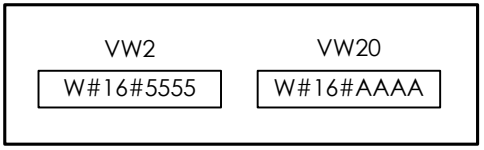
	名称	指令格式	影响 CR 值	
LD	NOT			☑ K5 ☑ K2
IL	NOT	NOT *OUT*	U	

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*。

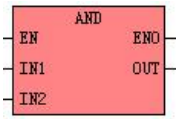
- IL
  如果 CR 值为 1，则该指令被执行：将 *OUT* 的每一个二进制位都取反，结果仍旧赋给 *OUT*。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 NOT 指令。
若 I0.0 为 1：将 VW2 的值按位取反，并将结果赋给 VW20。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
NOT %VW20 (* 若 CR 为 1：将 VW20 按位取反，结果仍放于 VW20 中 *)
(* 若 CR 为 0：不执行 NOT 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 NOT 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.2 AND（按位与）

➢ 指令及其操作数说明

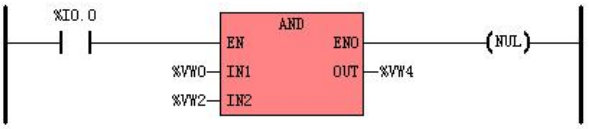
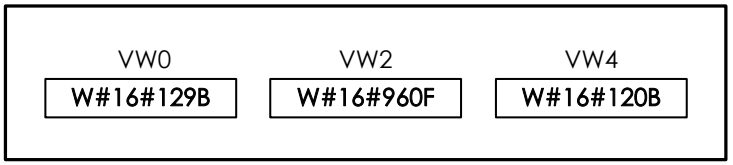
	名称	指令格式	影响 CR 值	
LD	AND			☑ K5 ☑ K2
IL	AND	AND IN1, OUT	U	

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*。

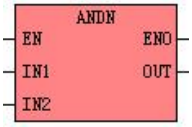
- IL
  参数 *IN1*、*OUT* 的数据类型必须一致。
  若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“与”运算后，将结果赋给 *OUT*。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 AND 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“与”运算，并将结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
AND %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按位相“与”，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 AND 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 AND 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.3 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			☑ K5 ☑ K2
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*。

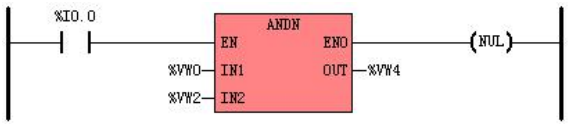
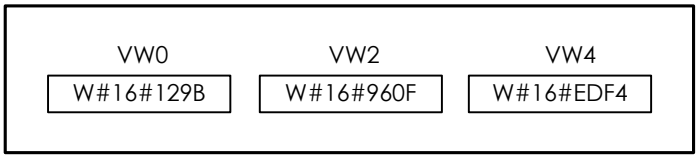
- IL

  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*。

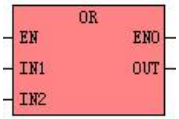
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 ANDN 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“与”运算并取反，最终的结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
ANDN %VW0, %VW2 (* 若 CR 为 1:将 VW0 和 VW2 的值按位相与并取反,结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 ANDN 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 ANDN 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.4 OR（按位或）

➢ 指令及其操作数说明

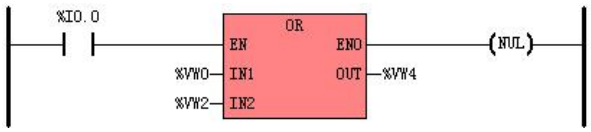
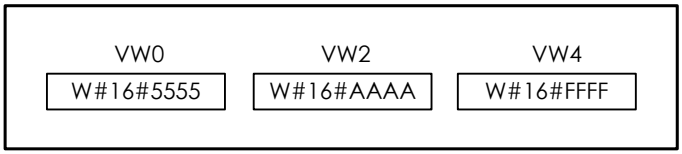
	名称	指令格式	影响 CR 值	
LD	OR			☑ K5 ☑ K2
IL	OR	OR IN1, OUT	U	

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*。

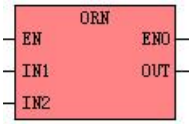
- IL
  参数 *IN1*、*OUT* 的数据类型必须一致。
  若 CR 值为 1，则该指令被执行：将 *IN1* 和 *OUT* 按二进制位进行“或”运算后，将结果赋给 *OUT*。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 OR 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“或”运算，并将结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
OR %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按位相或，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 OR 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 OR 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.5.5 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			☑ K5 ☑ K2
IL	ORN	ORN *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*。

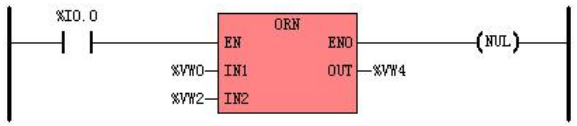
- IL

  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*。

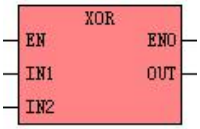
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 ORN 指令。
若 I0.0 为 1:将 VW0 和 VW2 的值按位进行“或”运算并取反，最终的结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
ORN %VW0, %VW2 (* 若 CR 为 1:将 VW0 和 VW2 的值按位相或并取反，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 ORN 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 ORN 指令被执行，则结果举例如下：
VW0 W#16#129B
VW2 W#16#960F
VW4 W#16#6960</td></tr>
</table>

6.5.6 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			☑ K5 ☑ K2
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*。

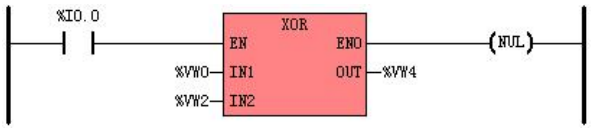
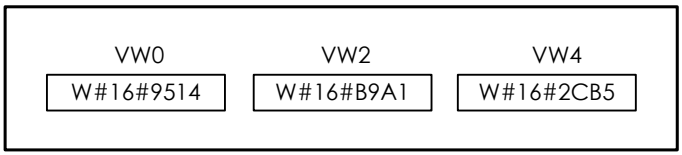
- IL

  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。

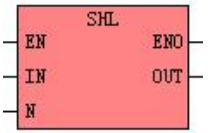
➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 XOR 指令。
若 I0.0 为 1：将 VW0 和 VW2 的值按位进行“异或”运算，并将结果赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
XOR %VW0, %VW2 (* 若 CR 为 1：将 VW0 和 VW2 的值按位相“异或”，结果仍放于 VW2 中 *)
(* 若 CR 为 0：不执行 XOR 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">参照上面的 LD 例子，若 XOR 指令被执行，则结果举例如下：
 
</td></tr>
</table>

6.6 移位指令

6.6.1 SHL（左移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SHL			☑ K5 ☑ K2
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

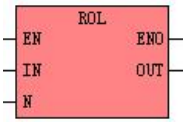
<table>
<tr><td rowspan="2">LD</td><td> 
</td></tr>
<tr><td>在 M0.0 的每个上升沿，都执行一次 SHL 指令，将 QB0 值的全部二进制位左移 1 位，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td>LD %M0.0
R_TRIG (* 取 M0.0 的上升沿，并将结果作为 CR 值 *)
SHL %QB0, B#1 (* 若 CR 为 1：将 QB0 值的全部二进制位左移 1 位，结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td>结果</td><td>若 SHL 指令被执行，则结果举例如下：

QB0初值： B#2#10000001

第一次移位后 第二次移位后 第三次移位后 第四次移位后
QB0值： B#2#00000010 B#2#00000100 B#2#00001000 B#2#00010000</td></tr>
</table>

6.6.2 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			☑ K5 ☑ K2
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

<table>
<tr><td>LD</td><td> 
</td></tr>
<tr><td></td><td>在 M0.0 的每个上升沿，都执行一次 ROL 指令，将 QB0 值的全部二进制位每次左移 1 位，移出的高位依次补到低位位置，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td>LD %M0.1
R_TRIG (* 取 M0.0 的上升沿，并将结果作为 CR 值 *)
ROL %QB0, B#1 (* 若 CR 为 1:将 QB0 值的全部二进制位循环左移 1 位,结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td>结果</td><td>若 ROL 指令被执行，则结果举例如下：

QB0初值： B#2#10100001

第一次移位后 第二次移位后 第三次移位后 第四次移位后
QB0值： B#2#01000011 B#2#10000110 B#2#00001101 B#2#00011010</td></tr>
</table>

6.6.3 SHR（右移）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SHR			☑ K5 ☑ K2
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

<table>
<tr><td rowspan="2">LD</td><td> 
</td></tr>
<tr><td>在 M0.0 的每个上升沿，都执行一次 SHR 指令，将 QB0 值的全部二进制位右移 1 位，并将结果仍赋给 QB0。</td></tr>
<tr><td>IL</td><td>LD %M0.0
R_TRIG (* 取 M0.0 的上升沿，并将结果作为 CR 值 *)
SHR %QB0, B#1 (* 若 CR 为 1：将 QB0 值的全部二进制位右移 1 位，结果仍放于 QB0 中 *)
(* 若 CR 为 0：不执行 SHL 指令 *)</td></tr>
<tr><td>结果</td><td>若 SHR 指令被执行，则结果举例如下：

QB0初值： B#2#10000001

第一次移位后 第二次移位后 第三次移位后 第四次移位后
QB0值： B#2#01000000 B#2#00100000 B#2#00010000 B#2#00001000</td></tr>
</table>

6.6.4 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			☑ K5 ☑ K2
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

LD

在 M0.0 的每个上升沿，都执行一次 ROR 指令，将 QB0 值的全部二进制位每次右移 1 位，移出的低位被依次补到高位位置，并将结果仍赋给 QB0。

IL

```
LD      %M0.1
R_TRIG              (* 取 M0.0 的上升沿，并将结果作为 CR 值  *)
ROR     %QB0, B#1   (* 若 CR 为 1:将 QB0 值的全部二进制位循环右移 1 位,结果仍放于 QB0 中 *)
                    (* 若 CR 为 0：不执行 SHL 指令 *)
```

结果

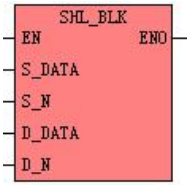
若 ROR 指令被执行，则结果举例如下：

QB0初值： **B#2#10100001**

	第一次移位后	第二次移位后	第三次移位后	第四次移位后
QB0值：	**B#2#11010000**	**B#2#01101000**	**B#2#00110100**	**B#2#00011010**

6.6.5 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			☑ K5 ☑ K2
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

S_N, D_N 参数最大值为 1024， 当输入大于 1024 时， 取 1024。 当 S_N 大于 D_N 时， 取 S_N = D_N。 S_N, D_N 都必须大于 0。

注意， S_DATA, D_DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

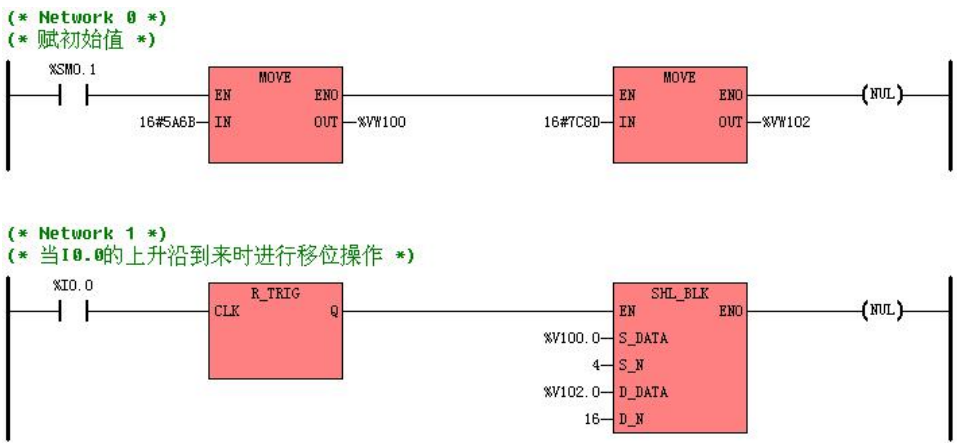
该指令执行时，将从 *D_DATA* 开始的连续 *D_N* 个二进制位向左移动 *S_N* 位，移出的高位被丢弃，同时将从 *S_DATA* 开始的连续 *S_N* 个二进制位补入 *D_DATA* 的最右端。

- 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

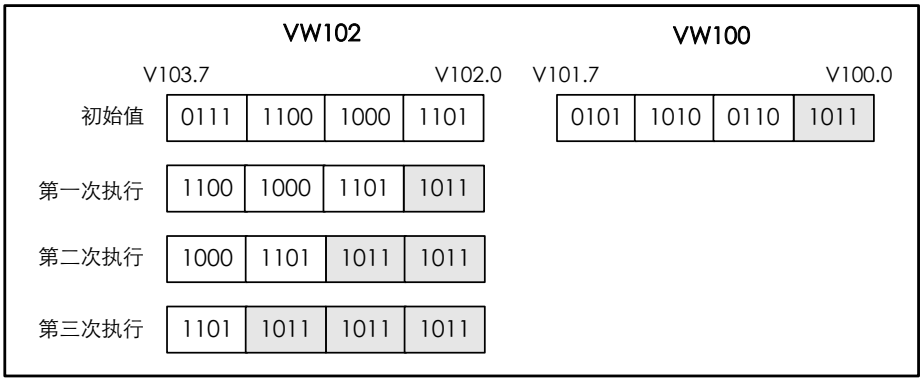
LD



IL

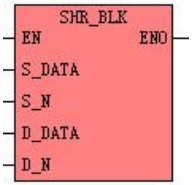
```
(* Network 0 *)
(*赋初始值*)
LD      %SM0.1
MOVE    16#5A6B, %VW100
MOVE    16#7C8D, %VW102
(* Network 1 *)
(*当 I0.0 的上升沿到来时进行移位操作*)
LD      %I0.0
R_TRIG
SHL_BLK %V100.0, 4, %V102.0, 16
```

结果



6.6.6 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			☑ K5 ☑ K2
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

S_N, D_N 参数最大值为 1024， 当输入大于 1024 时， 取 1024。 当 S_N 大于 D_N 时， 取 S_N = D_N。 S_N, D_N 都必须大于 0。

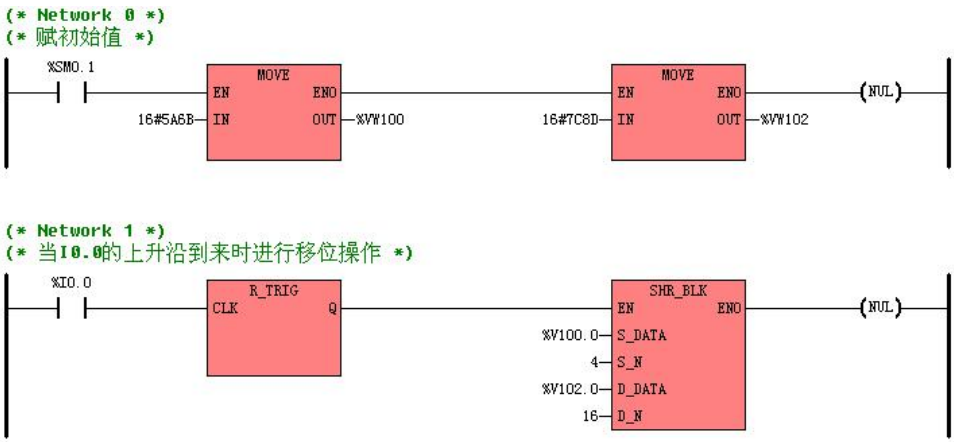
注意， S_DATA, D_DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

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

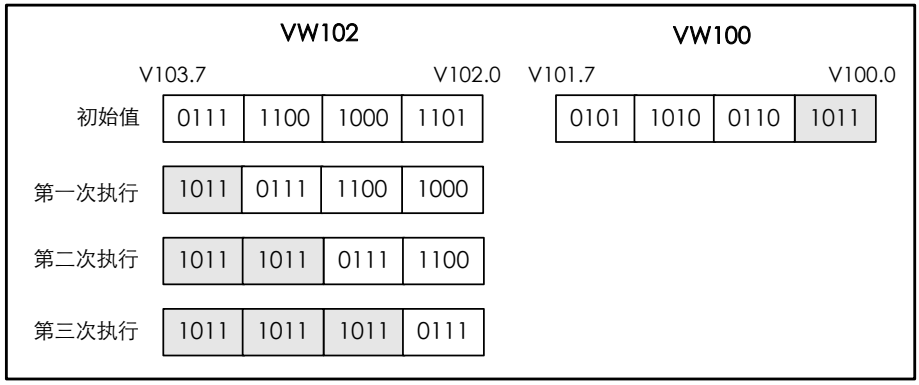
LD



IL

```
(* Network 0 *)
(*赋初始值*)
LD      %SM0.1
MOVE    16#5A6B, %VW100
MOVE    16#7C8D, %VW102
(* Network 1 *)
(*当 I0.0 的上升沿到来时进行移位操作*)
LD      %I0.0
R_TRIG
SHR_BLK %V100.0, 4, %V102.0, 16
```

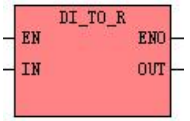
结果



6.7 类型转换

6.7.1 DI_TO_R（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	DI_TO_R			☑ K5 ☑ K2
IL	DI_TO_R	DI_TO_R *IN, OUT*	U	

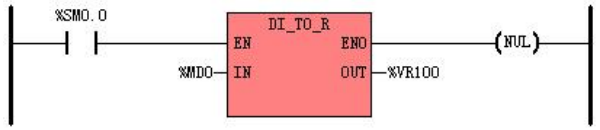
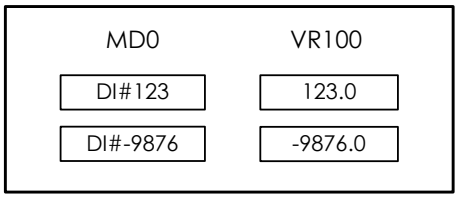
参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
OUT	输出	REAL	V、L

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。

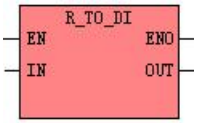
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DI_TO_R 指令总是执行：将 MD0 的值转换为实数并赋给 VR100。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
DI_TO_R %MD0, %VR100 (* 将 MD0 的值转换为实数并赋给 VR100 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.2 R_TO_DI（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	R_TO_DI			☑ K5 ☑ K2
IL	R_TO_DI	R_TO_DI *IN, OUT*	U	

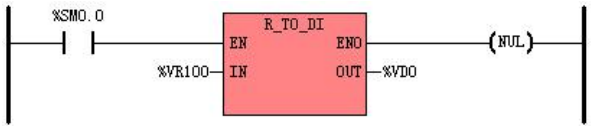
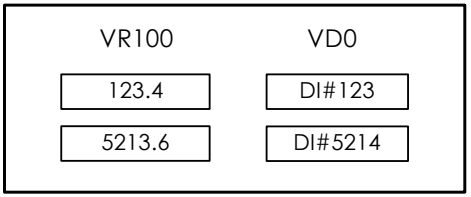
参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L 、常量
OUT	输出	DINT	M、V、L、SM

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。

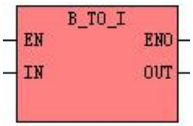
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 R_TO_DI 指令总是执行：将实数 VR100 转换为双整数并赋给 VD0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
R_TO_DI %VR100, %VD0 (* 将实数 VR100 转换为双整数并赋给 VD0 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.3 B_TO_I（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	B_TO_I			☑ K5 ☑ K2
IL	B_TO_I	B_TO_I *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

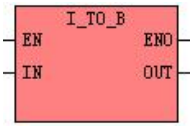
该指令将输入的字节型数据 *IN* 转换为整数并赋给 *OUT*。

- 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。

6.7.4 I_TO_B（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	I_TO_B			☑ K5 ☑ K2
IL	I_TO_B	I_TO_B *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

注意， I_TO_B 的输入参数超出输出数据类型的表达范围时， 会触发错误，输出值取 C 语言的强制类型转换时的值。

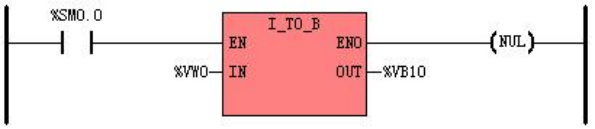
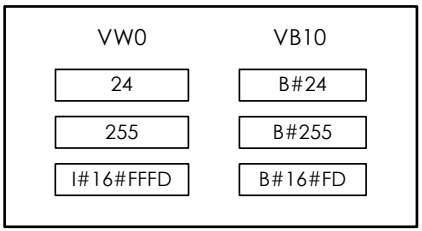
该指令将整数 *IN*转换为字节型数值并赋给 *OUT*。

字节型的取值范围为 B#0---B#255，若参数 *IN*的值超过了这个范围，则转换结果为其低字节值，同时 K5 会记录下溢出错误。

- LD
  如果 *EN*为 1，则该指令被执行。

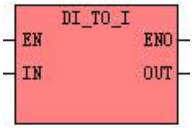
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 I_TO_B 指令总是执行：将 VW0 的低字节赋给 VB10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
I_TO_B %VW0, %VB10 (* 将 VW0 整数值转换为字节型值并赋给 VB10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.5 DI_TO_I（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	DI_TO_I			☑ K5 ☑ K2
IL	DI_TO_I	DI_TO_I *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

注意， DI_TO_I 的输入参数超出输出数据类型的表达范围时， 会触发错误，输出值取 C 语言的强制类型转换时的值。

该指令将双整数 *IN*转换为整数并赋给 *OUT*。

整数的取值范围为 -32768 --- 32767，若参数 *IN*的值超过了这个范围，则转换结果为其低字值，同时 K5 会记录下溢出错误。

- LD

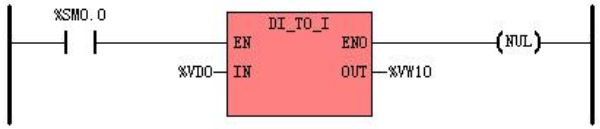
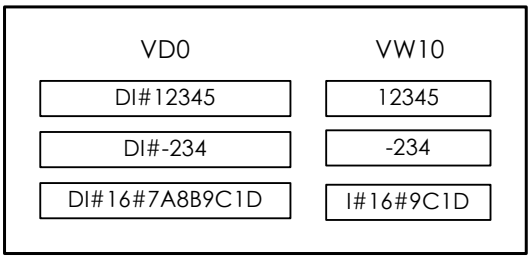
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

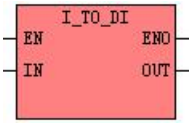
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DI_TO_I 指令总是执行：将 VD0 的低字赋给 VW10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
DI_TO_I %VD0, %VW10 (* 将 VD0 双整数值转换为整数并赋给 VW10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.6 I_TO_DI（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	I_TO_DI			☑ K5 ☑ K2
IL	I_TO_DI	I_TO_DI *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

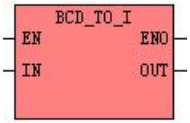
该指令将输入的整数 *IN*转换为双整数并赋给 *OUT*。

- 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。

6.7.7 BCD_TO_I（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	BCD_TO_I			☑ K5 ☑ K2
IL	BCD _TO_I	BCD _TO_I *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

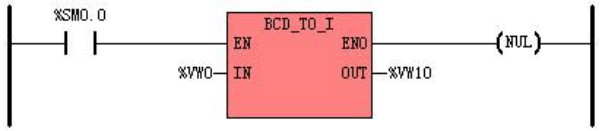
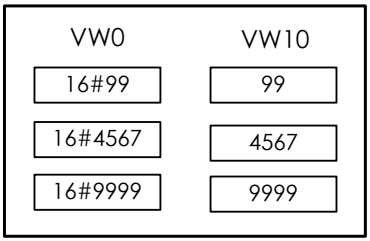
该指令将输入的 BCD 码 *IN*转换为整数并赋给 *OUT*。

注意：BCD 码采用 8421 码。*IN*的有效范围是 0∽9999 BCD。

- LD
  如果 *EN*为 1，则该指令被执行。

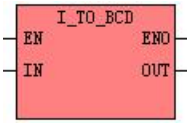
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 BCD_TO_I 指令总是执行：将 VW0 的值由 BCD 转换为整数并赋给 VW10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
BCD_TO_I %VW0, %VW10 (* 将 VW0 的值由 BCD 转换为整数并赋给 VW10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.8 I_TO_BCD（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	I_TO_BCD			☑ K5 ☑ K2
IL	I_TO_BCD	I _TO_BCD *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

注意， I_TO_BCD 的输入参数超出范围（见下）时， 会触发错误，输入小于最小值时， 取最小值， 输入大于最大值时取最大值。

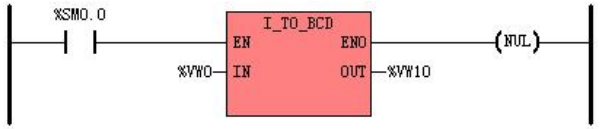
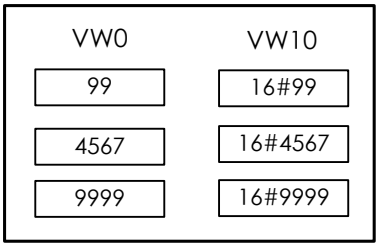
注意：BCD 码采用 8421 码。*IN*的有效范围是 0∽9999，若超过这个范围，则 K5 会记录下溢出错误，同时按如下方式处理：若 *IN* ≤ 0，则转换结果保持为 0，若 *IN* ≥ 9999，则转换结果保持为 9999 BCD。

该指令将输入的整数 *IN*转换为 BCD 码并赋给 *OUT*。

- LD
  如果 *EN*为 1，则该指令被执行。

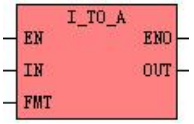
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 I_TO_BCD 指令总是执行：将 VW0 的值由整数转换为 BCD 并赋给 VW10。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
I_TO_BCD %VW0, %VW10 (* 将 VW0 的值由整数转换为 BCD 并赋给 VW10 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.9 I_TO_A（整型转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	I_TO_A			☑ K5 ☑ K2
IL	I_TO_A	I_TO_A IN, OUT, FMT	U	

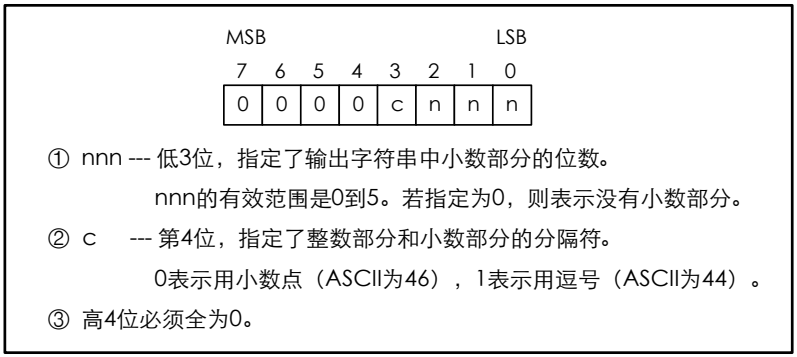
参数	输入/输出	数据类型	允许的内存区
IN	输入	INT	I、Q、M、V、L、SM、AI、AQ、T、C、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

注意，OUT 参数为一个可变长度的块内存参数，整个块内存都不可以落在非法内存区域，否则结果不可预期。

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：

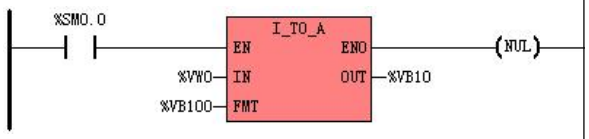
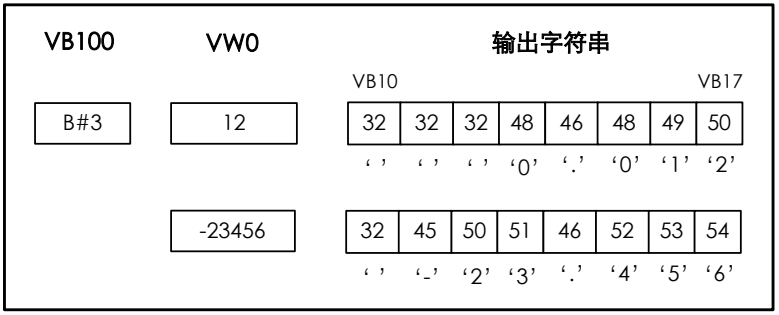


- LD

  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><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 I_TO_A 指令总是执行：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。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
I_TO_A %VW0, %VB10, %VB100</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.10 DI_TO_A（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	DI_TO_A			☑ K5 ☑ K2
IL	DI_TO_A	DI_TO_A IN, OUT, FMT	U	

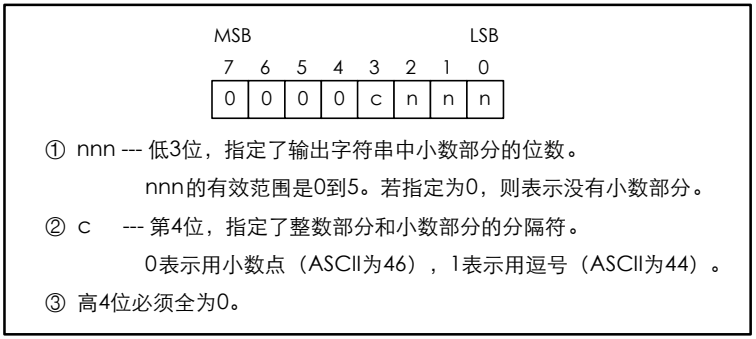
参数	输入/输出	数据类型	允许的内存区
IN	输入	DINT	I、Q、M、V、L、SM、HC、常量
FMT	输入	BYTE	I、Q、M、V、L、SM
OUT	输出	BYTE	Q、M、V、L、SM

注意，OUT 参数为一个可变长度的块内存参数，整个块内存都不可以落在非法内存区域，否则结果不可预期。

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：

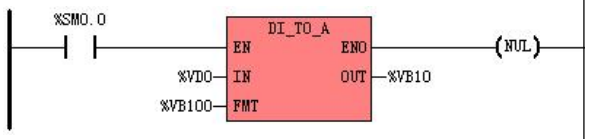
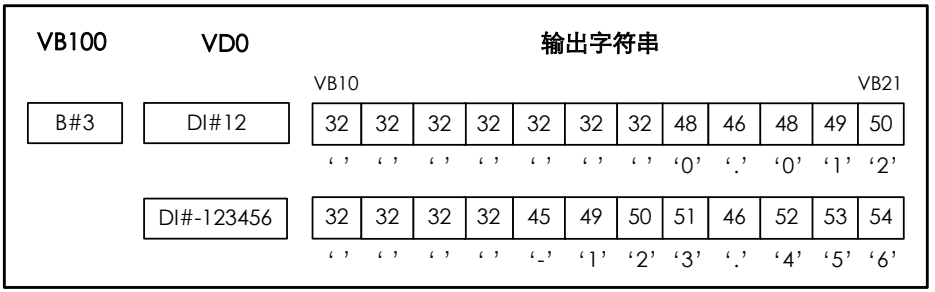


- LD

如果 *EN* 为 1，则该指令被执行。

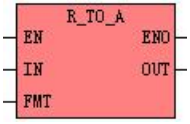
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DI_TO_A 指令总是执行：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>IL</td><td colspan="2">LD %SM0.0
DI_TO_A %VD0, %VB10, %VB100</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：

</td></tr>
</table>

6.7.11 R_TO_A（实型转 ASCII）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	R_TO_A			☑ K5 ☑ K2
IL	R_TO_A	R_TO_A 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

注意，OUT 参数为一个可变长度的块内存参数，整个块内存都不可以落在非法内存区域，否则结果不可预期。

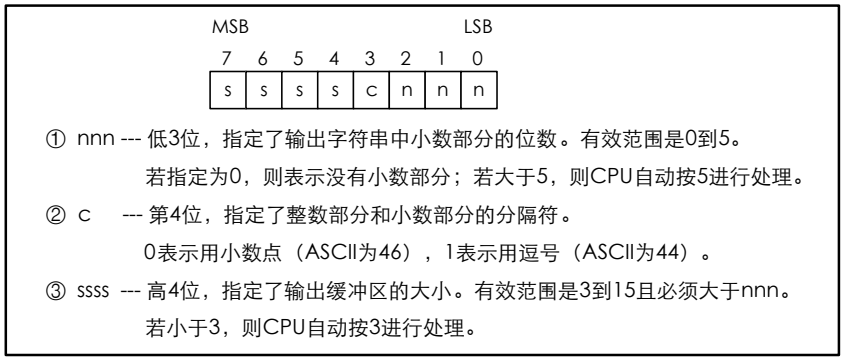
注意，此指令耗时较长，此指令在很多个同时运行时，可能会触发看门狗，可以加入 WDR 指令来延长看门狗的触发时间。

注意，R_TO_A 的输入超出范围或者输出长度超出本指令输出格式中允许的范围（见下）时，会触发错误，整个输出缓冲被填写成空格。输入值的范围，小于(4294960000)且大于(-2147480000)

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：



该指令使用时需要注意：

a) 输入 *IN*的允许范围是[-2147480000.0, 4294960000.0],若 *IN*超出这个范围或者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。

b) 输出缓冲区不能超出有效的内存范围，否则执行结果不可预期。

- LD

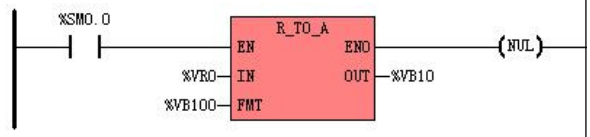
  如果 *EN*为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

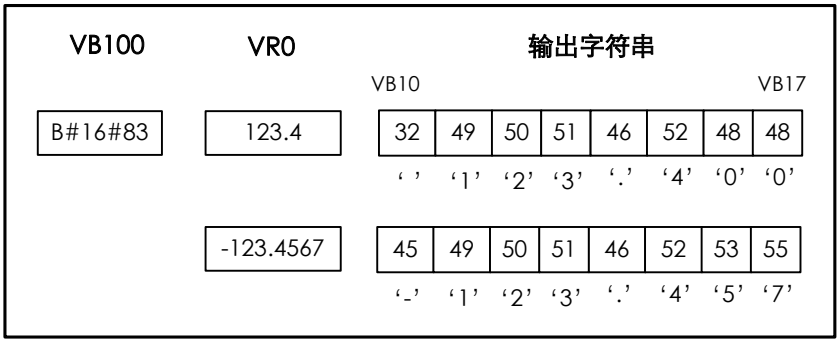
  该指令的执行不影响 CR 值。

➢ 指令使用举例

LD		SM0.0 恒为 1，因此 R_TO_A 指令总是执行：将 VR0 的值转换为字符串并格式化输出到 VB10 开始的缓冲区中。
IL	LD %SM0.0 R_TO_A %VR0, %VB10, %VB100	

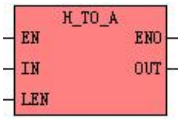
结果

结果举例如下：


VB100
VR0
输出字符串
VB10
VB17
B#16#83
123.4
32 49 50 51 46 52 48 48
‘ ’ ‘1’ ‘2’ ‘3’ ‘.’ ‘4’ ‘0’ ‘0’
-123.4567
45 49 50 51 46 52 53 55
‘-’ ‘1’ ‘2’ ‘3’ ‘.’ ‘4’ ‘5’ ‘7’

6.7.12 H_TO_A（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	H_TO_A			☑ K5 ☑ K2
IL	H_TO_A	H_TO_A 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

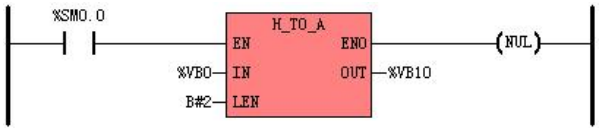
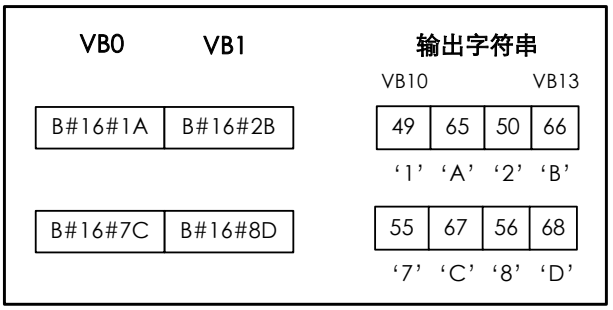
注意， IN，OUT 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

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。

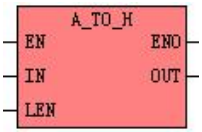
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 H_TO_A 指令总是执行：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。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
H_TO_A %VB0, %VB10, B#2</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.7.13 A_TO_H（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	A_TO_H			☑ K5 ☑ K2
IL	A_TO_H	A_TO_H 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

注意， IN，OUT 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

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），B#16#61∽B#16#66（表示字符 a∽f）。

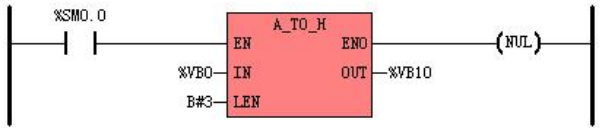
- LD

  如果 *EN*为 1，则该指令被执行。

- IL

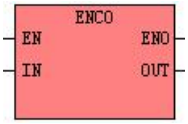
  如果 CR 值为 1，则该指令被执行。该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 A_TO_H 指令总是执行：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>IL</td><td colspan="2">LD %SM0.0
A_TO_H %VB0, %VB10, B#3</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
VB0 VB1 VB2 VB10 VB11
51 56 54 B#16#38 B#16#6x
‘3’ ‘8’ ‘6’
55 65 49 B#16#7A B#16#1x
‘7’ ‘A’ ‘1’
注：上面的x表示这半个字节（4位）中内容保持原有值不变。</td></tr>
</table>

6.7.14 ENCO（编码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	ENCO			☑ K5 ☑ K28
IL	ENCO	ENCO IN, OUT	U	

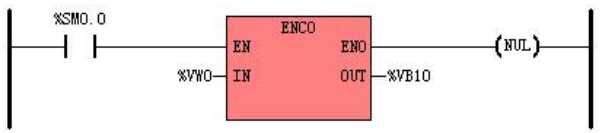
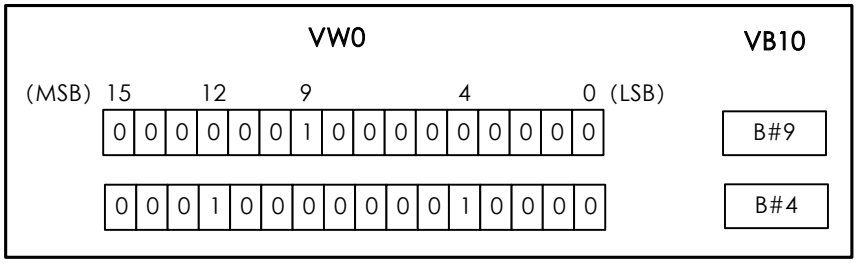
参数	输入/输出	数据类型	允许的内存区
IN	输入	WORD	I、Q、M、V、L、SM、常量
OUT	输出	BYTE	Q、M、V、L、SM

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。

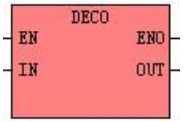
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 ENCO 指令总是执行：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。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
ENCO %VW0, %VB10</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下图，图中 VW0 的值采用了二进制来表示。
 
</td></tr>
</table>

6.7.15 DECO（解码）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	DECO			☑ K5 ☑ K2
IL	DECO	DECO IN, OUT	U	

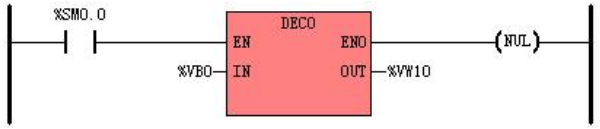
参数	输入/输出	数据类型	允许的内存区
IN	输入	BYTE	I、Q、M、V、L、SM、常量
OUT	输出	WORD	Q、M、V、L、SM

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。

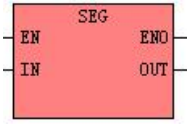
- IL
  如果 CR 值为 1，则该指令被执行。
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 DECO 指令总是执行：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。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0
DECO %VB0, %VW10</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下图，图中 VW10 的值采用了二进制来表示。
 
</td></tr>
</table>

6.7.16 SEG（七段码显示）

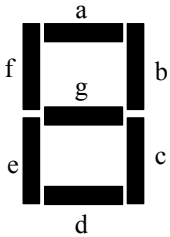
➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SEG			☑ K5 ☑ K2
IL	SEG	SEG 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	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 (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



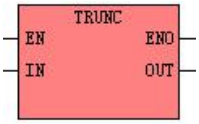
IN (低四位)	显示	*OUT* (- g f e d c b a)
8	8	0 1 1 1 1 1 1 1
9	9	0 1 1 0 0 1 1 1
A	A	0 1 1 1 0 1 1 1
B	B	0 1 1 1 1 1 0 0
C	C	0 0 1 1 1 0 0 1
D	D	0 1 0 1 1 1 1 0
E	E	0 1 1 1 1 0 0 1
F	F	0 1 1 1 0 0 0 1

- 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。

6.7.17 TRUNC（取整）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	TRUNC			☑ K5 ☑ K2
IL	TRUNC	TRUNC *IN, OUT*	U	

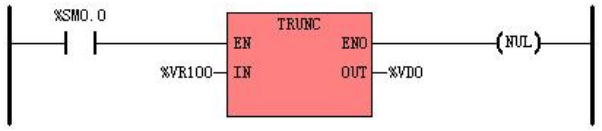
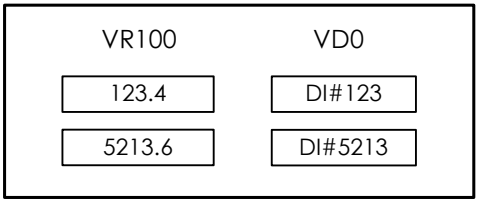
参数	输入/输出	数据类型	允许的内存区
IN	输入	REAL	V、L 、常量
OUT	输出	DINT	M、V、L、SM

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。

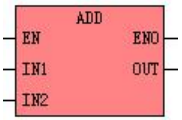
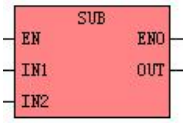
➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 TRUNC 指令总是执行：将实数 VR100 转换为双整数（小数部分被舍弃）并赋给 VD0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
TRUNC %VR100, %VD0 (* 将实数 VR100 舍弃小数部分后转换为双整数并赋给 VD0 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.8 数学运算

6.8.1 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			☑ K5 ☑ K2
	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。

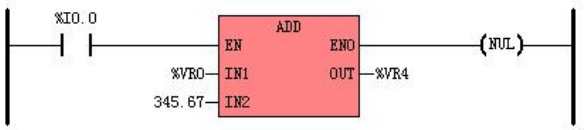
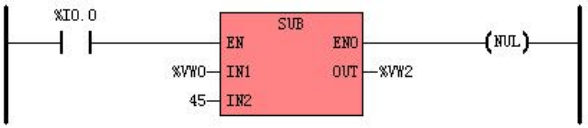
  若 *EN* 值为 1，则指令被执行。其中，*ADD* 指令的功能是：*OUT*＝*IN1*＋*IN2*；*SUB* 指令的功能是：*OUT*＝*IN1*－*IN2*。

- IL

  参数 *IN1*、*OUT* 的数据类型必须一致。

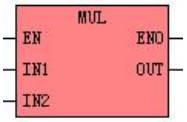
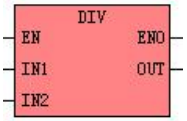
  若 CR 值为 1，则指令被执行。其中，*ADD* 指令的功能是：*OUT*= *OUT* + *IN1*；*SUB* 指令的功能是：*OUT*= *OUT*－*IN1*。ADD、SUB 指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>若 I0.0 为 0：
不执行 ADD 指令。
若 I0.0 为 1：
VR4 = VR4 + 345.67。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：
不执行 SUB 指令。
若 I0.0 为 1：
VW2 = VW2 - 45。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
ADD 345.67, %VR4 (* 若 CR 为 1：VR4 = VR4 + 345.67 *)
(* 若 CR 为 0：不执行 ADD 指令 *)</td></tr>
<tr><td></td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
SUB 45, %VW2 (* 若 CR 为 1：VW2 = VW2－45 *)
(* 若 CR 为 0：不执行 SUB 指令 *)</td></tr>
</table>

6.8.2 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			☑ K5 ☑ K2
	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

注意， DIV 的除数为 0 时， 会触发错误， 输出值维持上一次的值。 实数 0 的定义见 3.2 数据类型 3.4 常量

- 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*。

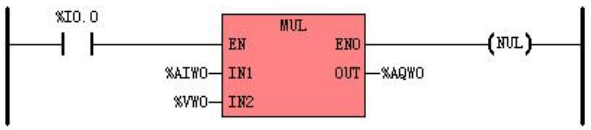
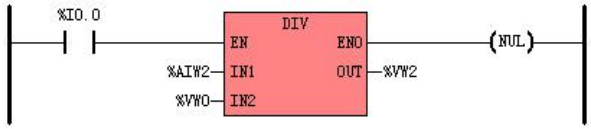
- IL

  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。

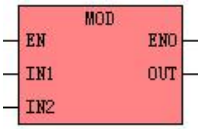
注意：若 *DIV* 指令的除数为 0，则指令的输出值维持上一次的结果，同时 K5 记录下“被 0 除”错误。

➢ 指令使用举例

<table>
<tr><td rowspan="2">LD</td><td> 
</td><td>若 I0.0 为 0：不执行 MUL 指令。
若 I0.0 为 1：将 AIW0 与 VW0 相乘，并将结果赋给 AQW0。</td></tr>
<tr><td> 
</td><td>若 I0.0 为 0：不执行 DIV 指令。
若 I0.0 为 1：将 AIW2 除以 VW0，并将结果赋给 VW2。</td></tr>
<tr><td rowspan="2">IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
MUL %AIW0, %VW0 (* 若 CR 为 1：VW0 = VW0×AIW0 *)
(* 若 CR 为 0：不执行 MUL 指令 *)</td></tr>
<tr><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
DIV %AIW2, %VW0 (* 若 CR 为 1：VW0 = VW0÷AIW2 *)
(* 若 CR 为 0：不执行 DIV 指令 *)</td></tr>
</table>

6.8.3 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			☑ K5 ☑ K2
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

注意，MOD 的除数为 0 时， 会触发错误， 输出值维持上一次的值。

- 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

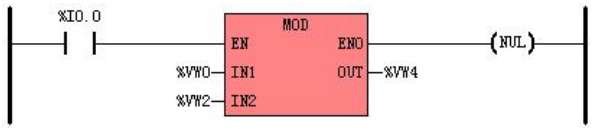
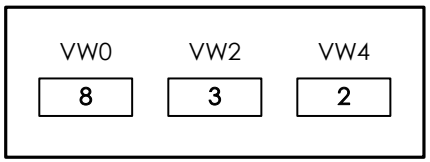
  参数 *IN1*、*OUT* 的数据类型必须一致。

  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。

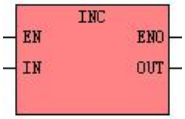
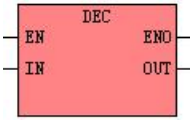
注意：若 *MOD* 指令的除数为 0，则指令的输出值维持上一次的结果，同时 K5 记录下“被 0 除”错误。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>若 I0.0 为 0：不执行 MOD 指令。
若 I0.0 为 1：将 VW0 除以 VW2，得到的余数赋给 VW4。</td></tr>
<tr><td>IL</td><td colspan="2">LD %I0.0 (* 建立 CR，其值为 I0.0 的值 *)
MOD %VW0, %VW4 (* 若 CR 为 1：将 VW4 除以 VW0，得到的余数赋给 VW4 *)
(* 若 CR 为 0：不执行 MOD 指令 *)</td></tr>
<tr><td>结果</td><td colspan="2">若上述 LD 例子中 MOD 指令得以执行，则结果举例如下：
 
</td></tr>
</table>

6.8.4 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			☑ K5 ☑ K2
	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。

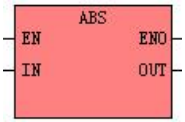
- IL

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。

6.8.5 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			☑ K5 ☑ K2
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

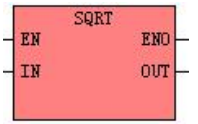
参数 IN、OUT 的数据类型必须一致。
该指令将输入 IN 求绝对值并将结果赋给 OUT，如下式所示：OUT＝|IN|。

- 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。

6.8.6 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			☑ K5 ☑ K2
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

注意，SQRT 的输入为负数时， 会触发错误， 输出值维持上一次的值。 实数 0 的定义见 3.2 数据类型 3.4 常量

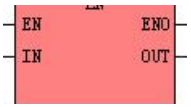
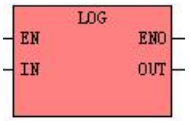
该指令将输入 IN 开平方并将结果赋给 OUT，如下式所示：OUT＝$\sqrt{IN}$ 。
注意：若输入参数 *IN*为负数，则指令的输出值维持上一次的结果，同时 K5 记录下错误。

- 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。

6.8.7 LN（自然对数）、LOG（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			☑ K5 ☑ K2
	LOG			
IL	LN	LN *IN*, *OUT*	U	
	LOG	LOG *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

注意，LN， LOG 的输入为 0 或负数时， 会触发错误， 输出值维持上一次的值。 实数 0 的定义见 3.2 数据类型 3.4 常量

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*)。

LOG 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*)。

注意：若输入参数 *IN* 为 0 或负数，则指令的输出值维持上一次的结果，同时 K5 记录下错误。

- LD

  若 *EN* 值为 1，则指令被执行。

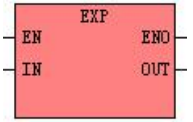
- IL

  如果 CR 值为 1，则指令被执行。

  LN、LOG 指令的执行不影响 CR 值。

6.8.8 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			☑ K5 ☑ K2
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

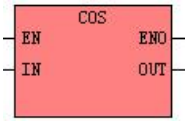
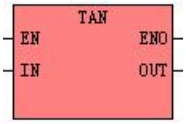
该指令将输入 *IN* 求以 e 为底的指数并将结果赋给 *OUT*，如下式所示：*OUT*＝e^{IN}。

- 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。

6.8.9 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			☑ K5 ☑ K2
	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

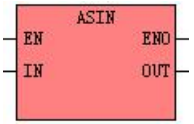
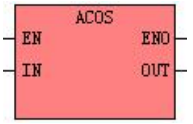
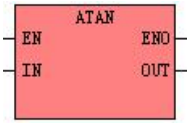
输入 *IN*表示的是弧度值。
SIN 指令将 *IN*求正弦并将结果赋给 *OUT*，如下式所示：*OUT*=SIN(*IN*)。
COS 指令将 *IN*求余弦并将结果赋给 *OUT*，如下式所示：*OUT*=COS(*IN*)。
TAN 指令将 *IN*求正切并将结果赋给 *OUT*，如下式所示：*OUT*=TAN(*IN*)。

- 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。指令的执行不影响 CR 值。

6.8.10 ASIN（反正弦）、ACOS（反余弦）、ATAN（反正切）

➢ 指令及其操作数说明

<table>
<tr><th></th><th>名称</th><th>指令格式</th><th>影响 CR 值</th><th rowspan="6">☑ K5
☑ K2</th></tr>
<tr><td rowspan="3">LD</td><td>ASIN</td><td> 
</td><td rowspan="3"></td></tr>
<tr><td>ACOS</td><td> 
</td></tr>
<tr><td>ATAN</td><td> 
</td></tr>
<tr><td rowspan="3">IL</td><td>ASIN</td><td>ASIN IN, OUT</td><td rowspan="3">U</td></tr>
<tr><td>ACOS</td><td>ACOS IN, OUT</td></tr>
<tr><td>ATAN</td><td>ATAN IN, OUT</td></tr>
</table>

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。指令的执行不影响 CR 值。

6.9 程序控制

6.9.1 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	lbl —(LBL)—		☑ K5 ☑ K2
	JMP	lbl —(JMP)—		
	JMPC	lbl —(JMPC)—		
	JMPCN	lbl —(MPCN)—		
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。

注意，此指令耗时可能较长， 会触发看门狗， 可以加入 WDR 指令来延长看门狗的触发时间，且，注意 JMP 的条件不要造成无法跳出的死循环。

- 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。

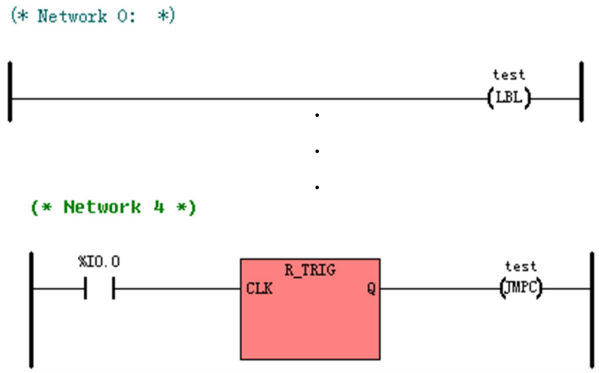
  JMP 指令用于无条件地将程序跳转到 *lbl* 标号处继续执行。

  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 *) test: ... (* NETWORK 4 *) LD %I0.0 R_TRIG JMPC test

6.9.2 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			☑ K5 ☑ K2
	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，KincoBuilder 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。

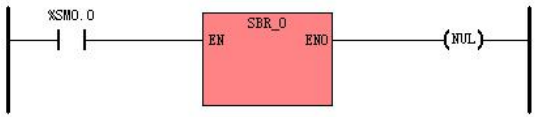
- IL

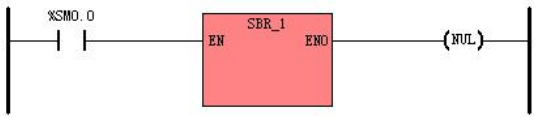
RETC 指令：若 CR 值为 1，则该指令被执行，否则该指令不起作用，程序继续向下依次执行。

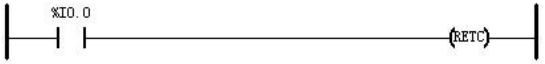
RETCN 指令：若 CR 值为 0，则该指令执行，否则该指令不起作用，程序继续向下依次执行。

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

<table>
<tr>
<td>LD</td>
<td>主程序：
(* Network 0 *)
 

(* Network 1 *)
 

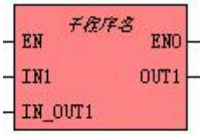
子程序 SBR_0：
(* Network 0 *)
 

(* Network 1 *)
 
</td>
<td>在子程序 SBR_0 中：
若 I0.0 为 0，则继续依次执行下面的指令。
若 I0.0 为 1，则返回到主程序中 SBR_0 的调用点处继续向下执行 Network 1 中的指令。</td>
</tr>
<tr>
<td>IL</td>
<td colspan="2">主程序：
LD %SM0.0 (* 建立 CR，其值恒为 I *)
CAL SBR_0 (* 调用子程序 SBR_0 *)
CAL SBR_1 (* 调用子程序 SBR_1 *)

子程序 SBR_0：
LD %I0.0 (* 建立 CR，其值为 I0.0 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 *)
LD %I0.1 (* 若 RETC 指令没起作用，则该指令及后续指令得以执行 *)
ANDN %I0.2
ST %Q0.0</td>
</tr>
</table>

6.9.3 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			☑ K5 ☑ K2
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

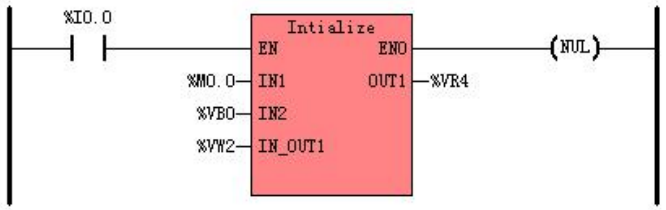
若 CR 值为 1，则调用并执行指定的子程序，否则该指令不起作用，程序继续向下依次执行。CAL 指令的执行不影响 CR 值，但是需要注意的是：CR 值可能在子程序中发生了变化。

➢ 指令使用举例

LD

主程序：

```
(* Network 0 *)
(* 调用子程序Initialize *)
```



子程序 Initialize 中的局部变量表：

	变量地址	变量名称	变量类型	数据类型
▶	%L0.0	IN1	VAR	BOOL
	%LB16	IN2	VAR	BYTE
	%LW22	IN_OUT1	VAR	INT
	%LD18	OUT1	VAR	REAL

在主程序中：
若 I0.0 为 0，则继续依次执行下面的指令。
若 I0.0 为 1，则调用并执行子程序 Initialize。

IL

主程序：

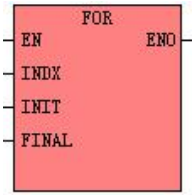
```
(* Network 0 *)
LD      %I0.0                                   (* 建立 CR，其值为 I0.0 的值 *)
CAL     Intialize %M0.0, %VB0, %VW2, %VR10      (* 若 CR 为 1，则调用并执行 Intialize *)
```

子程序 Initialize 中的局部变量表：

	变量地址	变量名称	变量类型	数据类型
▶	%L0.0	IN1	VAR	BOOL
	%LB16	IN2	VAR	BYTE
	%LW22	IN_OUT1	VAR	INT
	%LD18	OUT1	VAR	REAL

6.9.4 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			☑ K5 ☑ K2
	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

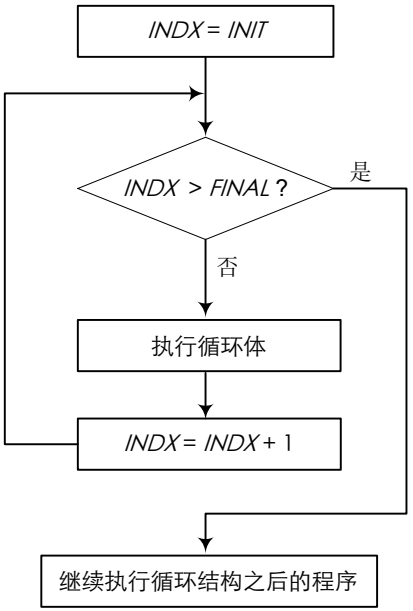
FOR/NEXT 用来实现循环，适用于循环次数已经确定的情况。

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。用户可以在循环体中使用 WDR 指令来延长看门狗的触发时间。

✧ ***FINAL* 参数不允许等于 32767， *INIT* 必须小于等于 *FINAL*，否则 PLC 不会执行此循环体。**

- 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。

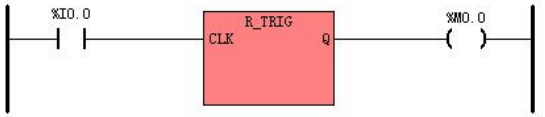
- IL

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。

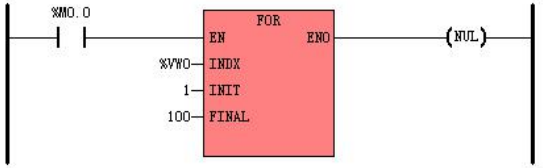
➢ 指令使用举例

LD

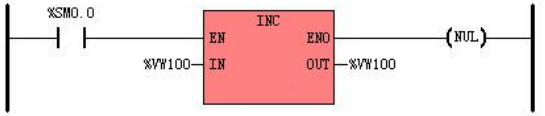
(* Network 0 *)
(* 若检测到I0.0的上升沿，则下面的循环体被执行100次 *)



(* Network 1 *)



(* Network 2 *)



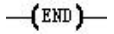
(* Network 3 *)

IL

```
(* Network 0 *)
(*若检测到 I0.0 的上升沿，则下面的循环体被执行 100 次*)
LD      %I0.0
R_TRIG
ST      %M0.0
(* Network 1 *)
LD      %M0.0
FOR     %VW0, 1, 100
(* Network 2 *)
LD      %SM0.0
INC     %VW100
(* Network 3 *)
LD      TRUE
NEXT
```

6.9.5 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			☑ K5 ☑ K2
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，KincoBuilder 软件自动地隐含调用了 END 指令。

- LD

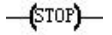
若指令左侧能量流的值为 1，则该指令被执行，否则该指令将不起作用，程序继续向下依次执行。

- 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。

6.9.6 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			☑ K5 ☑ K2
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。

6.9.7 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)—		☑ K5 ☑ K2
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。

6.10 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。Kinco-K5 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。

6.10.1 Kinco-K 系列如何处理中断事件?

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。Kinco-K 系列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。

6.10.2 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。

6.10.3 中断事件分类

- 通讯口中断

用于自由协议通讯。

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。

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。

➢ 时间中断

包含了定时中断和定时器中断。

定时中断会周期性的产生，周期单位为 0.1ms，可以利用它来完成周期性的任务。定时中断不受 PLC 扫描周期的影响，可以用于精确的定时。**(这类中断是可以产生不受 PLC 扫描周期影响的定时)**

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。定时器中断会受到 PLC 扫描周期的影响。**(这类中断受 PLC 扫描周期影响)**

6.10.4 中断事件列表

下表是 Kinco-K 系列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。比如 CPU504 只有 PORT0 一个通讯口，那么关于 PORT1 的中断事件就不支持。又比如 K5 系列 CPU 只支持 HSC0，HSC1，那么 HSC2 和 HSC3 的中断事件就不支持。

事件号	中断描述	分类
191	HSC3 使用多段 PV 值时第 32 个“CV=PV”	I/O 中断
…	…（依次加 1）	
161	HSC3 使用多段 PV 值时第 2 个“CV=PV”	
160	HSC3 使用多段 PV 值时第 1 个“CV=PV”	
159	HSC2 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
129	HSC2 使用多段 PV 值时第 2 个“CV=PV”	
128	HSC2 使用多段 PV 值时第 1 个“CV=PV”	
127	HSC1 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
97	HSC1 使用多段 PV 值时第 2 个“CV=PV”	
96	HSC1 使用多段 PV 值时第 1 个“CV=PV”	
95	HSC0 使用多段 PV 值时第 32 个“CV=PV”	
…	…（依次加 1）	
65	HSC0 使用多段 PV 值时第 2 个“CV=PV”	
64	HSC0 使用多段 PV 值时第 1 个“CV=PV”	
63-33	保留	
34	PORT 2：发送数据完成	通讯中断
33	PORT 2：接收数据完成	
32	PORT 1：发送数据完成	
31	PORT 1：接收数据完成	
30	PORT 0：发送数据完成	
29	PORT 0：接收数据完成	
28--27	保留	I/O 中断

26	I0.0 检测到下降沿	
25	I0.0 检测到上升沿	
24	I0.1 检测到下降沿	
23	I0.1 检测到上升沿	
22	I0.2 检测到下降沿	
21	I0.2 检测到上升沿	
20	I0.3 检测到下降沿	
19	I0.3 检测到上升沿	
18	HSC0 使用单个 PV 值时 CV=PV（当前值=预设值）	
17	HSC0 输入方向改变	
16	HSC0 外部复位	
15	HSC1 使用单个 PV 值时 CV=PV（当前值=预设值）	
14	HSC1 输入方向改变	
13	HSC1 外部复位	
12	HSC2 使用单个 PV 值时 CV=PV（当前值=预设值）	
11—10	保留	
9	HSC3 使用单个 PV 值时 CV=PV（当前值=预设值）	
8—5	保留	
4	定时中断 1。周期在 SMD16 中指定，单位 0.1ms。**（不受扫描周期影响）**	时间中断
3	定时中断 0。周期在 SMD12 中指定，单位 0.1ms。**（不受扫描周期影响）**	
2	定时器 T3 ET=PT（当前值=预设值）**（受扫描周期影响）**	
1	定时器 T2 ET=PT（当前值=预设值）**（受扫描周期影响）**	

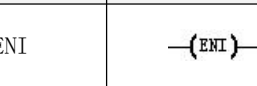
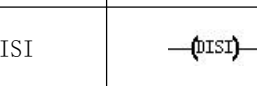
表 6-1 K5 的中断事件列表

事件号为 3，4 的定时中断 1， 定时中断 2， 不受 PLC 扫描的影响， 但受更高优先级的硬件中断影响， 所定时的时间大多偏长， 用户可以在测转速，流量时， 根据自己的应用程序， 加以补偿。

事件号为 1，2 的定时器中断，受 PLC 扫描的影响， 精确定时建议用 3，4 中断。

6.10.5 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			☑ K5 ☑ K2
	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

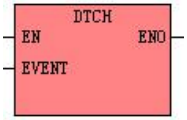
ENI 指令的作用是：若 CR 值为 1，则全局允许处理中断事件，否则不执行该指令。

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。

6.10.6 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			☑ K5 ☑ K2
	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，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

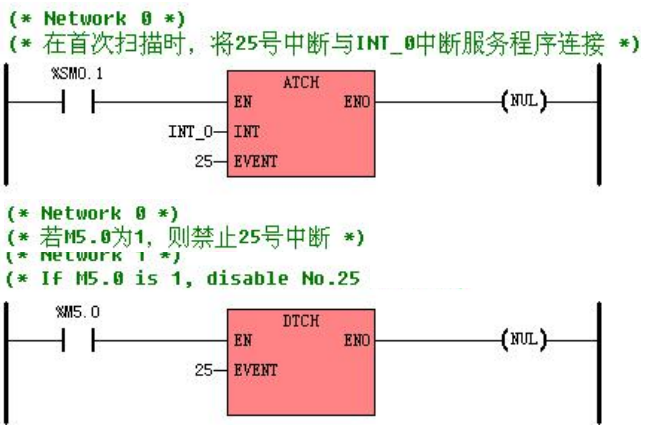
  若 *EN* 值为 1，则 *ATCH*、*DTCH* 指令 被执行，否则不执行。

- 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 *)
LD      %M5.0          (* 建立 CR，其值为 M5.0 *)
ENI                    (* 若 CR 值为 1，则允许进行中断处理 *)

LDN     %M5.0          (* 建立 CR，其值为 M5.0 取反后的值 *)
DISI                   (* 若 CR 值为 0，则禁止进行中断处理 *)

LD      %SM0.1         (* 在首次扫描时，将 25 号中断与 INT_0 中断服务程序连接 *)
ATCH    INT_0, 25

LD      %I1.0          (* 建立 CR，其值为 I1.0 *)
DTCH    25             (* 若CR值为1,则取消 25 号中断与所有中断服务程序的连接 *)
```

6.11 实时时钟

CPU504不支持实时时钟。其它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容供电。常温下，后备的时间不低于3年。

K2允许使用特定规格的的锂电池作为后备电池。当断电时，后备电池用于给实时时钟供电来维持时钟的运行，同时也给RAM供电来进行数据保持。，常温下，电池典型寿命为5年,断电电保持的时间累计不小于3年。

后备电池可以拆卸，在模块做侧面上部是电池盒的位置，电池就安装在电池盒中。当后备电池耗完电后，用户可以打开电池盒自行更换一个新的电池。

电池为CR2032带连接器的3V锂电池，形状如下图，用户可以单独订购电池。

6.11.1 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 CPU 时钟…**】菜单命令进入“调整 CPU 时钟”对话框，如下图。

- **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

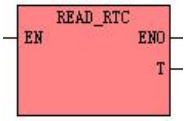
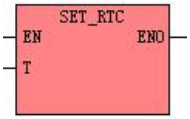
读到了实时时钟。 若背景色是黄色，则代表从 CPU 内读取实时时钟失败。

- **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。
- **使用夏时制**：在使用夏时制的国家或者地区，需要选中此项。
- 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。

注意：修改时区或夏时制之后， 必须重启 kincobuilder（这是 windows 的机制）。

6.11.2 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			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508 ☑ K2
	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）		

注意， T 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域， 否则结果不可预期。

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。注意该缓冲区不能超出有效的内存范围，否则执行结果不可预期。

时间缓冲区内的数据存储形式如下表所示。

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

表 6-2　实时时钟指令的时间缓冲区

提示：

（1） 执行 KincoBuilder 中的【PLC】→【调整 CPU 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使用菜单命令为 CPU 设置好正确的时钟。

⑵ CPU 不检查用户输入的日期、时间是否有效，无效的日期（比如 10 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。

6.11.3 RTC_R（读实时时钟）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	RTC_R			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508 ☑ K2
IL	RTC_R	RTC_R *FMT, WEEK, SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, CENTURY*	U	

参数	输入/输出	数据类型	允许的内存区
FMT	输入	BYTE	L、M、V、常量
WRRK	输出	BYTE	L、M、V
SECOND	输出	BYTE	L、M、V
MINUTE	输出	BYTE	L、M、V
HOUR	输出	BYTE	L、M、V
DAY	输出	BYTE	L、M、V
MONTH	输出	BYTE	L、M、V

YEAR	输出	BYTE	L、M、V
CENTURY	输出	BYTE	L、M、V

下表详细描述了各个参数的作用。

参数	描　述
EN	使能端。
FMT	输出格式，0 表示十进制，1 表示 BCD 码。
WRRK	星期，范围：1~7，其中 1 代表星期一，7 代表星期日。
SECOND	秒，范围：0~59
MINUTE	分，范围：0~59
HOUR	小时，范围：0~23
DAY	日，范围：1~31
MONTH	月，范围：1~12
YEAR	年，范围：0~99
CENTURY	世纪，固定为 20

RTC_R 用于从实时时钟中读取当前的日期和时间值并写入各个输出参数。
FMT 指定了各参数的格式， 若 FMT 为 0， 则为十进制；若 FMT 为 1， 则为 BCD 码。

- LD
  若 *EN* 值为 1，则执行 RTC_R 指令，否则不执行。

- IL
  若 CR 值为 1，则执行 RTC_R 指令，否则不执行。
  RTC_R 指令的执行不影响 CR 值。

6.11.4 RTC_W（写实时时钟）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	RTC_W			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508 ☑ K2
IL	RTC_W	RTC_W *FMT, WEEK, SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, CENTURY*	U	

参数	输入/输出	数据类型	允许的内存区
FMT	输入	BYTE	L、M、V、常量
WRRK	输入	BYTE	L、M、V 、常量
SECOND	输入	BYTE	L、M、V、常量
MINUTE	输入	BYTE	L、M、V、常量
HOUR	输入	BYTE	L、M、V、常量
DAY	输入	BYTE	L、M、V、常量
MONTH	输入	BYTE	L、M、V、常量

YEAR	输入	BYTE	L、M、V、常量
CENTURY	输入	BYTE	L、M、V、常量

下表详细描述了各个参数的作用。FMT,WEEK,SECOND,MINUTE,HOUR,DAY,MONTH,YEAR,CENTURY 参数必须同时为常量或同时为变量。

参数	描　述
EN	使能端。
FMT	输出格式，0 表示十进制，1 表示 BCD 码。
WRRK	星期，范围：1~7，其中 1 代表星期一，7 代表星期日。
SECOND	秒，范围：0~59
MINUTE	分，范围：0~59
HOUR	小时，范围：0~23
DAY	日，范围：1~31
MONTH	月，范围：1~12
YEAR	年，范围：0~99
CENTURY	世纪，固定为 20

RTC_W 用于根据各参数指定的时间来修改实时时钟。
FMT 指定了各参数的格式， 若 FMT 为 0， 则为十进制；若 FMT 为 1， 则为 BCD 码。

- LD
  若 *EN* 值为 1，则执行 RTC_R 指令，否则不执行。

- IL
  若 CR 值为 1，则执行 RTC_R 指令，否则不执行。
  RTC_R 指令的执行不影响 CR 值。

6.12 通讯指令

6.12.1 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及通信数据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。

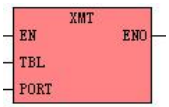
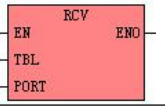
K 系列的 CPU 本体集成了 1 个、2 个或 3 个串行通讯口，这些串口默认采用 Modbus RTU 协议并作为从站。当执行用户程序中的自由通讯指令时，自由通讯方式就被激活，通讯口完全被自由通讯占用。当自由通讯完成后，CPU 又自动将通讯口切换到默认的协议。若 CPU 处于 STOP 状态，则自由通讯被禁止。

下面简要描述了用户进行自由通讯编程的总体步骤：

1) 在【PLC 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）。具体请参阅 4.3.4.1 CPU 参数配置 中的描述。
2) 设置自由通讯控制寄存器（见后面控制寄存器的定义）。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。

6.12.1.1 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			☑ K5 ☑ K2
	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（0-2）

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。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。

注意，TBL 参数为一个可变长度的块内存参数， 整个块内存都不允许超出有效的内存范围，否则结果不可预期。

- 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

Kinco-K5 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) 接收状态字节

位（只读）			值	含 义
PORT 0	PORT 1	PORT 2		
SM86.0	SM186.0	SM286.0	1	接收到的字符存在奇偶校验错误，但不会停止接收。
SM86.1	SM186.1	SM286.1	1	终止接收：达到最大接收字节数。
SM86.2	SM186.2	SM286.2	1	终止接收：接收一个字符超时。
SM86.3	SM186.3	SM286.3	1	终止接收：系统接收超时。
SM86.4	SM186.4	SM286.4	-	保留。
SM86.5	SM186.5	SM286.5	1	终止接收：接收到了用户定义的结束字符。

SM86.6	SM186.6	SM286.6	1	终止接收：参数错误，无起始条件接收或无接收结束条件等。
SM86.7	SM186.7	SM286.7	1	终止接收：用户使用了禁止接收命令。

(2) 接收控制字节

位			值	描 述
PORT 0	PORT 1	PORT 2		
SM87.0	SM187.0	SM287.0	-	保留。
SM87.1	SM187.1	SM287.1	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	SM287.2	0	忽略 SMW92/SMW192/ SMW292 中用户定义的接收字符超时值。
			1	使用 SMW92/SMW192/ SMW292 中用户定义的接收字符超时值。
SM87.3	SM187.3	SM287.3	-	保留。
SM87.4	SM187.4	SM287.4	0	忽略 SMW90/SMW190/SMW290 中用户定义的接收准备时间。
			1	使用 SMW90/SMW190/SMW290 中用户定义的接收准备时间。
SM87.5	SM187.5	SM287.5	0	忽略 SMB89/SMW189/SMW289 中用户定义的接收结束字符。
			1	使用 SMB89/SMW189/SMW289 中用户定义的接收结束字符。
SM87.6	SM187.6	SM287.6	0	忽略 SMB88/SMW188/SMW288 中用户定义的接收开始字符。
			1	使用 SMB88/SMW188/SMW288 中用户定义的接收开始字符。
SM87.7	SM187.7	SM287.7	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。

(3) 其它控制寄存器

控制字（字节）			描 述
PORT0	PORT1	PORT2	
SMB88	SMB188	SMB288	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。 如果要使本设置值生效，需要将 SM87.6/SM187.6/SM287.6 置为 1。
SMB89	SMB189	SMB289	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/SM287.5 置为 1。

SMW90	SMW190	SMW290	用于存放用户定义的接收准备时间（范围为 1~60000ms）。 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。 如果要使本设置值生效，需要将 SM87.4/SM187.4/SM287.4 置为 1。
SMW92	SMW192	SMW292	用于存放用户定义的接收字符超时值（范围为 1~60000ms）。 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，无论其它的结束条件如何。 如果要使本设置值生效，需要将 SM87.2/SM187.2/SM287.2 置为 1。
SMW94	SMW194	SMW294	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。

在自由通讯中，另外还有一个默认的系统接收超时，时间为 90 秒，此超时值的作用如下：在执行 RCV 指令后，若在此超时时间内串口上没有收到任何数据，则 CPU 将立刻终止接收并退出 RCV 指令；另外，CPU 在进入有效接收状态后（即接收到 SMB88 中定义的起始字符或者经过了 SMW90 中定义的接收准备时间后），将优先使用用户在 SMW92 中定义的接收字符超时值，若用户没有定义，则用该系统接收超时值来决定是否终止接收。

➢ 关于通讯中断

Kinco-K 系列提供了多种中断用于自由通讯。若用户需要了解更多关于中断的信息，请参阅6.10.1 Kinco-K 系列如何处理中断事件。

用户可以使用控制位 SM87.1、SM187.1 或 SM287.1 来禁止或允许 CPU 产生通讯中断。若将中断控制位设置为 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产生一个发送完成中断；CPU 在退出接收后（无论是正常还是异常退出）就会产生一个接收完成中断。

➢ 指令使用举例

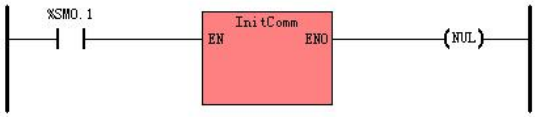
下面将举例说明自由通讯的使用。

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。

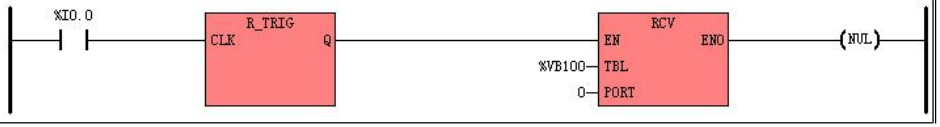
LD

主程序（MAIN）：

```
(* Network 0 *)
(* 首次扫描时，执行一次初始化子程序，目的是初始化自由通讯。 *)
```

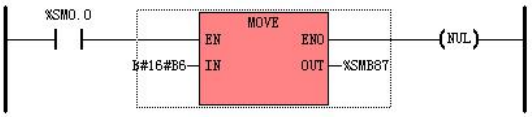



```
(* Network 1 *)
(* 利用I0.0来启动接收任务。 *)
```



InitComm（SBR00），初始化子程序：

(* Network 0 *)
(* 设定有效接收状态的起始条件和终止条件。 *)



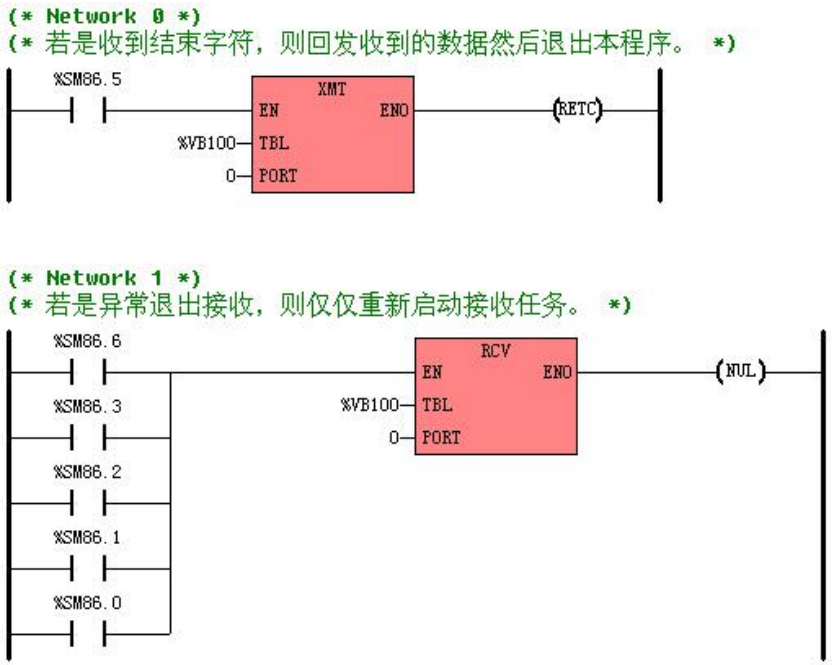
(* Network 1 *)
(* 设定接收准备时间为1ms，接收结束字符为回车符（ASCII为13）。 *)

(* Network 2 *)
(* 设定接收字符超时为500ms，设定每次接收的字符数最大为100个。 *)

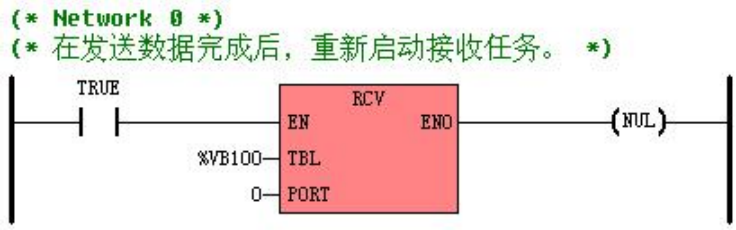
(* Network 3 *)
(* 将EndReceive程序绑定到接收完成中断，将EndSend程序绑定到发送完成中断。 *)

LD

EndReceive（INT00），接收完成中断服务程序：



EndSend（INT01），发送完成中断服务程序：



IL

主程序（MAIN）：

```
(* Network 0 *)
LD      %SM0.1
CAL     InitComm            (*首次扫描时，执行一次初始化子程序。*)
(* Network 1 *)
LD      %I0.0
R_TRIG
RCV     %VB100, 0           (*利用 I0.0 来启动接收任务。*)
```

InitComm（SBR00），初始化子程序：

```
(* Network 0 *)
LD      %SM0.0
MOVE    B#16#B6, %SMB87     (* 设定有效接收状态的起始条件和结束条件 *)
MOVE    1, %SMW90           (* 接收准备时间设定为 1ms *)
MOVE    B#16#D, %SMB89      (* 接收结束字符设定为回车符（ASCII 为 13）*)
MOVE    500, %SMW92         (* 接收字符超时设定为 500ms *)
MOVE    B#100, %SMB94       (* 每次接收的字符数设定为最大 100 个 *)
ATCH    EndReceive, 29      (* 将 EndReceive 程序绑定到接收完成中断 *)
ATCH    EndSend, 30         (* 将 EndSend 程序绑定到发送完成中断 *)
```

EndReive（INT00），接收完成中断服务程序：

```
(* NETWORK 0 *)
LD      %SM86.5
XMT     %VB100, 0           (* 收到结束字符则回发收到的数据 *)
RETC                        (* 然后退出本程序 *)

(* NETWORK 1 *)
LD      %SM86.0
OR      %SM86.1
OR      %SM86.2
OR      %SM86.3
OR      %SM86.6
RCV     %VB100, 0           (*若是异常退出接收状态，则重新启动接收任务*)
```

EndSend（INT01），发送完成中断服务程序：

```
(* NETWORK 0 *)
LD      TRUE
RCV     %VB100, 0           (*发送完成后重新启动接收任务*)
```

6.12.2 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，Kinco-K 系列提供 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。

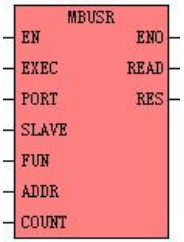
下面简要描述了用户进行 Modbus 主站编程的总体步骤：

1) 在【**PLC 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 4.3.4.1 CPU 参数配置 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 。

6.12.2.1 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			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508 ☑ K2
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（0-2）
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

注意：参数 *SLAVE, ADDR, COUNT* 必须同时为常量类型或同时为内存类型。

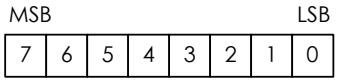
注意， READ 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域，否则结果不可预期。

K 系列 PLC 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。*SLAVE* 定义了目标从站的站号，允许的站号范围是 1~255。*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：



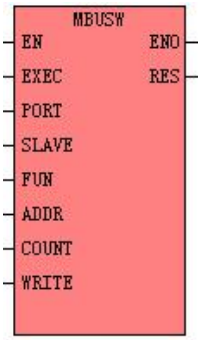
第7位 --- 当前的通讯状态。通讯时该位被置为0，否则被置为1。
第6位 --- 保留。
第5位 --- 非法的从站站号。
第4位 --- 数据个数错误。
第3位 --- 非法的寄存器地址。
第2位 --- 通讯口正被其它通讯任务占用。
第1位 --- 通讯超时：在用户指定的超时时间内仍没有收到从站的正确回应。
第0位 --- 接收数据帧错误，包括CRC校验错、功能码错等。

- 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。

6.12.2.2 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			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508 ☑ K2
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（0-2）
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

参数 *SLAVE, ADDR, COUNT* 必须同时为常量类型或同时为内存类型。

注意， WRITE 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域，否则结果不可预期。

K 系列 PLC5 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

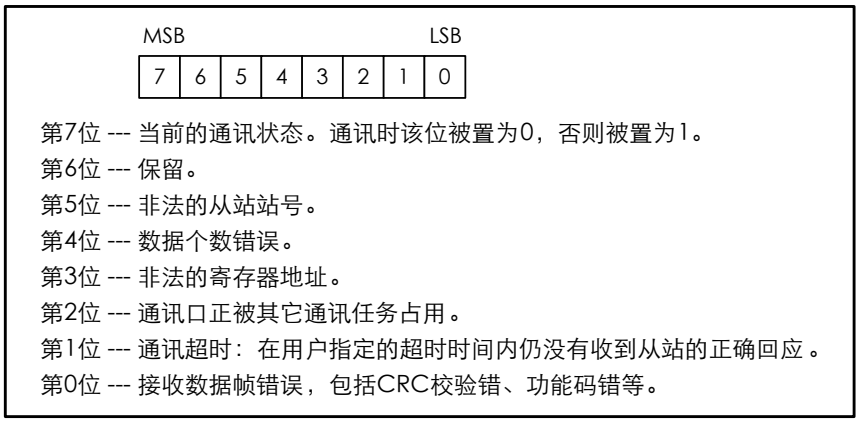
码有 5（写一个 DO）、6（写一个 AO）、15（写多个 DO）和 16（写多个 AO）。

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：



- LD

  如果 *EN* 为 1，则该指令被执行，否则不执行。

- IL

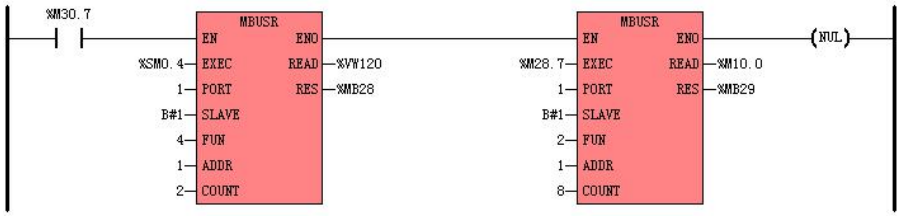
  如果 CR 值为 1，则该指令被执行，否则不执行。该指令的执行不影响 CR 值。

6.12.2.3 MBUSR、MBUSW 使用举例

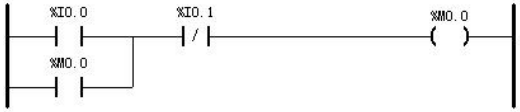
LD

```
(* 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。 *)
```

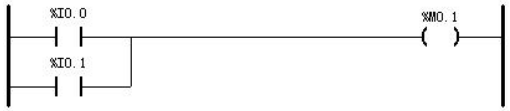

```
(* Network 1 *)
(* 若通讯口未被MBUSW指令占用，则执行MBUSR指令：
   每2秒钟从1#从站读取数据：先读1号和2号AI寄存器，完成后接着读1-8号DI寄存器。 *)
```



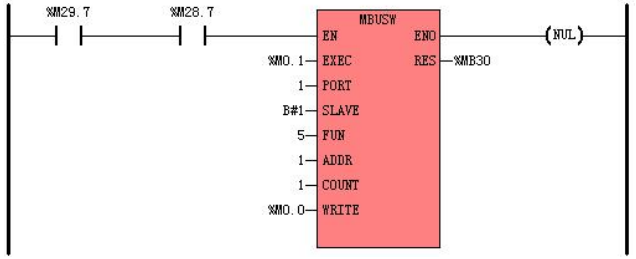

```
(* Network 2 *)
(* MBUSW指令举例 *)
```




```
(* Network 3 *)
```




```
(* 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。 *)
```



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
```

6.12.3 CANopen 功能及 SDO 指令

➢ **CANopen 支持的功能主要有：**（本功能 K2 系列 CPU 不支持）

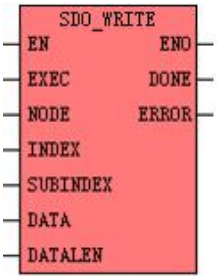
- 支持 NMT 管理报文；
- 支持 CANopen 预定义连接集模式，PDO 总数量为 32TxPDO 和 32RxPDO,每个从站可配置的 PDO 数量为 1-8TxPDO、 1-8RxPDO；
- 可以连接 32 个从站,特定版本支持 64 个从站，从站站号 1-126 任意设置；
- 支持紧急报文、节点保护、心跳报文；
- 可以对从站的启动过程进行设置；
- 支持 EDS 导入功能；
- 通过 SDO_WRITE 和 SDO_READ 命令字实现对 SDO 的操作；
- 支持 254、255 两种 PDO 传输模式，并具有定时发送 PDO 的功能，能同时发送 8 个 PDO 数据，且定时时间可设；
- 支持各种波特率通信：10K/20K/50K/125K/250K/500K/800K/1M；
- 通过系统字检查网络上主从站的状态；
- 具有多 PLC 通过 CAN 总线联网能力；
- 具有从站错误处理功能，当从站出错后，主站可以选择的处理方式有三种，即：停止节点、停止网络 、无 。

➢ **SDO 命令：**

SDO(服务数据对象，Servic Data Object)是 CANOpen 定义的一种通信对象（报文），主要用来在设备之间传输低优先级的数据，典型是用来对从设备进行配置、管理,比如用来修改伺服电流环、速度环、位置环的 PID 参数，PDO 配置参数等，这种数据传输跟 MODBUS 的方式一样，即主站发出后，需要从站返回数据响应。这种数据只适合对参数的设置，不适合于对实时性要求较高的数据传输。SDO 的具体的报文格式请参见《附录 F CANOpen 主站功能的使用》

6.12.3.1 SDO_WRITE

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SDO_WRITE			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508
IL	SDO_WRITE	SDO_WRITE *EXEC, NODE, INDEX, SUBINDEX, DATA, DATALEN, DONE, ERROR*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
NODE	输入	BYTE	I、Q、V、M、L、SM、常量
INDEX	输入	WORD	I、Q、V、M、L、SM、常量
SUBINDEX	输入	BYTE	I、Q、V、M、L、SM、常量
DATA	输入	BYTE	I、Q、V、M、L、SM
DATALEN	输入	BYTE	I、Q、V、M、L、SM、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERROR	输出	DWORD	Q、M、V、L、SM

NODE, INDEX, SUBINDEX, DATALEN 必须同时为常量或同时为变量。

注意， DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域，否则结果不可预期。

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
EXEC	启动端。*EXEC* 的上升沿用于启动 SDO 通信。最好能够保证让 *EN* 先于 *EXEC* 导通。
NODEID	待访问节点的地址
Index	待访问对象在 OD 中的索引

SubIndex	待访问对象在 OD 中的子索引
Data	待发送数据存放的起始字节
DataLen	接发送数据的长度，单位：字节
DONE	执行结果指示。 若 SDO 正在执行，DONE 为 0；若 SDO 通信结束（收到回应或者超时），DONE 为 1。
ERROR	错误信息。见下表

SDO 错误信息说明，见下表：

错误码	说明
0	无错误。
1	SDO 指令的数量超过了最大允许的数量。 K5 允许一个工程中 SDO_WRITE 和 SDO_READ 指令的数量最大卫 72 个。
2	主站不处于 Operational 状态，因此 SDO 报文不发送。
4	目标节点不存在或者因故障 STOP，因此 SDO 报文不发送。
5	上一次发送的相同命令尚未得到回应
6	指令的输入参数值有错误。
8	超时没有收到回应报文。 SDO 超时值再【主站及全局配置】页面中设定。
9	回应报文的数据长度错误。
10	回应报文不是期望的报文。

• LD

如果 EN 为 1，则该指令被执行，否则不执行。

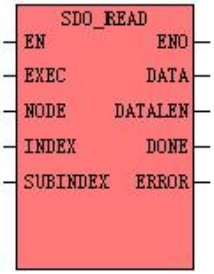
• IL

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。

6.12.3.2 SDO_READ

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	SDO_READ			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508
IL	SDO_READ	SDO_READ *EXEC, NODE, INDEX, SUBINDEX, DATA, DATALEN, DONE, ERROR*	U	

参数	输入/输出	数据类型	允许使用的内存区
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
NODE	输入	BYTE	I、Q、V、M、L、SM、常量
INDEX	输入	WORD	I、Q、V、M、L、SM、常量
SUBINDEX	输入	BYTE	I、Q、V、M、L、SM、常量
DATA	输出	BYTE	I、Q、V、M、L、SM
DATALEN	输出	BYTE	I、Q、V、M、L、SM
DONE	输出	BOOL	Q、M、V、L、SM
ERROR	输出	DWORD	Q、M、V、L、SM

NODE, INDEX, SUBINDEX 必须同时为常量或同时为变量。

注意， DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域，否则结果不可预期。

各个参数的具体使用说明，见下表：

参数	功能
EN	使能端。若 *EN* 为 1 时，则该指令被使能，允许执行。
EXEC	启动端。*EXEC* 的上升沿用于启动 SDO 通信。最好能够保证让 *EN* 先于 *EXEC* 导通。
NODEID	待访问节点的地址

Index	待访问对象在 OD 中的索引
SubIndex	待访问对象在 OD 中的子索引
Data	接收到数据存放的起始字节
DataLen	接收到数据的长度，单位：字节
DONE	执行结果指示。 若 SDO 正在执行，DONE 为 0；若 SDO 通信结束（收到回应或者超时），DONE 为 1。
ERROR	错误信息。见下表

SDO 错误信息说明，见下表：

错误码	说明
0	无错误。
1	SDO 指令的数量超过了最大允许的数量。 K5 允许一个工程中 SDO_WRITE 和 SDO_READ 指令的数量最大卫 72 个。
2	主站不处于 Operational 状态，因此 SDO 报文不发送。
4	目标节点不存在或者因故障 STOP，因此 SDO 报文不发送。
5	上一次发送的相同命令尚未得到回应
6	指令的输入参数值有错误。
8	超时没有收到回应报文。 SDO 超时值再【主站及全局配置】页面中设定。
9	回应报文的数据长度错误。
10	回应报文不是期望的报文。

• LD

如果 EN 为 1，则该指令被执行，否则不执行。

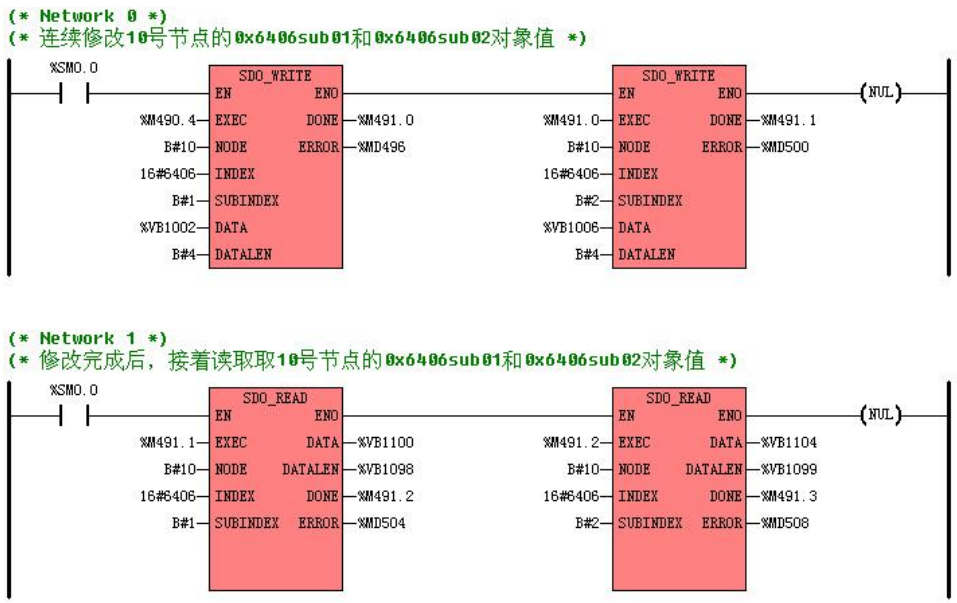
• IL

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。

6.12.3.3 SDO_WRITE、SDO_READ 使用举例

LD



IL

```
(* Network 0 *)
(*连续修改 10 号节点的 0x6406sub01 和 0x6406sub02 对象值*)
LD      %SM0.0
SDO_WRITE %M490.4, B#10, 16#6406, B#1, %VB1002, B#4, %M491.0, %MD496
SDO_WRITE %M491.0, B#10, 16#6406, B#2, %VB1006, B#4, %M491.1, %MD500
(* Network 1 *)
(*修改完成后，接着读取取 10 号节点的 0x6406sub01 和 0x6406sub02 对象值*)
LD      %SM0.0
SDO_READ %M491.1, B#10, 16#6406, B#1, %VB1100, %VB1098, %M491.2, %MD504
SDO_READ %M491.2, B#10, 16#6406, B#2, %VB1104, %VB1099, %M491.3, %MD508
```

6.12.4 CAN 通信指令

K5 系列 CPU 提供了 CANOpen 主站功能和 CAN 自由通信功能，均需配合 K541 模块使用。

CANOpen 主站和 CAN 自由通信指令允许同时使用。需要注意的是，同时使用时，网络上所有节点的波特率必须一致。

CAN 通信指令支持 CAN2.0A 和 CAN2.0B 标准，另外这些指令只支持数据帧，不支持远程帧。CAN 数据帧格式如下：

ID	字节 1-8
11 位（CAN2.0A，标准帧）或 29 位（CAN2.0B，扩展帧）	1-8 字节长度的数据

6.12.4.1 CAN_INIT（初始化 CAN 接口）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CAN_INIT			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508
IL	CAN_INIT	CAN_INIT CH, BAUD	U	

参数	输入/输出	数据类型	允许的内存区
EN	输入	BOOL	I、Q、V、M、L、SM
CH	输入	INT	常量（目前仅允许为 2）
BAUD	输入	INT	L、M、V、常量
ERR	输出	BOOL	L、M、V、常量

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。2 表示 K541 模块
BAUD	CAN 的波特率。 8 --- 1000K 7 --- 800K 6 --- 500K 5 --- 250K 4 --- 125K

	3 --- 50K 2 --- 20K 1--- 10K
ERR	指令执行是否成功。0 表示成功，1 表示有错误（比如参数错误）

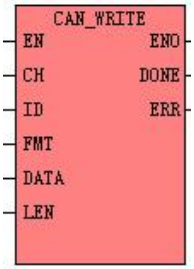
*EN*输入端的上升沿跳变会触发执行该指令，用于初始化指定的 CAN 接口（*CH*），并将 CAN 波特率设置为 BAUD 代表的数值。

- LD
  *EN*的上升沿会触发该指令执行一次，否则不执行。

- IL
  CR 的上升沿会触发该指令执行一次，否则不执行。
  该指令的执行不影响 CR 值。

6.12.4.2 CAN_WRITE（发送一次 CAN 报文）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CAN_WRITE			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508
IL	CAN_WRITE	CAN_WRITE CH, ID, FMT, DATA, LEN, DONE, ERR	U	

参数	输入/输出	数据类型	允许的内存区
CH	输入	INT	常量（目前仅允许为 2）
ID	输入	DWORD	L、M、V、常量
FMT	输入	BYTE	L、M、V、常量
DATA	输入	BYTE	L、M、V
LEN	输入	BYTE	L、M、V、常量
DONE	输出	BOOL	L、M、V
ERR	输出	BOOL	L、M、V

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。2 表示 K541 模块
FMT	报文格式。0 表示标准帧，1 表示扩展帧。
DATA	待发送数据存放的起始字节地址。
LEN	待发送数据的长度。单位：字节。
DONE	报文是否发送完成。在执行时 DONE 被置 0，发送完成则 DONE 被置 1。
ERR	报文发送是否出错。若发送失败（通常由于发送缓冲区满），则 ERR 被置 1。

ID, FMT, LEN, 必须同时为常量类型或同时为内存类型

注意， DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域，

否则结果不可预期。

待发送的 CAN 报文由 *ID* 号、格式（*FMT*，指明扩展帧或者标准帧）、数据（*DATA*，指明报文数据存放的起始地址）、长度 *LEN* 来指定。

EN 输入端的上升沿跳变会触发执行一次该指令，将待发送的报文写入 PLC 内部的发送缓冲区中，再由 PLC 调度通过通过指定的 CAN 接口 *CH* 发送出去。

若指令成功将报文写入发送缓冲区中，则表示执行完成，将 *DONE* 置为 1。若发生缓冲区已满，则表示发送失败，指令同时将 *DONE* 和 *ERR* 置为 1。

- LD

  EN 的上升沿会触发该指令执行一次，否则不执行。

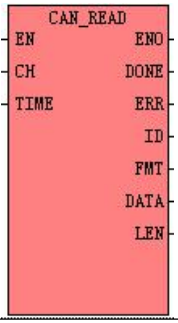
- IL

  CR 的上升沿会触发该指令执行一次，否则不执行。

  该指令的执行不影响 CR 值。

6.12.4.3 CAN_READ（接收一次 CAN 报文）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CAN_READ			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508
IL	CAN_READ	CAN_READ CH, TIME, DONE, ERR, FMT, DATA, LEN	U	

参数	输入/输出	数据类型	允许的内存区
CH	输入	INT	常量（1 和 2）
TIME	输入	INT	L、M、V、常量
DONE	输出	BOOL	L、M、V
ERR	输出	BOOL	L、M、V
ID	输出	DWORD	L、M、V
FMT	输出	BYTE	L、M、V
DATA	输出	BYTE	M、V
LEN	输出	BYTE	L、M、V

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。2 表示 K541 模块
TIME	超时时间。启动接收后，若在指定的这个时间内没有收到任何报文，则超时退出接收状态，并将 ERR 置 1
ID	接收到的报文的 ID 号。
FMT	接收到的报文格式。0 表示标准帧，1 表示扩展帧。
DATA	接收到的报文数据存放的起始字节地址。
LEN	接收到的报文数据长度。单位：字节。

DONE	是否接收完成。在执行时 DONE 被置 0，发送完成则 DONE 被置 1。
ERR	接收报文是否出错。若接收失败则 ERR 被置 1。

注意， DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域，否则结果不可预期。

*EN*输入端的上升沿跳变会触发执行该指令：启动接收状态，接收任何一条来自指定 CAN 接口 *CH* 的报文。

启动接收后，若在指定的超时时间 *TIME* 内接收到一条 CAN 报文，那么 PLC 就将报文相关数据分别置于输出参数 *ID*（ID 号）、*FMT*（格式，指明接收到的是扩展帧或者标准帧）、*DATA*（接收到的报文数据存放的起始地址)、LEN（长度）中，同时将 DONE 置为 1，退出接收。若在指定的超时时间 *TIM*) 内没有收到任何报文，则超时退出接收状态，并将 *DONE* 和 ERR 置 1。

CAN_READ 指令启动后，将会接收指定 CAN 接口的任何一条报文，因此，与其它协议（比如 CANOpen）混用时需要注意。

- LD

  EN 的上升沿会触发该指令执行一次，否则不执行。

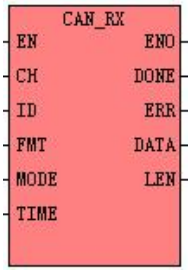
- IL

  CR 的上升沿会触发该指令执行一次，否则不执行。

  该指令的执行不影响 CR 值。

6.12.4.4 CAN_RX（接收特定 ID 号 CAN 报文）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	CAN_RX			☐ CPU504 ☑ CPU504EX ☑ CPU506 ☑ CPU506EA ☑ CPU508
IL	CAN_RX	CAN_RX CH, ID, FMT, MODE, TIME, DONE, ERR, FMT, DATA, LEN	U	

参数	输入/输出	数据类型	允许的内存区
CH	输入	INT	常量（目前仅允许为 2）
ID	输入	DWORD	L、M、V、常量
FMT	输入	INT	L、M、V、常量
MODE	输入	INT	L、M、V、常量
TIME	输入	INT	L、M、V、常量
DONE	输出	BOOL	L、M、V
ERR	输出	BOOL	L、M、V
DATA	输出	BYTE	M、V
LEN	输出	BYTE	L、M、V

参数	描　述
EN	使能端。
CH	所用的 CAN 接口。2 表示 K541 模块
ID	待接收报文的 ID
FMT	待接收报文的格式。0 表示标准帧，1 表示扩展帧。
MODE	接收模式。0 表示永久接收模式，1 表示单次接收模式
TIME	接收超时时间。单位 ms
DONE	在单次接收模式下，DONE 是接收成功标志位。
ERR	接收超时标志位。
DATA	最近一次接收的报文数据存放的首地址。
LEN	最近一次接收的报文的数据长度，单位：字节。

ID，FMT，MODE，TIME，必须同时为常量类型或同时为内存类型

注意， DATA 参数为一个可变长度的块内存参数， 整个块内存都不可以落在非法内存区域，否则结果不可预期。

该指令用于接收指定 ID 号（*ID*）和格式（*FMT*）的报文。

MODE 参数指明了接收方式，若 *MODE* 为 1，则为单次接收模式，指令只接收一次指定报文，收到后则退出；若 *MODE* 为 0，则为永久接收模式，指令会一直接收指定报文。

EN 输入端的上升沿跳变会触发执行该指令，PLC 进入接收状态，同时将 *DONE*、*ERR* 清 0。若为单次接收模式，那么如果在超时时间 *TIME* 内接收到指定报文，则将 *DONE* 置 1，指令退出接收状态，如果在超时时间 *TIME* 内没有收到指定报文，那么 *DONE* 和 *ERR* 被置 1，指令退出接收状态；若为永久接收模式，那么指令会一直监视指定 CAN 接口 *CH* 并接收所有的指定报文，若在一次成功接收之后，在 *TIME* 内没有再次接收到指定报文，则将 ERR 置 1，之后若再次成功接收，那么就会将 ERR 清 0。

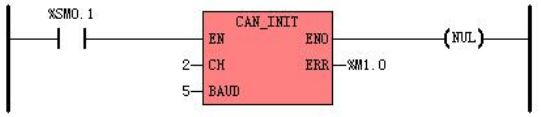
- LD
  EN 的上升沿会触发执行该指令并进入接收状态，否则不执行。

- IL
  CR 的上升沿会触发执行该指令并进入接收状态，否则不执行。
  该指令的执行不影响 CR 值。

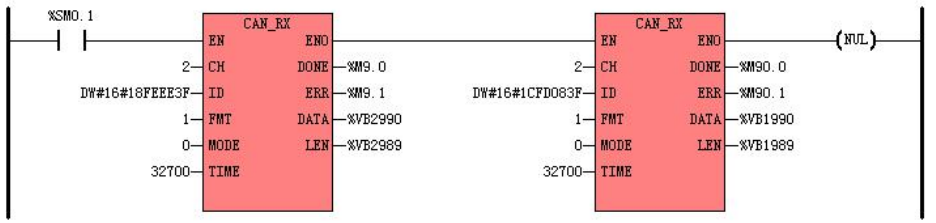
➢ 指令使用举例

LD

```
(* Network 0 *)
(* 设置K541波特率 *)
```




```
(* Network 1 *)
(* 指定将要自动接收的报文ID、格式。采用永久接收模式。
PLC只要接收到指定的报文，就会将报文数据存放到指令指定的内存区域中。 *)
```



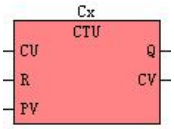
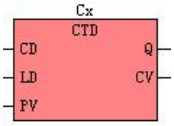
6.13 计数器

计数器是 IEC61131-3 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 3.6.5 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。

6.13.1 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			☑ K5 ☑ K2
	CTD			
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
CD	输入	BOOL	能量流
LD	输入	BOOL	I、Q、M、V、L、SM、T、C
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* 输入端的下降沿进行递减计数（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

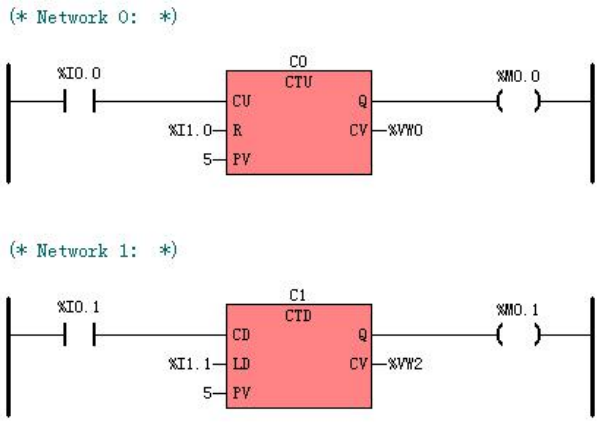
CTU 指令对 CR 值的上升沿进行递增计数（每次加 1）。*Cx* 将一直计数直至达到并保持在最大值 32767。当计数值大于等于预设值 *PV* 时，*Cx* 的状态值被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值被清零。

CTD 指令对 CR 值的下降沿进行递减计数（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。

➢ CTU、CTD 使用举例

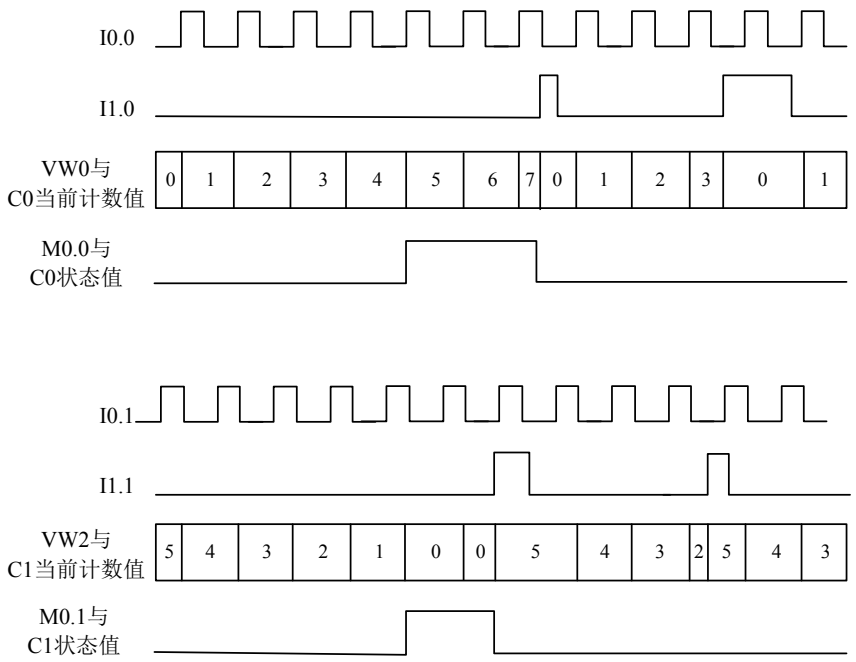
LD



IL

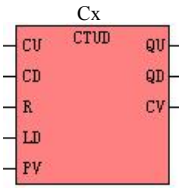
```
(* NETWORK 0 *)
LD      %I0.0
CTU     C0, %I1.0, 5
ST      %M0.0

(* NETWORK 1 *)
LD      %I0.1
CTD     C1, %I1.1, 5
ST      %M0.1
```

时序图



6.13.2 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  		☑ K5 ☑ K2
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

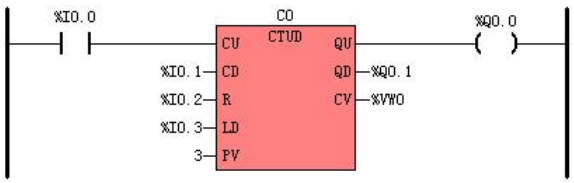
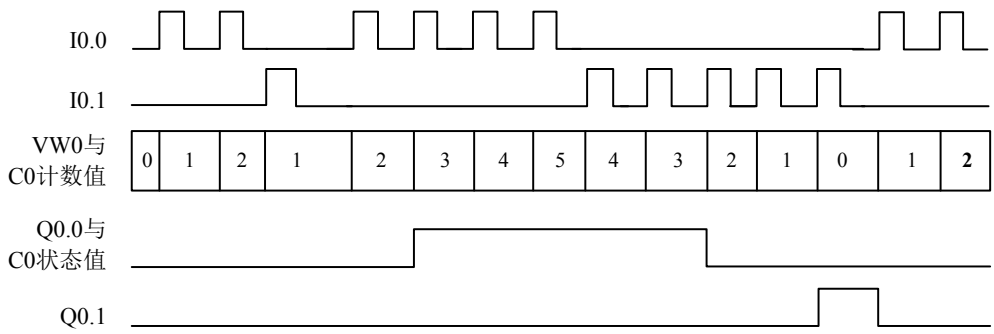
CTUD 指令对 *CU* 输入端的上升沿进行递增计数（每次加 1）并对 *CD* 输入端的上升沿进行递减计数（每次减 1），*Cx* 的计数值被赋给 *CV*。当 *CV* 大于等于预设值 *PV* 时，*QU* 及 *Cx* 的状态值均被置为 1，否则均被置为 0。当 *CV* 等于 0 时，*QD* 被置为 1，否则被置为 0。若 *R* 输入为 1，则 *Cx* 被复位，其计数值及 *CV* 均被清零。若 *LD* 输入为 1，则将 *PV* 重新赋值给 *Cx* 的计数值及 *CV*。当 *R* 及 *LD* 同时输入为 1 时，*R* 优先。

- 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。

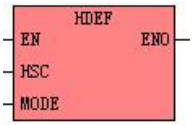
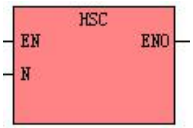
➢ CTUD 使用举例

<table>
<tr><td>LD</td><td>IL</td></tr>
<tr><td> 
</td><td>LD %I0.0
CTUD C1, %I0.1, %I0.2, %I0.3, 4 ,%Q0.1
ST %Q0.0</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.13.3 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			☑ K5 ☑ K2
	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(0-11，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*。

注意 HSC 指令只用在需要的时候调用一次就可以启动高速计数。若用 SM0.0 一直调用，则高速计数器会一直被初始化过，无法使用。

- 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。

6.13.3.1 Kinco-K 系列中的高速计数器

	K5 系列
高速计数器 单相 双相	2 个（HSC0 和 HSC1） 2 个，最高计数频率 60KHz 2 个，最高计数频率 20KHz
	K2 系列
高速计数器 单相 双相	4 个（HSC0，HSC1，HSC2，HSC3） 4。HSC0、HSC1 最高计数频率 50KHz，HSC2、HSC3 最高计数频率 20KHz。 4，HSC0、HSC1 最高计数频率 50KHz，HSC2、HSC3 最高计数频率 10KHz。

高速计数器具有各种工作模式，在启动高速计数器之前，必须用 HDEF 指令为其指定一种工作模式。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。

6.13.3.2 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、启动和复位信号。

若启动信号为 0，则时钟信号和复位信号均被忽略并且当前计数值保持不变。若启动信号为 1 且复位信号为 0，则允许高速计数器计数。若复位信号和启动信号均为 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。

HSC 0				
模式	**描述**	**I0.1**	**I0.0**	**I0.5**
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			复位	
2			复位	启动
3	带外部方向控制的单相增/减计数器	时钟		方向
4			复位	方向
6	带增/减计数时钟输入的双相计数器	时钟（减）	时钟（增）	
9	A/B 相正交计数器	时钟 A 相	时钟 B 相	

HSC1（K2 系列不支持 5 和 11 模式）					
模式	**描述**	**I0.4**	**I0.6**	**I0.3**	**I0.2**
0	带内部方向控制的单相增/减计数器				

1	方向控制位：SM47.3	复位		时钟	
2		复位	启动		
3	带外部方向控制的单相增/减计数器			时钟	方向
4		复位			方向
5		复位	启动		方向
6	带增/减计数时钟输入的双相计数器			时钟（减）	时钟（增）
7		复位			
8		复位	启动		
9	A/B 相正交计数器			时钟 A	时钟 B
10		复位			
11		复位	启动		

HSC 2（K5 系列不支持此通道）			
模式	**描述**	**I0.4**	**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
9	A/B 相正交计数器	时钟 B 相	时钟 A 相

HSC 3（K5 系列不支持此通道）			
模式	**描述**	**I0.6**	**I0.7**
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
9	A/B 相正交计数器	时钟 B 相	时钟 A 相

6.13.3.3 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6-1和图6-2适用于所有具有复位输入信号和启动输入信号的工作模式。图6-3至图6-5描述了高速计数器各个工作模式的时序图。

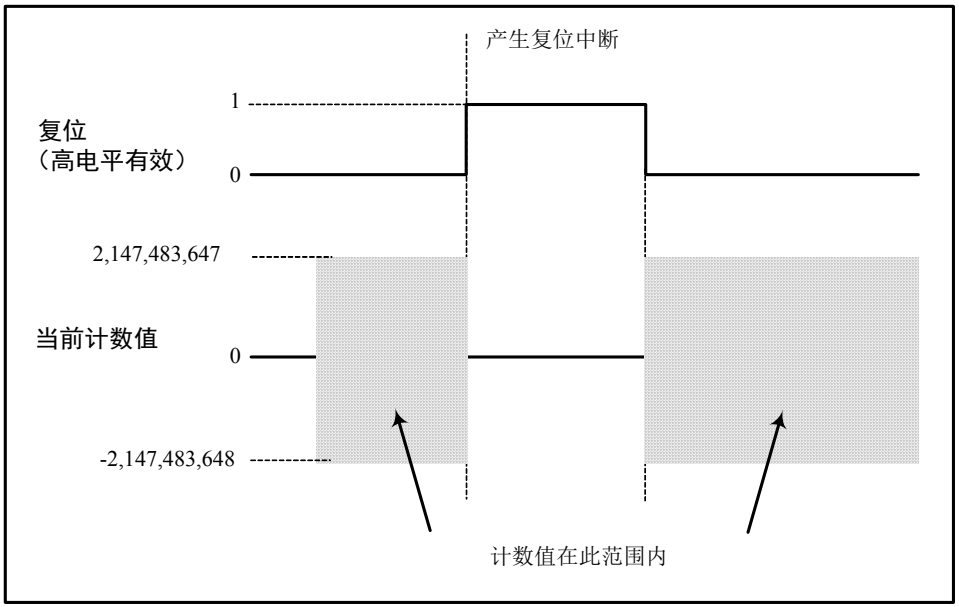


图 6-2 有复位信号无启动信号的高速计数器时序图

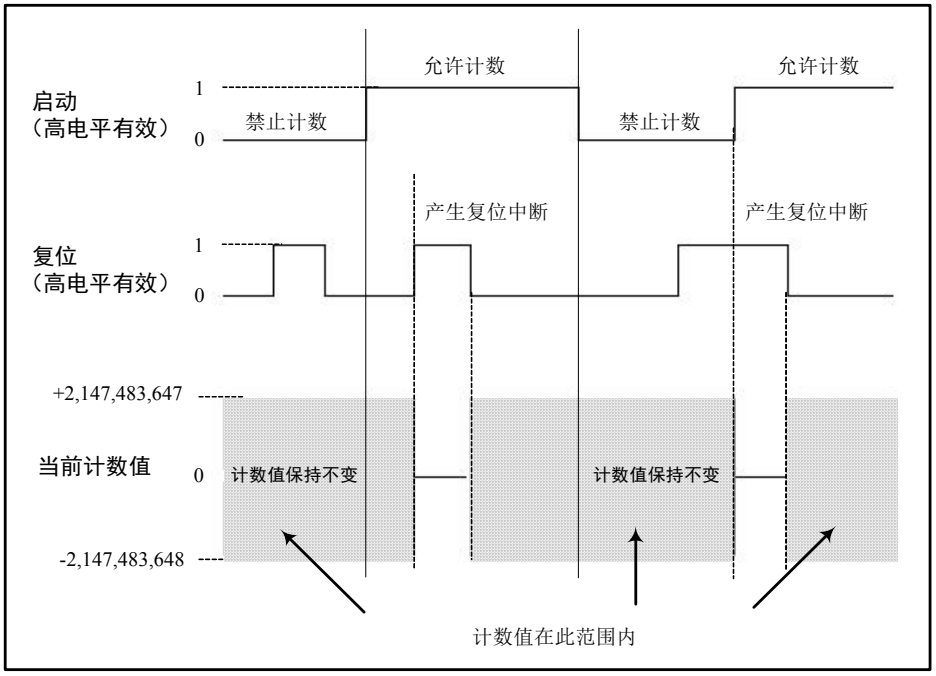


图 6-3 有复位信号和启动信号的高速计数器时序图

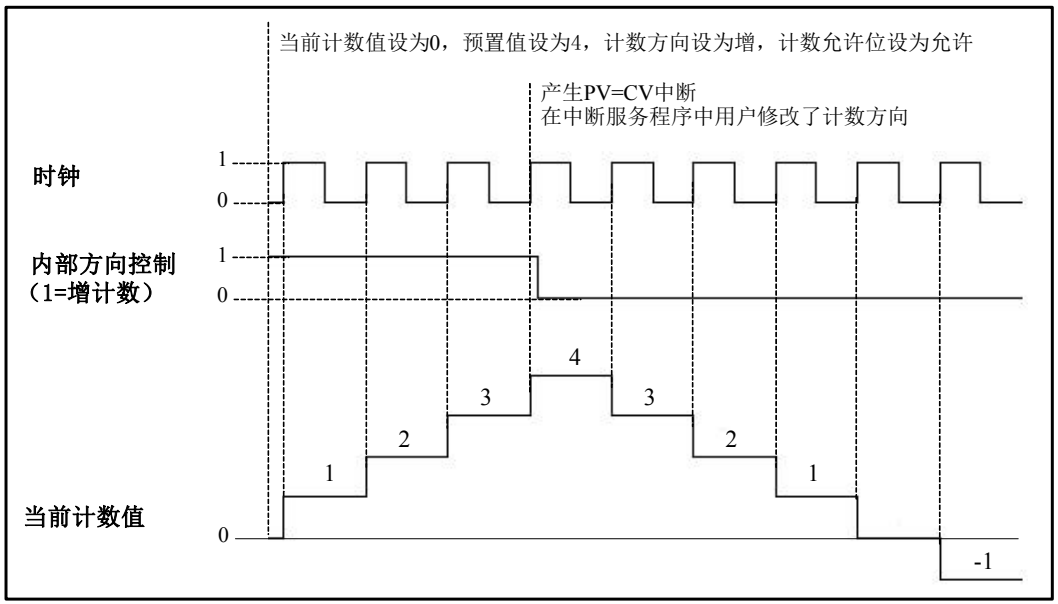


图 6-4 模式 0、1、2 的时序图

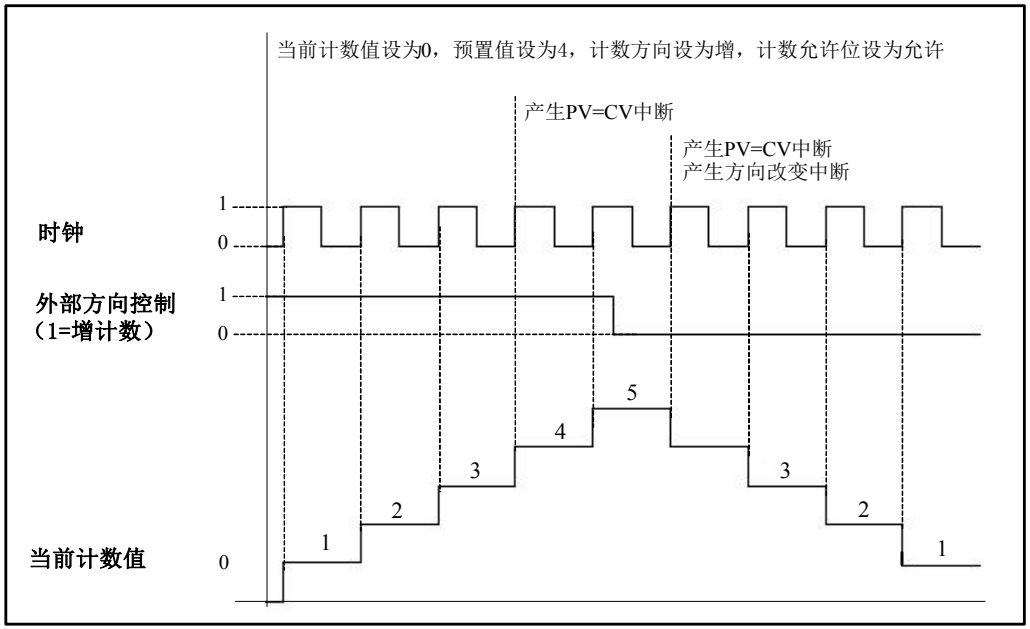


图 6-5 模式 3、4、5 的时序图

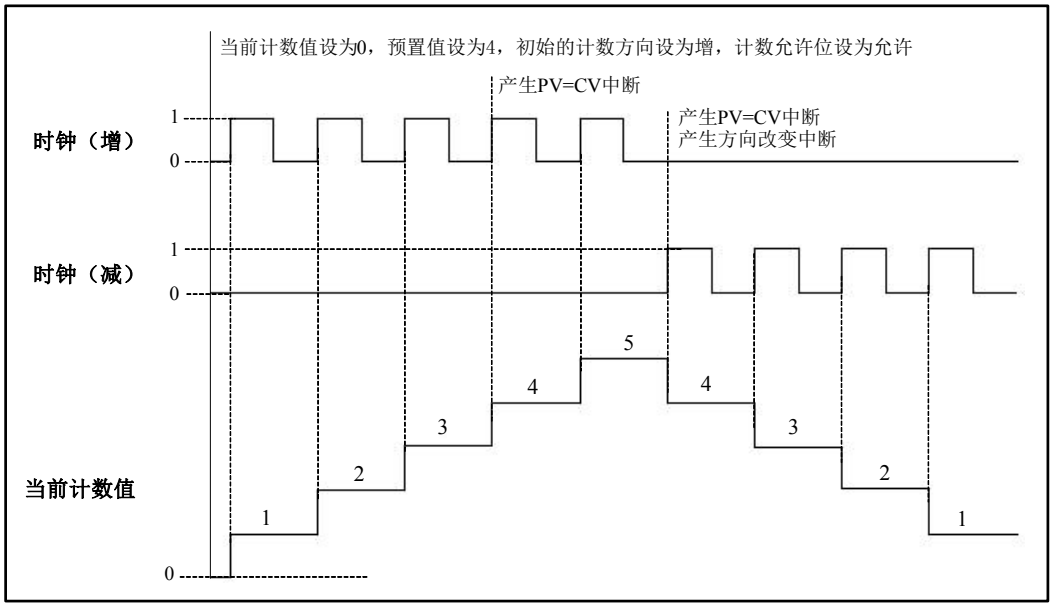


图 6-6 模式 6、7、8 的时序图

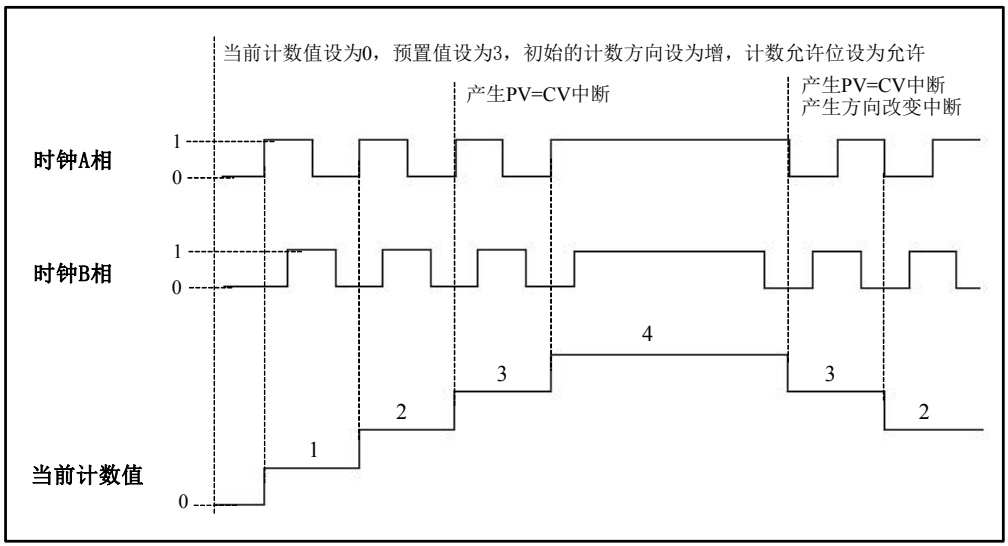


图 6-7 模式 9、10、11 的时序图（正交，1x）

6.13.3.4 控制寄存器和状态寄存器

需要说明的是，K5 系列不支持 HSC2 和 HSC3，所以下表中的相对应寄存器也对 K5 系列无效

➢ 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），当前值和预置值各一个（均为 32 位有符号双整数）。当前值指定计数的起始值，若将当前值写入高速计数器，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寄存器。

HSC0	HSC1	HSC2	HSC3	描 述
SM37.0	SM47.0	SM57.0	SM127.0	复位信号的有效电平：0=高电平；1=低电平
SM37.1	SM47.1	SM57.1	SM127.1	启动信号的有效电平：0=高电平；1=低电平
SM37.2	SM47.2	SM57.2	SM127.2	正交计数器速率：0=1x 速率；1=4x 速率*
SM37.3	SM47.3	SM57.3	SM127.3	计数方向：0=减计数；1=增计数。
SM37.4	SM47.4	SM57.4	SM127.4	是否向 HSC 中写入计数方向：0=否；1=是。
SM37.5	SM47.5	SM57.5	SM127.5	是否向 HSC 中写入新预置值：0=否；1=是。
SM37.6	SM47.6	SM57.6	SM127.6	是否向 HSC 中写入新当前值：0=否；1=是。
SM37.7	SM47.7	SM57.7	SM127.7	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	SMD128	当前值
SMD42	SMD52	SMD62	SMD132	预置值

另外 K2 系列 CPU 的所有高速计数器均允许指定最大 32 个预置值（PV），其控制寄存器描述如下：

HSC0	HSC1	HSC2	HSC3	描 述
SM141.0	SM151.0	SM161.0	SM171.0	是否使用多段预置值：0=否；1=是
SM141.1	SM151.1	SM161.1	SM171.1	预置值是相对值还是绝对值：0 =绝对；1=相对
SM141.2	SM151.2	SM161.2	SM171.2	预置值比较（“CV=PV”）中断是否循环产生： 0=否；1=是。 注意：只有相对值方式才允许设定为循环产生。
SM141.3	SM151.3	SM161.3	SM171.3	保留
SM141.4	SM151.4	SM161.4	SM171.4	是否更新段数及预置值：0=否；1=是
SM141.5	SM151.5	SM161.5	SM171.5	是否复位中断变量：0=是；1=否
SM141.6	SM151.6	SM161.6	SM171.6	保留
SM141.7	SM151.7	SM161.7	SM171.7	保留
HSC0	**HSC1**	**HSC2**	**HSC2**	**描 述**
SMW142	SMW152	SMW162	SMW172	预置值表的起始位置（用相对于 VB0 的字节偏移来表示），必须为奇数。

表 6-3 高速计数器的控制寄存器

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 和 2（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、当前值和预置值上电后的缺省值均为 0。

- 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（K2 系列还提供一个状态字），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。

HSC0	HSC1	HSC2	HSC3	描 述
SM36.0	SM46.0	SM56.0	SM126.0	保留
SM36.1	SM46.1	SM56.1	SM126.1	保留
SM36.2	SM46.2	SM56.2	SM126.2	保留
SM36.3	SM46.3	SM56.3	SM126.3	多段 PV 值表设置是否有错误：0=否，1=是。
SM36.4	SM46.4	SM56.4	SM126.4	保留
SM36.5	SM46.5	SM56.5	SM126.5	当前计数方向：0=减；1=增。
SM36.6	SM46.6	SM56.6	SM126.6	当前计数值是否等于预置值：0=否；1=是。
SM36.7	SM46.7	SM56.7	SM126.7	当前计数值是否大于预置值：0=否；1=是。
HSC0	HSC1	HSC2	HSC3	描 述（K5 系列不支持）
SMB140	SMB150	SMB160	SMB170	正在运行的 PV 值段序号（从 0 开始）。

表 6-4　高速计数器的状态寄存器

- 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。

Kinco-K 系列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 2.6.2.1 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 6-7　读取高速计数器的计数值示例

6.13.3.5 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。

K2 系列的每个 PV 值还支持“计数值=预置值”中断。PV 值可以指定为相对值或者绝对值方式，若选择为相对值方式，那么“计数值=预置值”中断允许选择循环发生。

请参阅 6.10 中断指令 了解更多信息。

6.13.3.6 使用高速计数器

➢ **方法一：使用相关指令进行编程**

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。

➢ **使用高速计数器**

下面将以模式 9 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，（1x 计数方式），SMB37 = B#16#F0 表明了：
   - 允许HSC0计数；
   - 允许向HSC0中写入新的当前值和预置值
   - 设置启动信号和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 9（代表采用模式 9）；
8) 执行 HSC 指令来配置并启动 HSC0。

➢ **改变计数方向（模式 0、1、2）**

下面描述了如何来改变 HSC0 的计数方向（模式 0、1、2）。

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 HSC0 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 HSC0 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）**

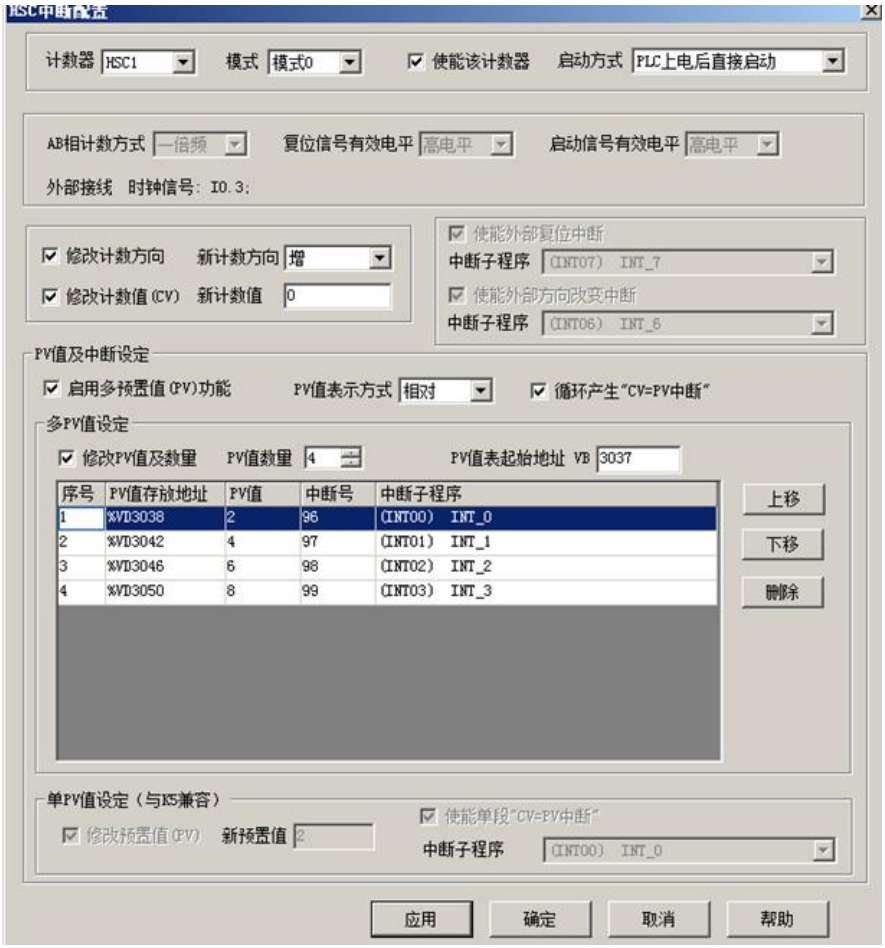
下面描述了如何禁止 HSC0。

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。

➢ **方法二：使用 HSC 向导（K5 系列不支持向导功能）**

在 K2 中，还为高速计数器提供了配置向导。用户可以直接利用该向导对所有的高速计数器进行配置，无需再进行复杂的编程。向导如下图。

即使通过向导对 HSC 进行了配置，用户也可以在程序中按照“方法一”来随时对高速计数器进行参数修改、启动、停止。



HSC 向导的使用方法如下：

1） 在【计数器】中，选择将要使用的计数器。

2） 选中【使能该计数器】，然后将会允许进行后续的配置。

3） 在【模式】中，选择将要使用的计数器模式。

4） 在【启动方式】中，选择该高速计数器的启动方式。

启动方式有如下两种：

“在程序中调用 HSC 指令”： 若选择这种方式，那么在用户程序中，通过调用 HSC 指令来启动该计数器。在调用 HSC 指令之前，无需再配置各寄存器和调用 HDEF 指令。

“PLC 上电后直接启动”：若选择这种方式，那么该高速计数器在 PLC 上电后就自动运行，无需调用任何指令。

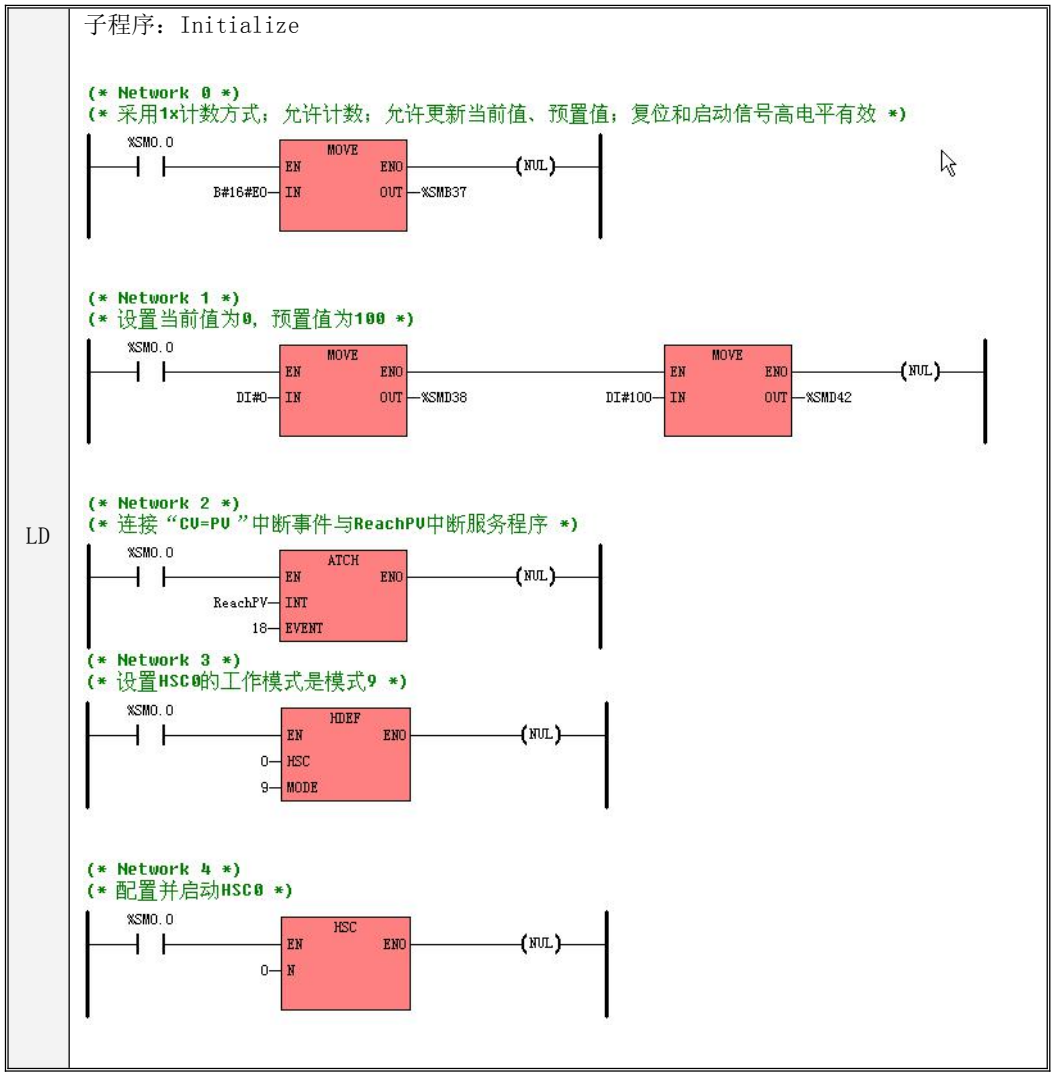
5） 若要使用多段 PV 值方式，则选中【启用多预置值（PV）功能】，然后可以对 PV 值、数量、关联的中断子程序等进行配置。若选中【修改 PV 值及数量】，则可以调整【PV 值数量】中的数值，从而修改 PV 值个数。

6） 若要使用单 PV 值方式，则首先选中“单 PV 值设定（与 K5 兼容）”中的【修改 PV 值】，然后可以修改 PV 值及关联的中断子程序。

7） 其它的配置项，请参考前文的描述，按实际需求进行配置。

6.13.3.7 高速计数器示例

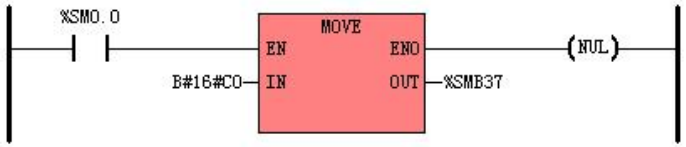
下面还是以 HSC0 为例来进行编程。



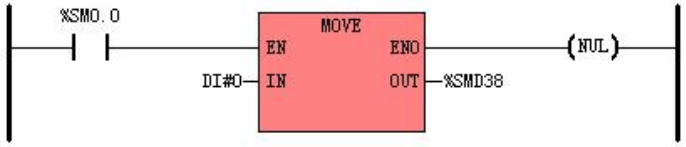
LD

中断服务程序：ReachPV

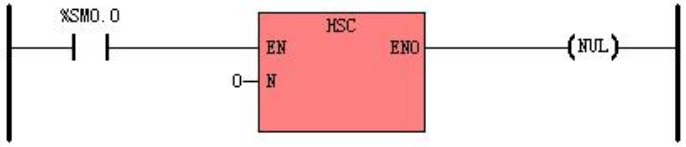
```
(* Network 0 *)
(* 允许HSC0计数；允许写入新的当前值。 *)
```




```
(* Network 1 *)
(* 将当前值设置为0，也就是让HSC0从0开始重新计数 *)
```

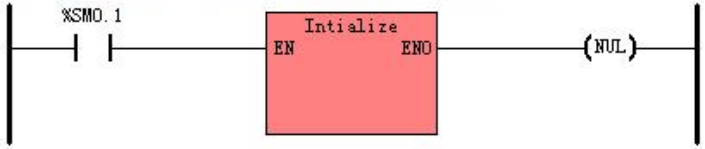



```
(* Network 2 *)
(* 配置HSC0 *)
```



主程序：MAIN

```
(* Network 0 *)
(* 调用一次HSC0的初始化子程序即可 *)
```



IL

子程序：Initialize

```
(* Network 0 *)
LD      %SM0.0
MOVE    B#16#E0, %SMB37          (* 配置控制字节 *)
MOVE    DI#0, %SMD38             (* 设置当前值为 0 *)
MOVE    DI#100, %SMD42           (* 设置预置值为 100 *)
ATCH    ReachPV, 18              (* 连接“CV=PV”中断与 ReachPV 中断程序 *)
HDEF    0, 9                     (* 设置 HSC0 的工作模式是模式 9 *)
HSC     0                        (* 配置并启动 HSC0 *)
```

中断服务程序：ReachPV

```
(* Network 0 *)
LD      %SM0.0
MOVE    B#16#C0, %SMB37          (* 允许 HSC0 计数；允许更新当前值 *)
MOVE    DI#0, %SMD38             (* 将当前值设置为 0 *)
HSC     0                        (* 配置 HSC0 *)
```

主程序：MAIN

```
LD      %SM0.1
CAL     Intialize                (* 调用一次 HSC0 的初始化子程序即可 *)
```

6.13.4 PLS（PTO 或者 PWM 输出）

PLS 指令可以实现 PTO 或者 PWM 输出功能。

- PTO：Pulse Train Output，脉冲串输出。
- PWM：Pulse-Width Modulation，脉宽调制。

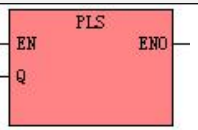
K5 支持 2 路高速脉冲输出，相应地就提供了 2 个 PTO/PWM 脉冲发生器用于产生 PTO/PWM 输出。其中，第一个脉冲发生器分配在 Q0.0，称为 PWM0 或者 PTO0；第二个分配在 Q0.1，称为 PWM1 或者 PTO1 。K5 脉冲输出的最高频率为 200kHz。

K2 支持 3 路高速脉冲输出，其中前 2 路与 K5 使用的通道一致，第 3 路分配在 Q0.4，称为 PWM2 或者 PTO2。K2 的通道 Q0.0、Q0.1 最高输出频率 50KHz，通道 Q0.4 最高输出频率 10KHz。

PTO/PWM 发生器和 DO 映像寄存器共同使用内存地址 Q0.0、Q0.1（或 Q0.4）。如果用户程序中调用了某个通道的高速输出指令，那么 PTO/PWM 发生器将控制输出通道，并禁止普通 DO 的输出。

注意：若Q0.0、Q0.1或者Q0.4是继电器类型的，则避免使用高速脉冲输出功能！

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PLS			☑ K5 ☑ K2
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、1 或者 2）

PLS 指令的作用是：读取 SM 区中相应控制寄存器的值并配置高速脉冲输出的特性，然后启动高速脉冲输出，直到完成指定的脉冲输出功能。脉冲输出通道由参数 *Q* 指定，0 表示使用 Q0.0 输出，1 表示使用 Q0.1 输出，2 表示使用 Q0.4 输出。

注意：用户程序中，仅在需要时执行一次 PLS 指令即可，建议利用边沿指令的输出结果来调用 PLS 指令。若 PLS 的 EN 端一直保持为 1，那么 PLS 指令将无法正常输出。

- 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。

6.13.4.1 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），最大周期值为 65535。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。

- **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），最大周期值为 65535。脉冲个数的范围是：2～4,294,967,295。如果指定脉冲数小于 2，则 PLC 将设置相应的错误标志位并禁止输出。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。

- **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 8 个字节，包括一个周期值（16 位无符号整数）、保留值（暂时未用到，16 位符号整数）和一个脉冲个数值（32 位无符号双整数）。也就是说，在同一段中，所有脉冲的输出频率是相同的。多段操作使用 PLS 指令来配置并启动。

包络表的起始位置存储在 SMW168（对应 PTO0）、SMW178（对应 PTO1）、SMW268（对应 PTO2）中，时基通过 SM67.3（对应 PTO0）、SM77.3（对应 PTO1）、SM87.3（对应 PTO1）设置，可以选择微秒或毫秒。包络表中的所有周期值必须使用同一个时基，并且在包络执行时不能改变。

包络表的格式如下表所示。

字节偏移 (1)	长 度	段 数	描 述
0	8 位		段数（1 到 64）
1	16 位	第 1 段	初始周期（2 到 65535 时基）
3	16 位		保留
5	32 位		脉冲个数（1 到 4,294,967,295）
9	16 位	第 2 段	初始周期（2 到 65535 时基）
11	16 位		保留
13	32 位		脉冲数（1 到 4,294,967,295）
…		…	…

（1）所有偏移量均是相对于包络表起始位置的偏移字节数。

注意：包络表的起始位置必须为 V 区中的奇数地址，如 VB3001。

6.13.4.2 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。如下表。

Q0.0	Q0.1	Q0.4	描 述	
SM67.0	SM77.0	SM97.0	PTO/PWM	是否更新周期值：0=否；1=是
SM67.1	SM77.1	SM97.1	PWM	是否更新脉宽值：0=否；1=是
SM67.2	SM77.2	SM97.2	PTO	是否更新脉冲个数：0=否；1=是
SM67.3	SM77.3	SM97.3	PTO/PWM	时基：0=1μs；1=1ms
SM67.4	SM77.4	SM97.4	PWM	更新方法：0=异步更新；1=同步更新
SM67.5	SM77.5	SM97.5	PTO	操作方式：0=单段操作；1=多段操作
SM67.6	SM77.6	SM97.6	功能选择：0= PTO；1=PWM	
SM67.7	SM77.7	SM97.7	PTO/PWM	允许或禁止此功能：0=禁止；1= 允许
Q0.0	Q0.1	Q0.4	描 述	
SMW68	SMW78	SMW98	PTO/PWM	周期值，范围 2~65535
SMW70	SMW80	SMW100	PWM	脉宽值，范围 0~65535
SMD72	SMD82	SMD102	PTO	脉冲个数，范围 1~4,294,967,295
SMW168	SMW178	SMW218	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。	

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，包络表也得先设置好，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。如下表。

Q0.0	Q0.1	Q0.4	描 述
SM66.0	SM76.0	SM96.0	保留
SM66.1	SM76.1	SM96.1	保留
SM66.2	SM76.2	SM96.2	保留
SM66.3	SM76.3	SM96.3	PWM 是否空闲：0=否；1=是
SM66.4	SM76.4	SM96.4	PTO 周期值、脉冲个数设置是否有错误：0=否；1=是 **注：周期值、脉冲个数值必须大于 1。**
SM66.5	SM76.5	SM96.5	PTO 是否由于用户命令而终止：0=否；1=是
SM66.6	SM76.6	SM96.6	保留
SM66.7	SM76.7	SM96.7	PTO 是否空闲：0＝忙；1＝空闲

PTO 空闲位、PWM 空闲位指明了 PTO 输出、PWM 输出是否已经结束。

6.13.4.3 使用 PTO 功能

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#85 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；
- 允许更新脉冲个数和周期值。

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。

➢ **改变 PTO 周期（单段操作）**

按照如下步骤来改变 PTO0 周期值：

1) 根据期望的操作来设置控制字节 SMB67:
例如，SMB67 = B#16#81 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；
- 允许更新周期值。

2) 将期望的周期值赋给 SMW68。
3) 执行 PLS 指令来配置并启动 PTO0，具有新周期值的 PTO 就会立即接着启动。

➢ **改变 PTO 脉冲个数（单段操作）**

按照如下步骤来改变 PTO0 输出的脉冲个数：

1) 根据期望的操作来设置控制字节 SMB67:
例如，SMB67 = B#16#84 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；

- 允许更新脉冲个数。

2) 将期望的脉冲个数赋给 SMD72。
3) 执行 PLS 指令来配置并启动 PTO0，就会立即接着输出新指定个数的脉冲。

➢ **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) 将包络表的起始位置（奇数，表示包络表起始地址相对于 VB0 的字节偏移）赋给 SMW168。
3) 设置包络表中的相关数值。
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。

6.13.4.4 使用 PWM 功能

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；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#D3 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；
- 允许更新脉宽值和周期值。

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。

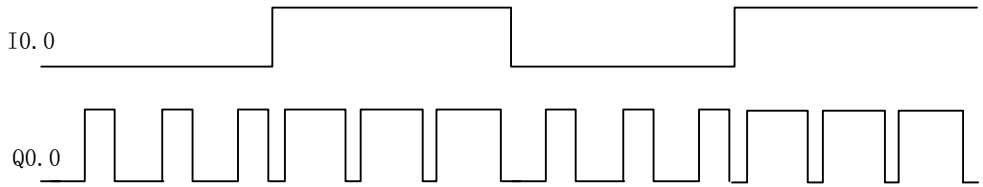
2) 将期望的脉宽值赋给 SMW70。
3) 执行 PLS 指令来配置并启动 PWM0。

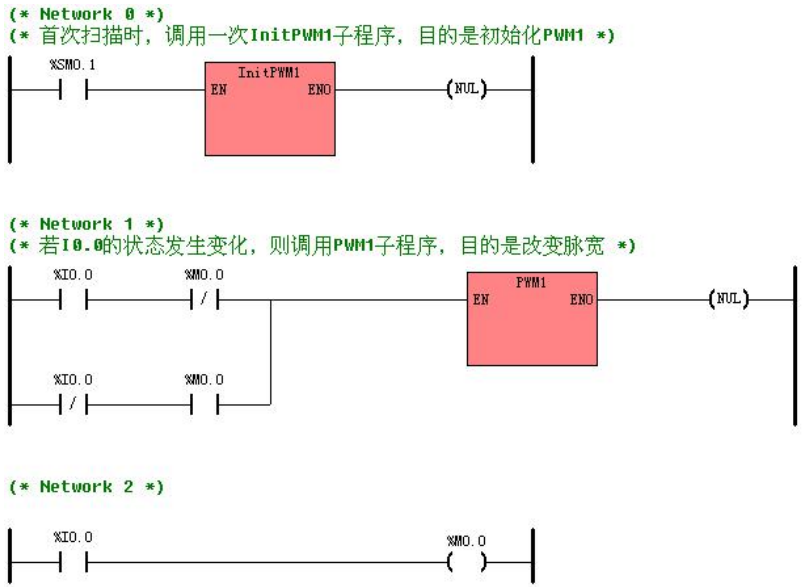
6.13.4.5 示例

➢ PWM 示例

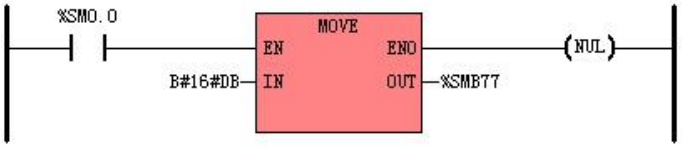
示例中使用了 PWM1，在 Q0.1 输出。周期为 10ms。

通过 I0.0 信号来改变输出脉冲的占空比，若 I0.0 为 0，则占空比为 40%；若 I0.0 为 1，则占空比为 80%。时序的示意图如下：

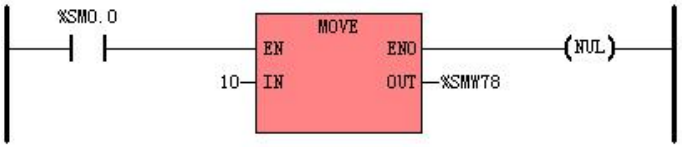


LD	主程序 MAIN： 
LD	子程序 InitPWM1：

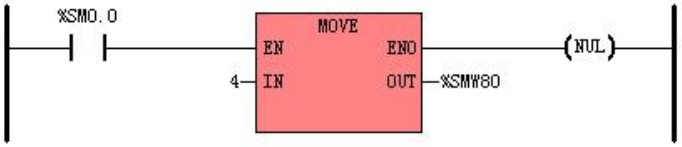
```
(* Network 0 *)
(* 选用PWM1，其时基为1ms，允许更新脉冲周期值、宽度值 *)
```



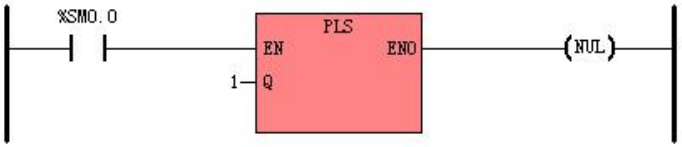

```
(* Network 1 *)
(* 设定PWM1的脉冲周期为10ms *)
```




```
(* Network 2 *)
(* 设定PWM1的脉宽为4ms *)
```



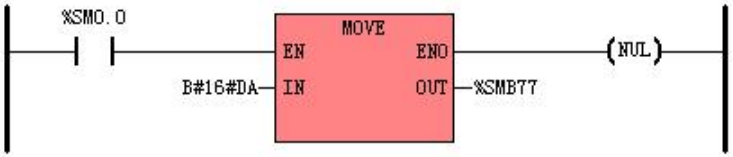

```
(* Network 3 *)
(* 配置并执行PWM1 *)
```



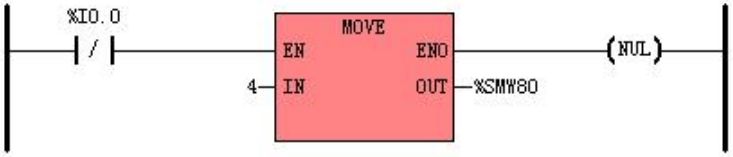
LD

子程序 PWM1：

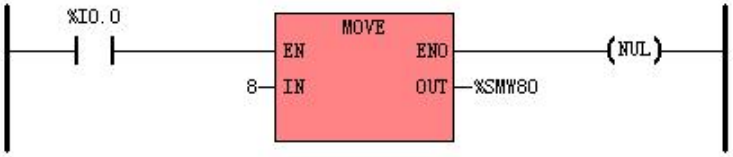
```
(* Network 0 *)
(* 选用PWM1，其时基为1ms，允许更新脉宽值 *)
```



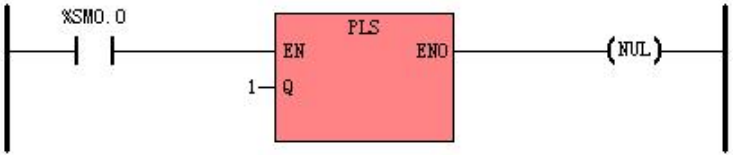

```
(* Network 1 *)
(* 若I0.0为0，则设定PWM1的脉宽为4ms *)
```




```
(* Network 2 *)
(* 若I0.0为1，则设定PWM1的脉宽为8ms *)
```




```
(* Network 3 *)
(* 配置PWM1以修改脉宽 *)
```



IL

主程序 MAIN：

```
(* NETWORK 0 *)
LD      %SM0.1
CAL     InitPWM1            (* CPU 启动时调用一次 InitPWM1 子程序，初始化 PWM1 *)
LD      %I0.0
ANDN    %M0.0               (* 在 M0.0 中存放着上一次扫描时 I0.0 的值 *)
ST      %M0.1
LDN     %I0.0
AND     %M0.0
OR      %M0.1
CAL     PWM1                (* 若 I0.0 的值发生变化，则调用 PWM1 子程序 *)
LD      %I0.0
ST      %M0.0               (* 将 I0.0 的状态存储于 M0.0 中 *)
```

子程序 InitPWM1：

```
(* NETWORK 0 *)
LD      %SM0.0
MOVE  B#16#DB, %SMB77       (* 选用 PWM1，设定其时基为 1ms 并允许其操作 *)
MOVE  10, %SMW78            (* 设定 PWM1 的脉冲周期为 10ms *)
MOVE  4, %SMW80             (* 设定 PWM1 的脉宽度为 4ms *)
PLS    1                    (* 执行 PWM1，使用 Q0.1 输出 *)
```

子程序 PWM1：

```
(* NETWORK 0 *)
LD      %SM0.0
MOVE   B#16#DA, %SMB77      (* 选用 PWM1，其时基为 1ms，允许更新脉宽值 *)
LDN     %I0.0               (* 若 I0.0 为 0  *)
MOVE   4, %SMW80            (* 则设定 PWM1 的脉宽为 4ms *)
LD      %I0.0               (* 若 I0.0 为 1  *)
MOVE   8, %SMW80            (* 则设定 PWM1 的脉宽为 8ms *)
LD      %SM0.0
PLS     1                   (* 配置 PWM1 以修改脉宽 *)
```

6.14 定时器

定时器是 IEC61131-3 标准中定义的功能块之一，共有 TON、TOF 和 TP 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 一节。

6.14.1 定时器的时基

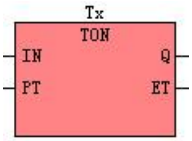
K 系列提供了三种时基的定时器，定时器号决定了其时基，其中 T0-T3 的时基为 1ms，T4-T19 的是 10ms，T20-T255 的是 100ms。

定时器的最大定时时间为 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。

只有定时器指令正在执行时，PLC 才对定时器的 ET 值进行更新，因此会受到扫描周期的影响。如需精确定时，请使用定时中断

6.14.2 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			☑ K5 ☑ K2
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

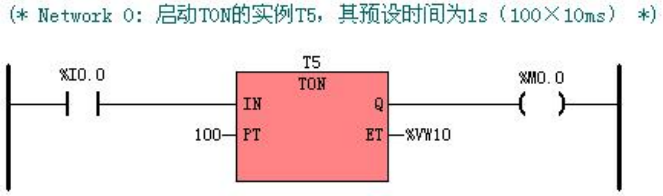
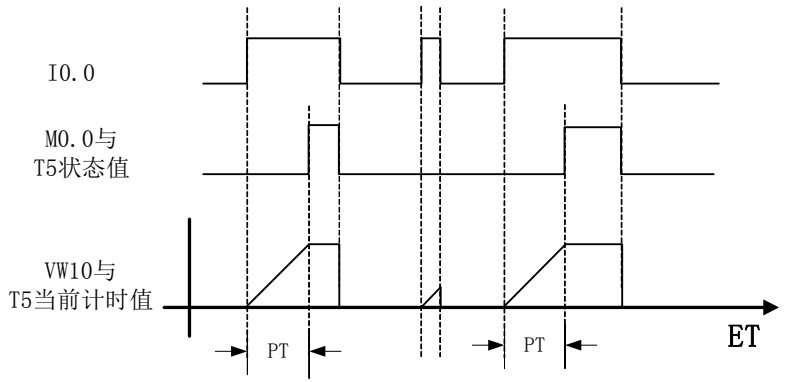
- LD

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。

➢ TON 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* NETWORK 0 *)
LD %I0.0
TON T5, 100
ST %M0.0</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.14.3 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			☑ K5 ☑ K2
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

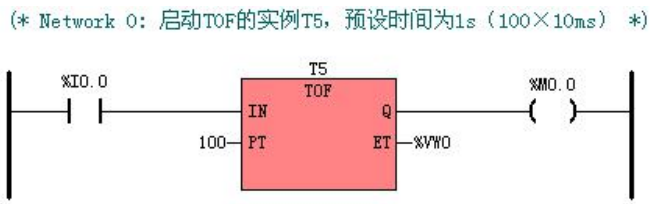
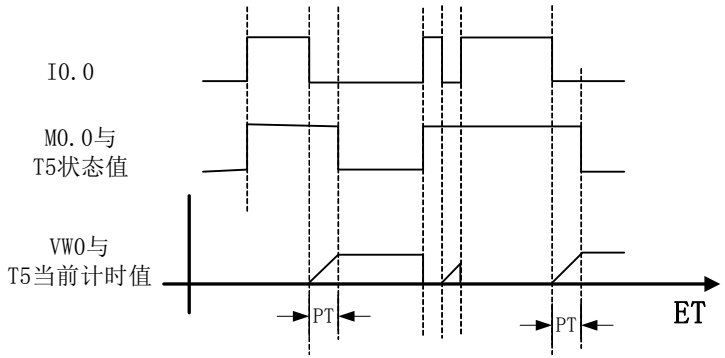
- LD

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

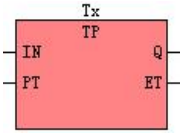
若检测到 CR 值的下降沿，则 *Tx*开始启动定时，当计时值大于等于预设值 *PT*时，*Tx*停止，其状态值被置为 0。若 CR 值变为 1，则 *Tx*被复位，其状态值被置为 1，且计时值被清零。每次扫描 *TOF*后，CR 值均被设置为 *Tx*的状态值。

➢ TOF 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* NETWORK 0 *)
LD %I0.0
TOF T5, 100
ST %M0.0</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

6.14.4 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			☑ K5 ☑ K2
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

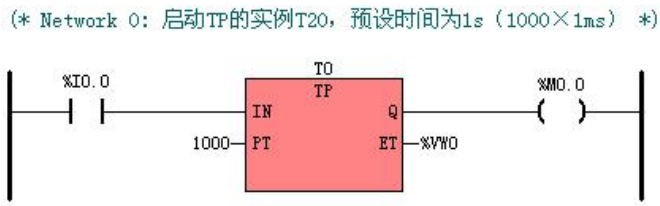
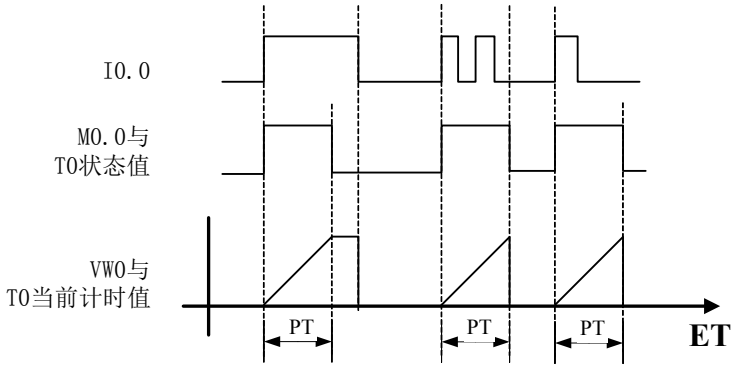
- LD

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。

➢ TP 使用举例

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td> 
</td><td>(* NETWORK 0 *)
LD %I0.0
TP T0, 1000
ST %M0.0</td></tr>
<tr><td colspan="2">时序图</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

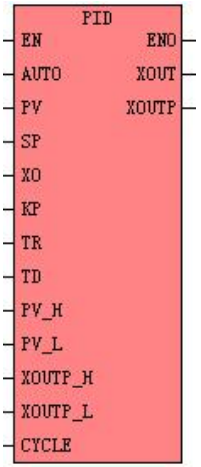
6.15 PID 回路

PID 指令是一种数字 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。

注意，PID 运算所需时间较长，建议避免在中断程序中调用 PID 功能。

6.15.1 PID

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	
LD	PID			☑ K5 ☑ K2
IL	PID	PID AUTO, PV, SP, XO, KP, TR, 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	PV 值，即被控量的测量值。
SP	输入	INT	V	设定值，即被控量的目标值。
XO	输入	REAL	V	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	比例系数
TR	输入	REAL	V	积分时间，单位：秒。0 表示无积分。
TD	输入	REAL	V	微分时间，单位：秒。0 表示无微分。
PV_H	输入	INT	V	*PV* 值的上限
PV_L	输入	INT	V	*PV* 值的下限
XOUTP_H	输入	INT	V	XOUTP 值的上限
XOUTP_L	输入	INT	V	XOUTP 值的下限
CYCLE	输入	DINT	V	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	PID 的输出值，范围是 0.0～1.0。
XOUTP	输出	INT	AQ、V	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

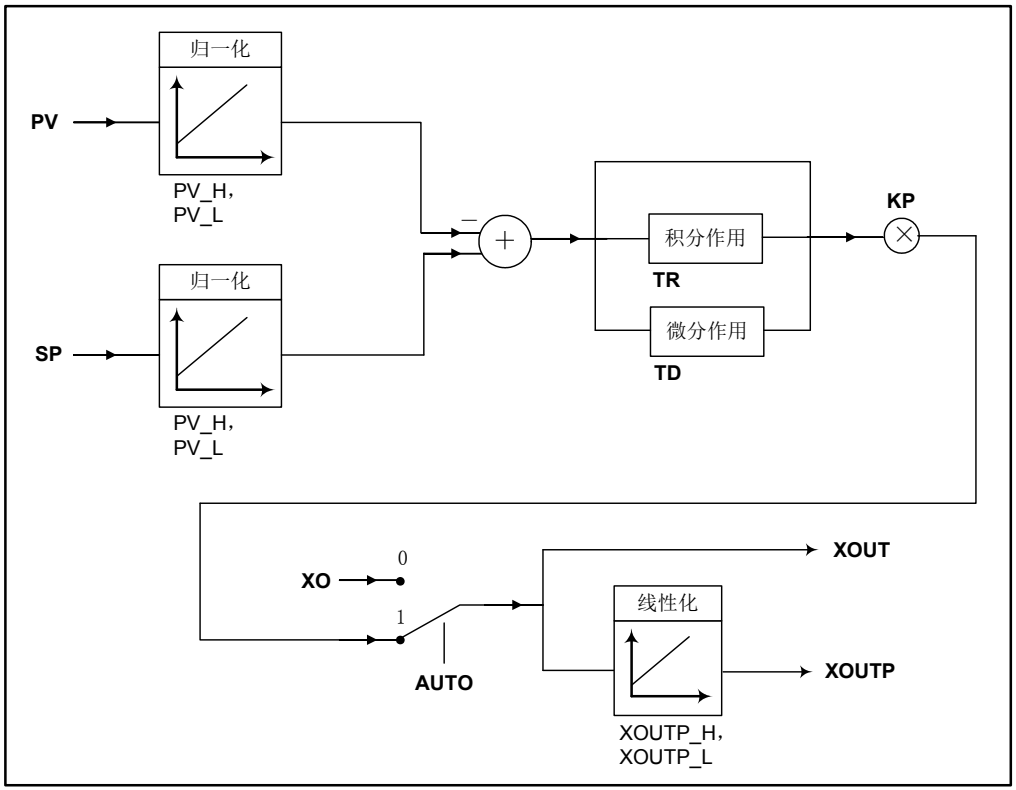
比例作用是按照比例来反映被控量的偏差，一旦出现偏差，比例调节器将立即产生调节作用来减少偏差。用户通过 *KP* 参数来设置 PID 的比例系数。若 *KP* 大于 0，则 PID 是正作用；若 *KP* 小于 0，则 PID 是反作用。比例系数大，则调节速度快，但容易造成超调，使系统的稳定性下降。

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 *TR* 参数来设置 PID 的积分时间常数（若 *TR* 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



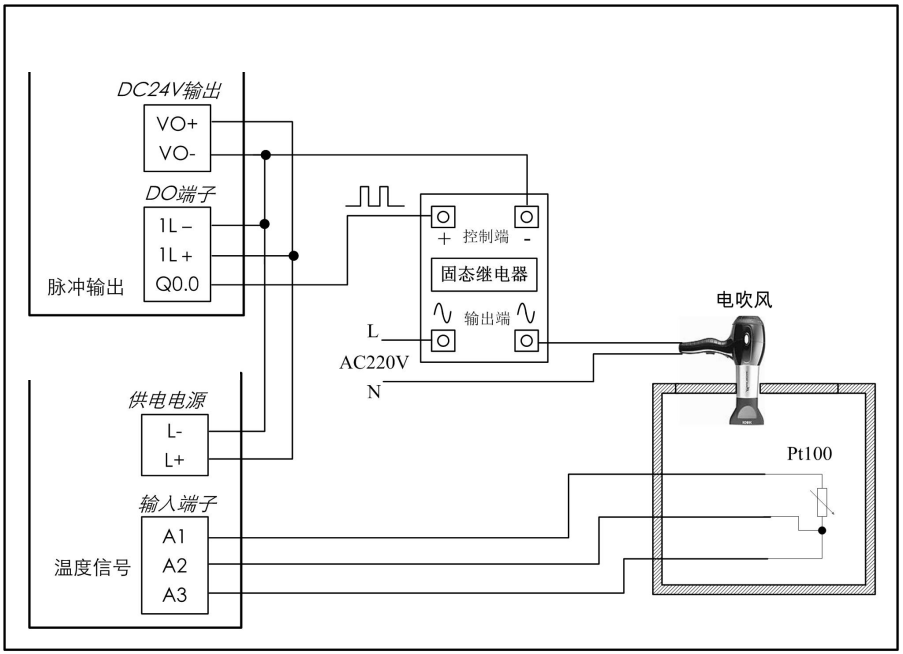
➢ PID 使用举例

我们建立如下模拟系统：对一个箱子内的温度进行控制，升温依靠电吹风加热，降温依靠自然冷却。

PLC 的配置是：K506-24AT+K531-04RD。另外，SP、KP、TR、TD 等参数的修改通过一个 HMI 来完成。

在这个系统中，使用一个 Pt100 来测量温度，其信号接入 RTD 模块的 AIW0 通道。Q0.0 输出脉冲串并接至固态继电器的控制端，利用 PID 的输出来相应调整脉冲宽度，从而控制电吹风的通断时间，实现温度的控制。

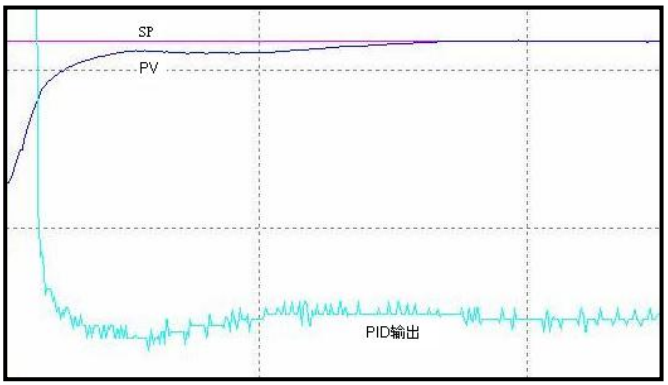
模拟系统如下图所示：



下面简单描述一下示例的编程思路：

1) 利用一个定时器实现了简单的脉宽调制输出。

2) 在程序中使用了如下控制方式：若PV值低于SP值超过5℃，则使用PID手动方式并输出100%；若PV值与SP值之间的偏差小于5℃，则使用PID自动方式。这种方式的优点是：在偏差较大时使用PID手动方式，避免了自动方式下由于积分累积值过大而造成的超调很大的问题；另外，也有利于减小调节时间。当SP设定为50℃时，控制效果如下图。



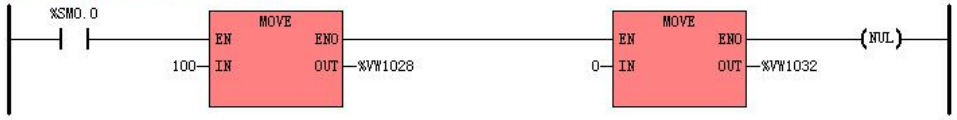
程序如下：

LD

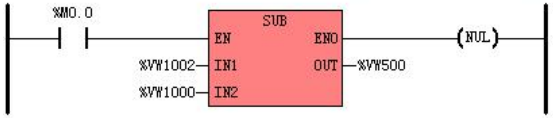
```
(* Network 0 *)
(* 设定PV值、采样周期。
   采样周期是一个能够影响控制效果的重要参数，需要根据被控对象的实际特性来设定。 *)
```


```
(* Network 1 *)
(* PV的最大、最小值 *)
```

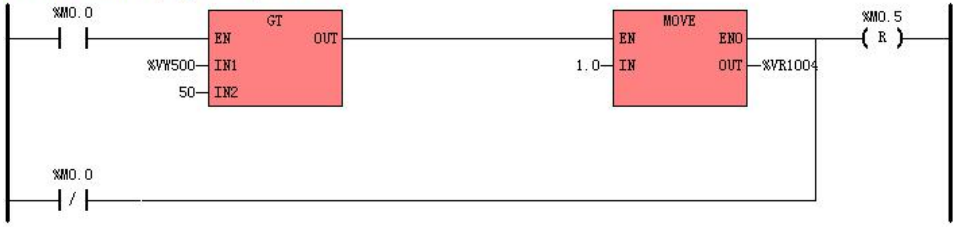

```
(* Network 2 *)
(* XOUT_P的最大、最小值 *)
```




```
(* Network 3 *)
(* M0.0---HMI上的启动/停止自动控制按钮。
   控制方式：若PV值低于SP值超过5℃，则使用PID手动方式并输出100%；
             若PV值与SP值之间的偏差小于5℃，则使用PID自动方式。 *)
```

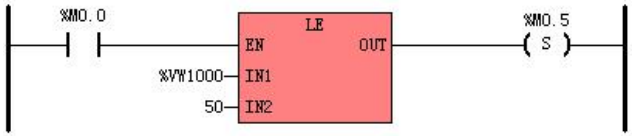



```
(* Network 4 *)
(* M0.5---PID手/自动方式参数 *)
```

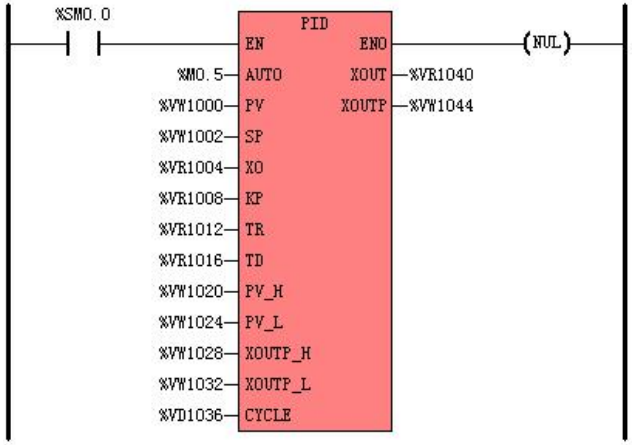


LD

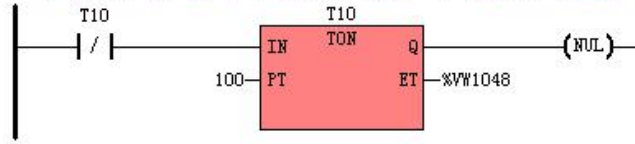
(* Network 5 *)



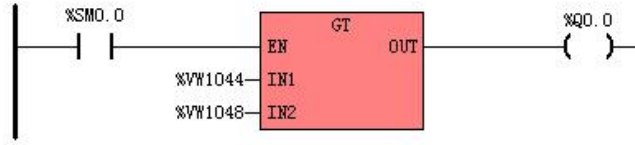
(* Network 6 *)
(* SP、X0、KP、TR、TD等参数需要通过人机界面来设定。 *)



(* Network 7 *)
(* 周期为1秒的脉冲串输出，利用PID的输出来调整脉冲宽度。 *)



(* Network 8 *)



IL

```
(* Network 0 *)
(*设定 PV 值、采样周期。*)
(*采样周期是一个能够影响控制效果的重要参数，需要根据被控对象的实际特性来设定。*)
LD          %SM0.0
MOVE     %AIW0, %VW1000
MOVE     DI#1000, %VD1036
(* Network 1 *)
(*PV 的最大、最小值*)
LD          %SM0.0
MOVE     1000, %VW1020
MOVE     0, %VW1024
(*XOUT_P 的最大、最小值*)
MOVE     100, %VW1028
MOVE     0, %VW1032
(* Network 3 *)
(*M0.0---HMI 上的启动/停止自动控制按钮。*)
(*控制方式：若 PV 值低于 SP 值超过 5℃，则使用 PID 手动方式并输出 100%；*)
(* 若 PV 值与 SP 值之间的偏差小于 5℃，则使用 PID 自动方式。*)
LD          %M0.0
MOVE     %VW1002, %VW500
SUB        %VW1000, %VW500
(* Network 4 *)
(*M0.5---PID 手/自动方式参数*)
LD          %M0.0
GT          %VW500, 50
MOVE     1.0, %VR1004
ORN        %M0.0
R           %M0.5
(* Network 5 *)
LD        %M0.0
LE        %VW1000, 50
S         %M0.5
(* Network 6 *)
(*SP、X0、KP、TR、TD 等参数需要通过人机界面来设定。*)
LD        %SM0.0
PID       %M0.5, %VW1000, %VW1002, %VR1004, %VR1008, %VR1012, %VR1016, %VW1020, %VW10
          24, %VW1028, %VW1032, %VD1036, %VR1040, %VW1044
(* Network 7 *)
(*周期为 1 秒的脉冲串输出，利用 PID 的输出来调整脉冲宽度。*)
LDN       T10
TON       T10, 100
(* Network 8 *)
LD        %SM0.0
GT        %VW1044, T10
ST        %Q0.0
```

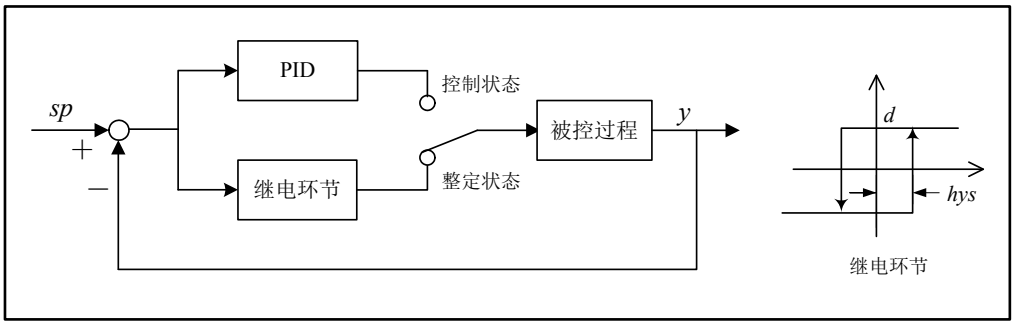
6.15.2 PID 自整定

K5 提供了 PID 自整定功能，并且支持 8 路 PID 同时进行自整定。自整定适用于 PID、PI、PD、P 等各种结构的 PID 回路，并且 PID 的作用方式允许是正作用或者反作用。

6.15.2.1 综述

Kinco-K5 采用了基于继电反馈的参数自整定方法。这种方法由 Astrom 和 Hagglundto 在 1984 年提出，克服了传统 Z-N 法的一些弊端，并且简便可靠，因此得到了广泛的应用。

继电自整定是一种闭环自整定方法，在自整定过程中，系统始终处于受控的闭环状态，其原理如下图所示。



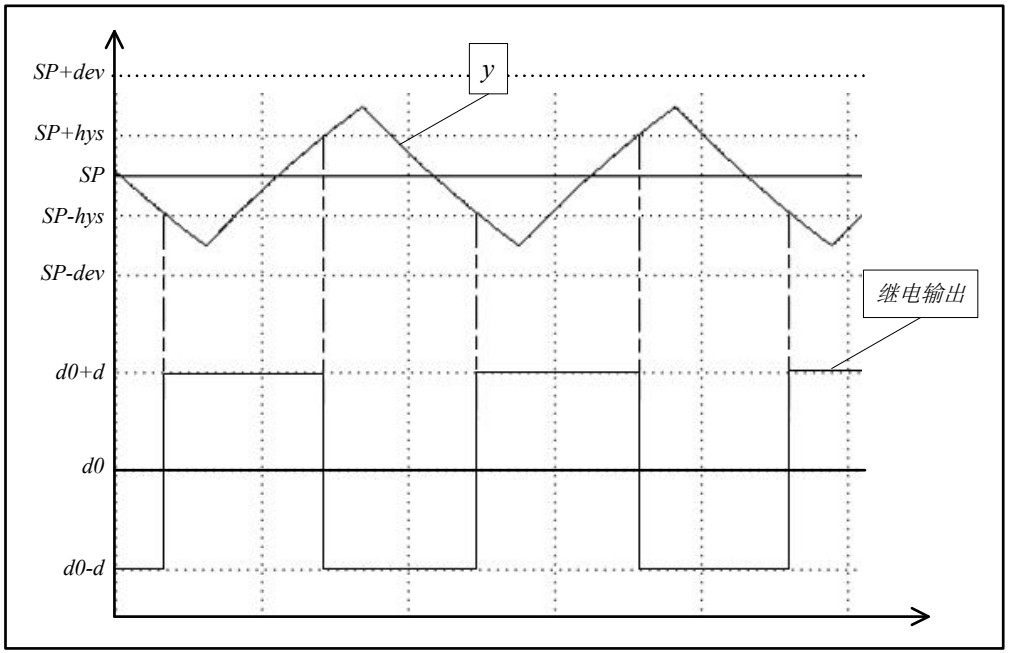
在开始自整定之前，用户需要先将系统置于 PID 自动控制状态，当 PV 值达到 SP 值并围绕 SP 形成振荡后，就可以将系统切向整定状态，PLC 就启动了 PID 自整定程序。由于在闭环中接入了具有滞环的继电环节，从而使系统产生了极限环周期振荡，PLC 通过测量这个极限环振荡的信息来计算出被控过程的临界增益和临界周期，再进一步计算出不同结构的 PID 参数，从而达到自整定的目的。

自整定计算出的 PID 参数与用户指定的动态响应类型有关，包括快速响应、中速响应、慢速响应和极慢速响应。快速响应系统会产生超调，对应于欠阻尼整定状态；中速响应系统可能会使控制过程处于产生超调的边界状态，对应于临界阻尼状态；慢速和超慢速响应系统不会产生超调，分别对应于过阻尼和严重过阻尼整定状态。

继电自整定方法适用于一阶惯性加纯滞后的被控对象（典型的比如温度），如果对其它类型的对象使用这种方法，则可能得不到较好的 PID 参数。对于时间常数比较大的对象，自整定过程将耗时比较长。另外，对于一些干扰因素较多且比较频繁的系统，在整定过程中就要求振荡的幅度足够大，严重时可能影响形成稳定的极限环振荡，从而造成本次自整定的失败。

6.15.2.2 继电环节的参数与极限环振荡

下图简要描述了如何利用继电环节形成极限环振荡。



图中，*d0*、*d* 分别是继电环节输出的初始幅值、变化量。*hys* 是继电环节滞环的宽度，称为滞后值。滞后值的作用是减少自整定过程中干扰的影响，它实质上是过程值相对于 SP 值的一个死区范围，若过程值相对于 SP 的偏差在这个范围之内，则不会引起控制器改变输出。另外，*dev* 是我们期望的过程值的最大偏差，称为偏差值。在自整定过程中，控制器会适当调整输出，使过程的振荡保持在这一范围之内。

从图中可以看出，继电输出值的改变会引起过程值的变化，当过程值达到 SP 值然后超出了滞后值区域时，PLC 就会反向调整继电输出值，这样最终会促使 PV 值向相反的方向变化，如此反复进行，就会形成一个稳定的极限环振荡，PLC 将利用包含在这个极限环中的信息来计算出被控过程的临界参数。

6.15.2.3 使用 PID 自整定

在使用之前，请仔细阅读上一节 继电环节的参数与极限环振荡，以了解一些关键参数的作用。下面将详细介绍使用 PID 自整定的步骤。

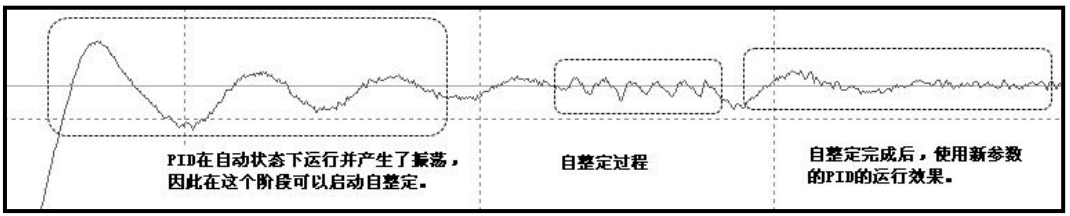
① 首先，用户需要先配置好自整定的参数，包括自整定参数表首地址寄存器、自整定参数表。参数的设置必须依据于被控过程的具体特性。下面简要描述一下关键参数的配置。

- 注意：继电输出变化量、滞后值、偏差值指的是 PID 内部运算的值。比如，继电输出变化量设置为 0.1，则表示输出每次的变化量是 10%；滞后值设置为 0.02，假定被控变量是温度，温度范围是 0～100℃，那么 AI 的采样值范围就是 0～1000，在 PID 运算时被线性转换为 0～1，所以滞后值 0.02 就相当于 2℃。

- 动态响应类型：取决于用户期望被控过程的响应速度。若期望被控过程响应比较快，并且允许有一定的超调，则可以考虑选择快速、中速响应类型；若不期望被控过程反应很灵敏，并且不容许有超调，则可以考虑选择慢速、极慢速响应类型。
- 偏差值：取决于被控过程能够容许的振荡范围。用户可以选择使用指定的偏差值（KINCO_DEV），也可以选择自动计算偏差值。若选择自动，偏差值就输出在 KINCO_AT_DEV 中，它的值是所用滞后值的 4.5 倍。
- 滞后值：取决于过程变量中干扰的影响程度。用户可以选择使用指定的滞后值（KINCO_HYS），也可以选择自动计算滞后值。若选择自动，滞后值就输出在 KINCO_AT_HYS 中，另外，若被控过程的时间常数很大，那么自动计算滞后值将耗时很长。

② 设定一组 PID 参数，然后将要整定的 PID 转为自动模式运行，让过程变量值达到 SP 值并围绕 SP 值形成比较有规律的振荡。

用户观察到 PV 值形成振荡后，就可以将相应的自整定控制位（SM249.x）赋值为 1 来启动自整定。在自整定过程中，控制位必须保持为 1，若用户将控制位赋值为 0 则会终止自整定。可参照下图：



③ 继电自整定将使系统产生极限环周期振荡。Kinco-K5 的自整定至少需要大约 6 个振荡周期的时间才能完成，因此若被控过程的时间常数较大的话，那么自整定也将耗时较长。

④ 在自整定过程中，用户程序可以读取状态字节（KINCO_AT_STAT）来了解当前自整定的状态以及是否有报警发生。并且可以读取结果字节（KINCO_AT_RES）来了解自整定的结果，KINCO_AT_RES 的最高位指明了自整定是否完成，其余 7 位的组合值指明了自整定终止的原因。

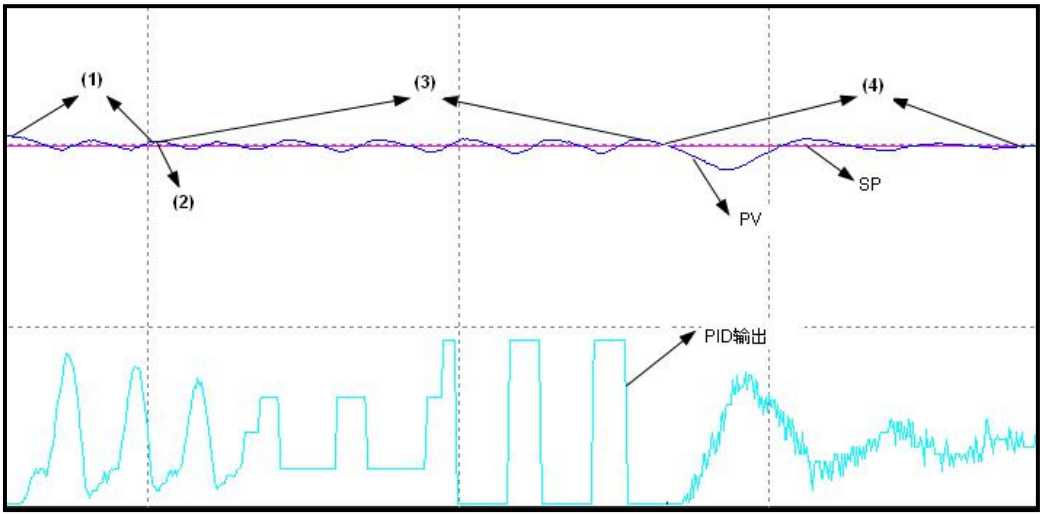
⑤ 自整定正常完成后，得到的新 PID 参数会直接赋给相应 PID 功能块的 KP、TR、TD 输入参数，并且也会输出到自整定参数表中的 KINCO_AT_Kp、KINCO_AT_Ti 和 KINCO_AT_Td 中。

一次自整定完成后，用户只需将相应的控制位（SM249.x）置 0 且至少保持一个 PID 采样周期后，然后再次将控制位置 1 将开始一个新的自整定；

由于干扰等因素的影响，自整定得到的 PID 参数未必是最优的，用户可以根据具体情况适当地调整这些参数。另外，若新参数还是能够使 PV 值围绕 SP 值形成振荡，那么就可以在这些参数的基础上再运行一次自整定。

6.15.2.4 示例

这里采用了与 6.15.1 PID 示例中相同的模拟系统，关于此系统的描述和 PLC 配置请参阅前文。下图描述了本次自整定的过程和最终控制效果：



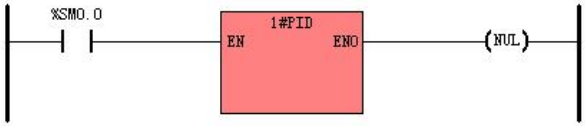
（1）首先设定了一组参数，并让 PID 自动运行，使 PV 值围绕 SP 值形成了振荡。
（2）在此处启动了 PID 自整定。
（3）PID 自整定过程。
（4）自整定完成后，使用新参数的 PID 在自动方式下运行。在这里我们给系统增加了扰动，观察 PID 的控制效果。

具体程序如下：

LD

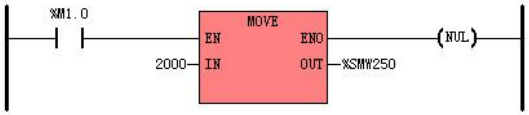
主程序 MAIN：

```
(* Network 0 *)
```

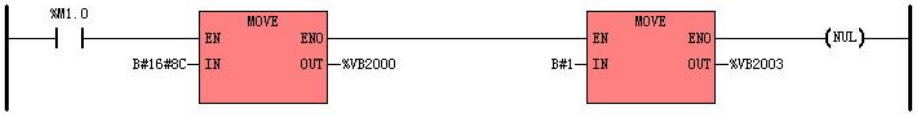


子程序（SBR02）1#PID：

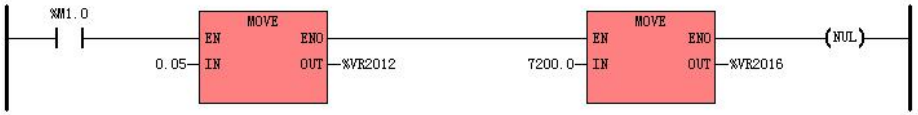
```
(* Network 0 *)
(* ****下面设置第1路PID的自整定*****
先设定自整定参数表的起始地址，这里设置为2000，代表VB2000。
M1.0---HMI上的启动/停止自整定按钮。 *)
```




```
(* Network 1 *)
(* KDN_AT_CFG：慢速响应，PID，自动计算滞后值和偏差值。
   自动计算滞后和偏差值耗时可能较长。若能根据实际情况来手动设置合理的值，自整定结果也会较好。
   PV值初始滤波时间：设为1秒。一般情况下这个参数必需配置以滤除扰动的影响。 *)
```



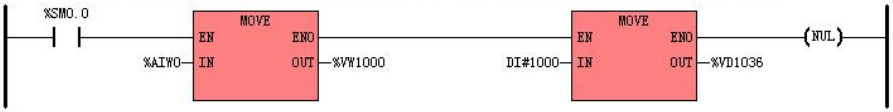

```
(* Network 2 *)
(* 继电输出变化量KDN_STEP：根据实际特性设定，原则是足以引起被控对象值的明显变化
   看门狗时间：设定为7200秒。 *)
```




```
(* Network 3 *)
(* 控制PID自整定的启动、停止。 *)
```

LD

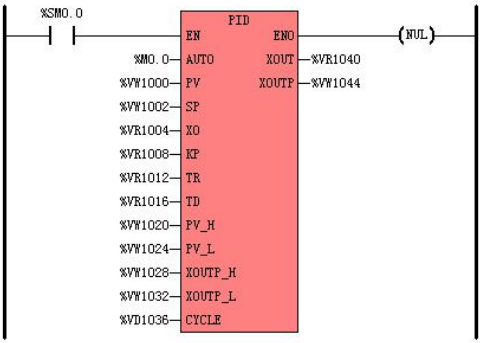
(* Network 4 *)
(* *****下面是调用PID的程序*******
设定PV值、采样周期。
采样周期是一个能够影响控制效果的重要参数，需要根据被控对象的实际特性来设定 *)



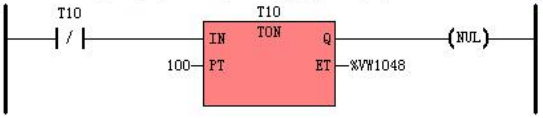
(* Network 5 *)
(* PV的最大、最小值 *)

(* Network 6 *)
(* XOUT_P的最大、最小值 *)

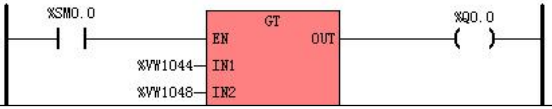
(* Network 7 *)
(* M0.0---HMI上的启动/停止PID自动方式按钮。
注意：在自整定之前，需要通过HMI设置一组P、I、D参数让被控过程形成振荡。 *)



(* Network 8 *)
(* 下面是形成周期为1秒的方波输出，其占空比由PID的输出控制。 *)



(* Network 9 *)



IL

主程序 MAIN：

```
(* Network 0 *)
LD      %SM0.0
CAL     1#PID
```

IL

子程序（SBR02）1#PID：

```
(* Network 0 *)
(*****下面设置第 1 路 PID 的自整定*****)
(*先设定自整定参数表的起始地址，这里设置为 2000，代表 VB2000。*)
(*M1.0---HMI 上的启动/停止自整定按钮。*)
LD      %M1.0
MOVE    2000, %SMW250
(* Network 1 *)
(*KINCO_AT_CFG：慢速响应，PID，自动计算滞后值和偏差值。*)
(*自动计算滞后和偏差值耗时可能较长。*)
(*若能根据实际情况来手动设置合理的值，自整定结果也会较好。*)
(*PV 值初始滤波时间：设为 1 秒。一般情况下这个参数必需配置以滤除扰动的影响。*)
LD      %M1.0
MOVE    B#16#8C, %VB2000
MOVE    B#1, %VB2003
(* Network 2 *)
(*继电输出变化量：根据实际特性设定，原则是足以引起被控对象值的明显变化。*)
(*看门狗时间：设定为 7200 秒。*)
LD      %M1.0
MOVE    0.05, %VR2012
MOVE    7200.0, %VR2016
(* Network 3 *)
(*控制 PID 自整定的启动、停止。*)
LD      %M1.0
ST      %SM249.0
```

IL

```
 (* Network 4 *)
(******下面是调用 PID 的程序********)
(*设定 PV 值、采样周期。*)
(*采样周期是一个能够影响控制效果的重要参数，需要根据被控对象的实际特性来设定*)
LD      %SM0.0
MOVE    %AIW0, %VW1000
MOVE    DI#1000, %VD1036
(* Network 5 *)
(*PV 的最大、最小值*)
LD      %SM0.0
MOVE    1000, %VW1020
MOVE    0, %VW1024
(* Network 6 *)
(*XOUT_P 的最大、最小值*)
LD      %SM0.0
MOVE    100, %VW1028
MOVE    0, %VW1032
(* Network 7 *)
(*M0.0---HMI 上的启动/停止 PID 自动方式按钮。*)
(*注意：在自整定之前，需要通过 HMI 设置一组 P、I、D 参数让被控过程形成振荡。*)
LD      %SM0.0
PID     %M0.0, %VW1000, %VW1002, %VR1004, %VR1008, %VR1012, %VR1016, %VW1020, %VW102
        4, %VW1028, %VW1032, %VD1036, %VR1040, %VW1044
(* Network 8 *)
(*下面是形成周期为 1 秒的方波输出，其占空比由 PID 的输出控制。*)
LDN     T10
TON     T10, 100
(* Network 9 *)
LD      %SM0.0
GT      %VW1044, T10
ST       %Q0.0
```

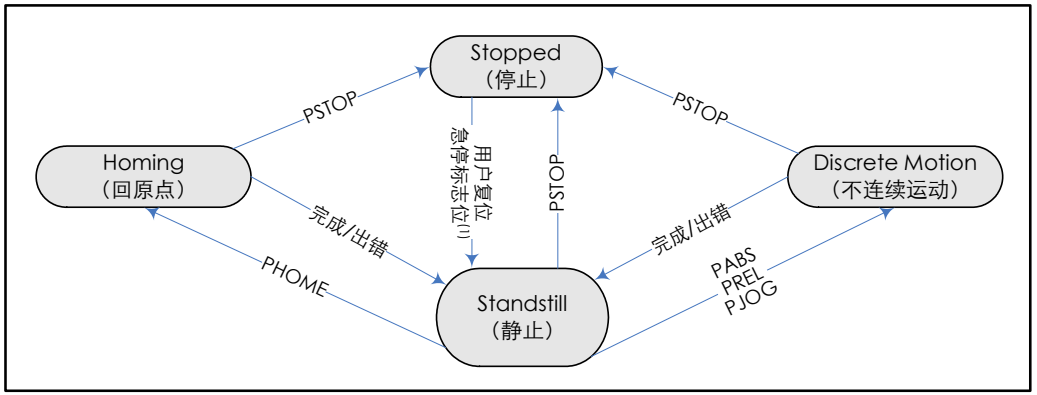
6.16 定位控制

Kinco-K5 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。Kinco-K2 提供了 3 路高速脉冲输出，输出通道分别使用使用 Q0.0、Q0.1 和 Q0.4，可以进行 3 个轴的定位控制。

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（K2 系列的范围是 20HZ-50KHZ)，用户指定的输出频率必须在这个范围之内。

6.16.1 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，PLC 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。

6.16.2 定位控制指令的相关变量

需要说明的是，K5 系列 PLC 不支持 Q0.4 输出通道，所以下面 Q0.4 相对应寄存器也对 K5 系列无效

6.16.2.1 电机方向控制信号

针对定位控制指令，PLC 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，用来禁止或者允许使用相应的方向输出通道。如下表。

	Q0.0	Q0.1	Q0.4
方向输出通道	Q0.2	Q0.3	Q0.5

方向使能控制位	SM201.3	SM231.3	SM251.3

方向输出通道用于输出电机的方向控制信号，正转时输出为 0，反转时输出为 1。

方向使能控制位用来禁止或者使能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。

6.16.2.2 控制寄存器和状态寄存器

针对定位控制指令，Kinco-K 系列在 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	Q0.4	描 述
SMD212	SMD242	SMD262	只读。当前值（正转时增加，反转时减少），表示当前已经输出的脉冲个数。
SMD208	SMD238	SDM258	读写。新当前值。与相应标志位配合，用于修改当前值。
Q0.0	Q0.1	Q0.4	描 述
SM201.7	SM231.7	SM251.7	读写。急停标志位。 若该位为 1，则表示处于急停状态，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。
SM201.6	SM231.6	SM251.6	读写。用于决定是否复位当前值 1 – 将当前值清零。 0 – 当前值保持不变。
SM201.5	SM231.5	SM251.5	保留
SM201.4	SM231.4	SM251.4	读写。用于决定是否修改当前值 1 – 将当前值清零。 0 – 当前值保持不变。
SM201.3	SM231.3	SM251.3	方向使能控制位。 1– 禁止方向输出，方向通道作为普通 DO。 0 – 使能方向输出。
SM201.0~ SM201.2	SM231.0~ SM231.2	SM251.0~ SM251.2	保留

6.16.2.3 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	加/减速时间太短或初始速度太低，导致初始脉冲周期超过了每段的时间。
2	初始速度 MINF 超过最高允许速度（200KHz）
3	初始速度 MINF 低于最低允许速度（125Hz）
4	加速和减速过程所需的脉冲个数超过了总脉冲个数
5	初始速度 MINF 超过了最高速度 MAXF

6.16.2.3 如何修改定位控制中的当前值

3 路高速输出通道各有一个当前值寄存器，分别为 SMD212、SMD242 和 SMD262，其中存放着相应通道已经输出的脉冲个数。 当前值寄存器是只读值，不允许在程序中直接进行修改。若需要修改当前值，则可以采取如下方法：

- **方法一**

利用复位控制位来将当前值清除为 0。

3 个通道的复位控制位分别为 SM201.6、SM231.6 和 SM251.6。

只要复位控制位为 1， PLC 就会将相应的当前值寄存器清 0。因此复位控制位仅需要保持一个扫描周期即可发挥作用，使用时注意避免复位控制位长时间保持为 1，也尽量避免在运动过程中（包括 PHOME、PREL、PABS、PJOG、PFLO_F 指令正在执行）来复位当前值，以免计数出现误差。

下面以通道 0 为例来说明如何复位当前值：

```
(* Network 0 *)
(*以原点信号为基准，当运动到原点时，要求将当前值清 0.*)
LD      %SM0.0
PHOME   0, %M0.0, %M0.1, %M0.2, %VW0, %VW2, %VW4, %VD6, %VW10, %M0.4, %M0.5, %MB1
(* Network 1 *)
(*PHOME 指令完成后，利用 DONE 标志位将当前值清 0。*)
LD      %M0.4
R_TRIG
ST      %SM201.6
```

- **方法二**

利用下述寄存器，可以将当前值设置为任意值。

Q0.0	Q0.1	Q0.4	描 述
SMD208	SMD238	SDM258	读写。新当前值。与相应标志位配合，用于修改当前值。
SM201.4	SM231.4	SM251.4	读写。用于决定是否修改当前值 1 – 将当前值清零。 0 – 当前值保持不变。

以通道 0 为例来说明使用方法：若 SM201.4 为 0，则保持当前值 SMD212 不变。若 SM201.4 为 1，则将 SMD208 中的值赋值给当前值 SMD212。注意尽量避免在运动过程中（包括 PHOME、PREL、

PABS、PJOG、PFLO_F 指令正在执行）来修改当前值寄存器，以免当前值计数出现误差。

下面的示例程序是以通道 0 为例，说明如何修改当前值：

```
(* Network 0 *)
(*以原点信号为基准，当运动到原点时，要求将当前值设置为 100.*)
LD      %SM0.0
PHOME   0, %M0.0, %M0.1, %M0.2, %VW0, %VW2, %VW4, %VD6, %VW10, %M0.4, %M0.5, %MB1
(* Network 1 *)
(*PHOME 指令完成后，利用 DONE 标志位来修改当前值。*)
LD      %M0.4
R_TRIG
MOVE    DI#100, %SMD208
ST      %SM201.4
```

2.3.3.2 如何修改定位控制指令中的最高输出频率？

PREL（相对运动）和 PABS（绝对运动）在脉冲输出过程中不会去改变最高输出频率。它们在启动时会读取当前最低频率、最高频率和加减速时间参数值，并根据这些值自动计算适当的加减速段数，然后启动脉冲输出。在脉冲输出过程中，PREL 和 PABS 不会再读取上述参数值，因此这些参数值的变化对于脉冲输出没有影响。

PJOG（点动）指令在执行过程中会实时读取输入频率参数（*MAXF*）值，并根据新的频率值来调整输出脉冲的频率。

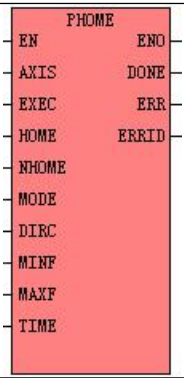
PHOME（回原点）指令在运行到最高频率段之后、在回原点和原点信号产生之前，会实时读取最高频率参数（*MAXF*）值，并根据新的频率值自动计算加速或者减速段数，然后加速或者减速运行到新的频率值再维持匀速输出。

PFLO_F（可变频率的脉冲串输出）提供随时可变频率的输出。具体见指令介绍。

6.16.3 PHOME（回原点）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于

LD	PHOME			☑ K5 ☑ K2
IL	PHOME	PHOME *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	WORD	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

MODE, DIRC, MINF, MAXF, TIME, 必须同时为常量类型或同时为内存类型

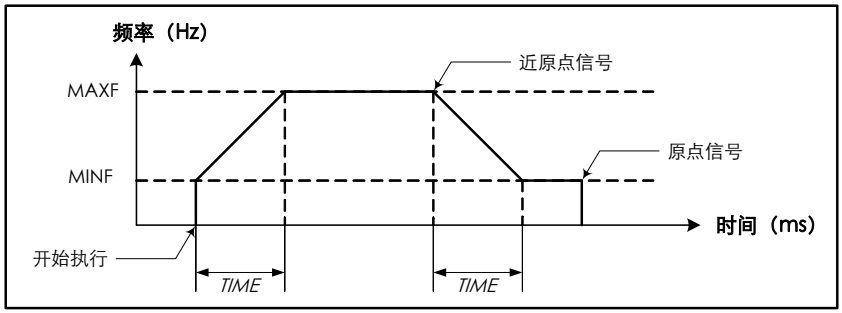
下表对各个参数的作用进行了详细的描述。

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 Q0.1，2 表示使用 Q0.4。
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 表示正转；1 表示反转。 关于方向控制信号的输出请参阅 6.16.2.1 电机方向控制信号 中的描述。

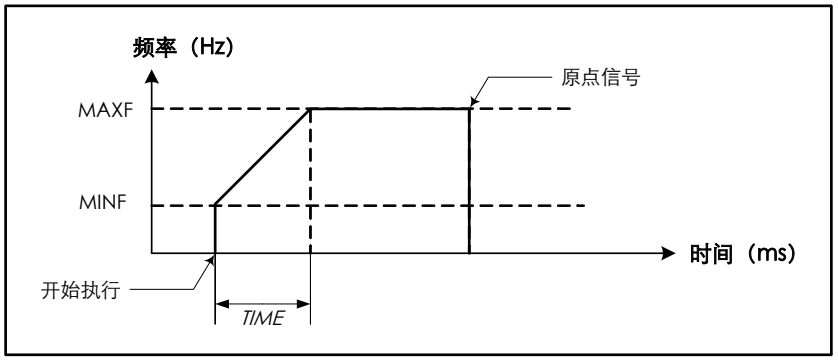
MINF	输出脉冲的初始速度（即初始频率），单位：Hz。 *MINF* 不允许低于 125Hz，也不允许超过 *MAXF*。
MAXF	输出脉冲的最高速度（即最高频率），单位：Hz。 *MAXF* 的允许范围是 125Hz~200KHz。*MAXF* 必须大于等于 *MINF*。
TIME	加/减速时间，单位：ms。本指令采用相同的加速时间和减速时间。 加速时间是由 *MINF* 加速到 *MAXF* 所需的时间，减速时间是由 *MAXF* 减速到 *MINF* 所需的时间。
DONE	完成标志位。当指令正常执行完成时，*DONE* 由 0 跳变到 1。
ERR	出错标志位。若指令执行时发生错误，则该标志位被置 1。
ERRID	错误码。

PHOME 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：



PHOME 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

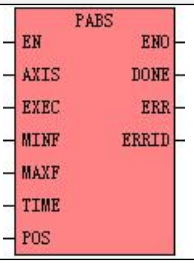
需要根据实际要求自行改变当前值。

- 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。

6.16.4 PABS（绝对运动）

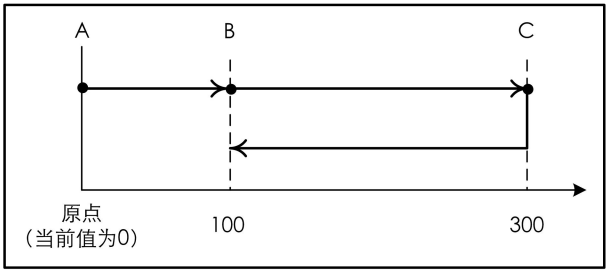
➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PABS			☑ K5 ☑ K2
IL	PABS	PABS AXIS, EXEC, MINF, MAXF, TIME, POS, DONE, ERR, ERRID	U	

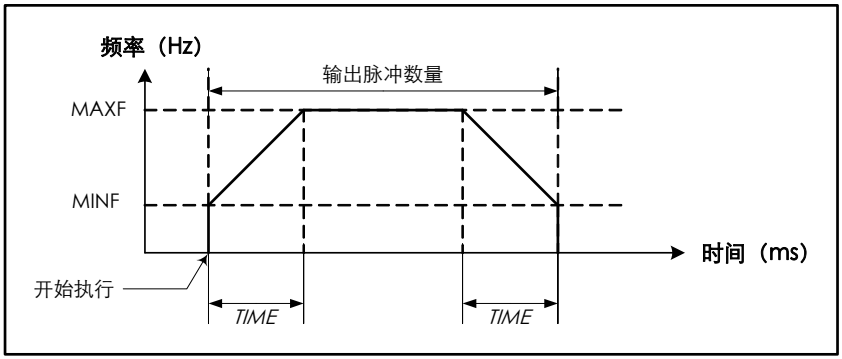
参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0 或者 1）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	WORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
POS	输入	DINT	I、Q、M、V、L、SM、HC、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

MINF,MAXF,TIME,POS, 必须同时为常量类型或同时为内存类型

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 Q0.1，2 表示使用 Q0.4。
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。 *MINF* 不允许低于 125Hz，也不允许超过 *MAXF*。
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 125Hz~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。
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。

PABS 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。PABS 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 PABS 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。

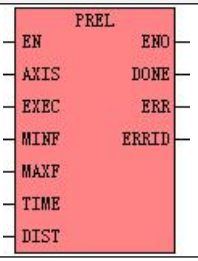
- IL

  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。

6.16.5 PREL（相对运动）

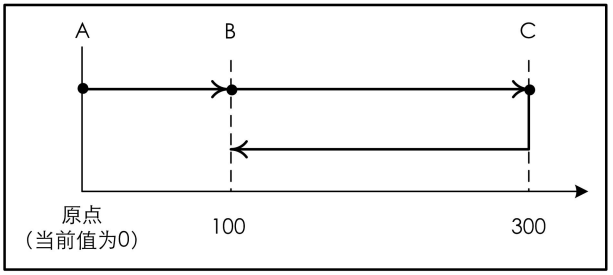
➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PREL			☑ K5 ☑ K2
IL	PREL	PREL AXIS, EXEC, MINF, MAXF, TIME, DIST, DONE, ERR, ERRID	U	

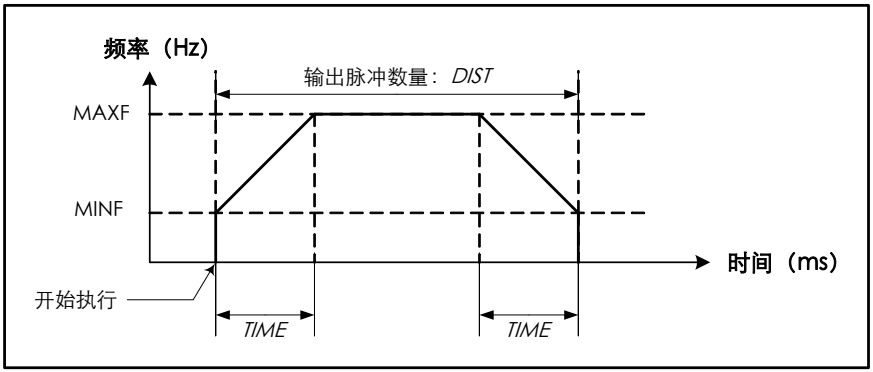
参数	输入/输出	数据类型	允许的内存区
AXIS	输入	INT	常量（0 或者 1）
EXEC	输入	BOOL	I、Q、V、M、L、SM、RS、SR
MINF	输入	WORD	I、Q、M、V、L、SM、常量
MAXF	输入	DWORD	I、Q、M、V、L、SM、常量
TIME	输入	WORD	I、Q、M、V、L、SM、常量
DIST	输入	DINT	I、Q、M、V、L、SM、HC、常量
DONE	输出	BOOL	Q、M、V、L、SM
ERR	输出	BOOL	Q、M、V、L、SM
ERRID	输出	BYTE	Q、M、V、L、SM

MINF, MAXF, TIME, DIST 必须同时为常量类型或同时为内存类型

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 Q0.1，2 表示使用 Q0.4。
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。 *MINF* 不允许低于 125Hz，也不允许超过 *MAXF*。
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 125Hz~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。
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。

PREL 指令采用相对式定位，输出脉冲的数量就是移动量 *DIST*。PREL 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 PREL 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。

6.16.6 PJOG（点动）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PJOG	PJOG EN ENO AXIS DONE EXEC ERR MAXF ERRID DIRC		☑ K5 ☑ K2
IL	PJOG	PJOG AXIS, EXEC, MINF, DIRC, 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
MAXF	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

MAXF，DIRC 必须同时为常量类型或同时为内存类型

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 Q0.1，2 表示 Q0.4
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 125Hz~200KHz。
DIRC	电机的运转方向：0 表示正转；1 表示反转。 关于方向控制信号的输出请参阅 6.16.2.1 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 PJOG 指令从指定的通道 *AXIS* 持续输出脉冲串，频率为 *MAXF*，若 *MAXF* 是变量，那么在脉冲输出过程中可以随时改变这个输出频率值。若 *EXEC* 输入为 0，则立即停止输出。

PJOG 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：

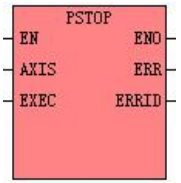
- IL

  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。

6.16.7 PSTOP（急停）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PSTOP			☑ K5 ☑ K2
IL	PSTOP	PSTOP 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。

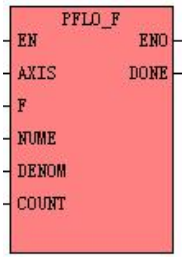
PSTOP 指令用于立即停止 *AXIS* 通道的脉冲输出，从而使当前的运动紧急停止，同时将急停标志位（SM201.7/SM231.7）置位。用户需要使用程序将急停标志位清 0，否则 CPU 不会再执行任何定位控制指令。

- 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。

6.16.8 PFLO_F（可变频率的脉冲串输出）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PFLO_F			☑ K5 ☑ K2

IL	PFLO_F	PFLO_F AXIS, F, NUME, DENOM, COUNT, DONE	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）
F	输入	DINT	L、M、V、常量
NUME	输入	INT	L、M、V、常量
DENOM	输入	INT	L、M、V、常量
DONE	输出	BOOL	Q、M、V、L
COUNT	输入	DWORD	L、M、V、常量

F, NUME, DENOM, COUNT 必须同时为常量类型或同时为内存类型

下表对各个参数的作用进行了详细的描述。

参数	描　述
EN	使能端。若 EN 为 1，则执行跟随脉冲输出，否则停止脉冲输出。
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.2。
F	输入频率。单位：Hz
NUME	输出脉冲频率的电子齿轮分子
DENOM	输出脉冲频率的电子齿轮分母
DONE	完成标志位。若有脉冲输出，则为 0；若停止脉冲输出则为 1。

若 *EN* 为 1，则执行指令，从指定的通道 *AXIS* 持续输出脉冲，输出频率等于输入频率 *F* 乘以电子齿轮（*NUME*/ *DENOM*），总共输出 *COUNT* 个脉冲。当输出脉冲个数达到 *COUNT* 个则完成输出，将标志位 *DONE* 置 1。若输入频率 *F* 小于 0，则表示反转，对应的方向通道输出为 1；若 F 大于 0，则表示正转，对应的方向通道输出为 0。

注意：若计算得到输出频率小于最低允许频率 30Hz，PLC 则停止输出,当输出频率再次超过最低频率后才会继续跟随输出。

若 EN 输入为 0，则立即停止输出。

- LD

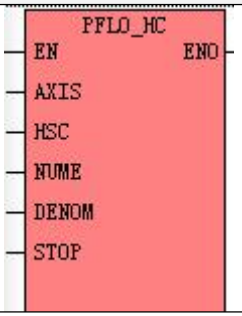
  当 *EN* 为 1 时，，则该指令被执行：

- IL

  当 CR 值为 1 时，则该指令被执行。该指令的执行不影响 CR 值。

6.16.9 PFLO_HC（跟随高速计数的脉冲串输出）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	PFLO_HC			☑ K5 ☑ K2
IL	PFLO_HC	PFLO_HC AXIS, HSC, NUME, DENOM, COUNT, DONE	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）
HSC	输入	BYTE	常量
NUME	输入	INT	L、M、V、常量
DENOM	输入	INT	L、M、V、常量
DONE	输出	BOOL	Q、M、V、L
COUNT	输入	DWORD	L、M、V、常量

F，NUME，DENOM，COUNT 必须同时为常量类型或同时为内存类型

下表对各个参数的作用进行了详细的描述。

参数	描　述
EN	使能端。若 EN 为 1，则执行跟随脉冲输出，否则停止脉冲输出。
AXIS	所用的高速输出通道。0 表示使用 Q0.0，1 表示使用 Q0.1，2 表示使用 Q0.2。
HSC	高速计数器通道。0,1,2,3 分别表示 HSC0，HSC1，HSC2，HSC3
NUME	输出脉冲频率的电子齿轮分子
DENOM	输出脉冲频率的电子齿轮分母
DONE	完成标志位。若有脉冲输出，则为 0；若停止脉冲输出则为 1。

若 *EN* 为 1，则执行指令，从指定的通道 *AXIS* 持续输出脉冲，输出频率等于 HSC 指定的通道的频率乘以电子齿轮（*NUME*/ *DENOM*），总共输出 *COUNT* 个脉冲。当输出脉冲个数达到 *COUNT* 个则

完成输出，将标志位 *DONE* 置 1。若 HSC 输入的频率小于 0，则表示反转，对应的方向通道输出为 1；若大于 0，则表示正转，对应的方向通道输出为 0。

注意：若计算得到输出频率小于最低允许频率 30Hz，PLC 则停止输出,当输出频率再次超过最低频率后才会继续跟随输出。

若 EN 输入为 0，则立即停止输出。

- LD

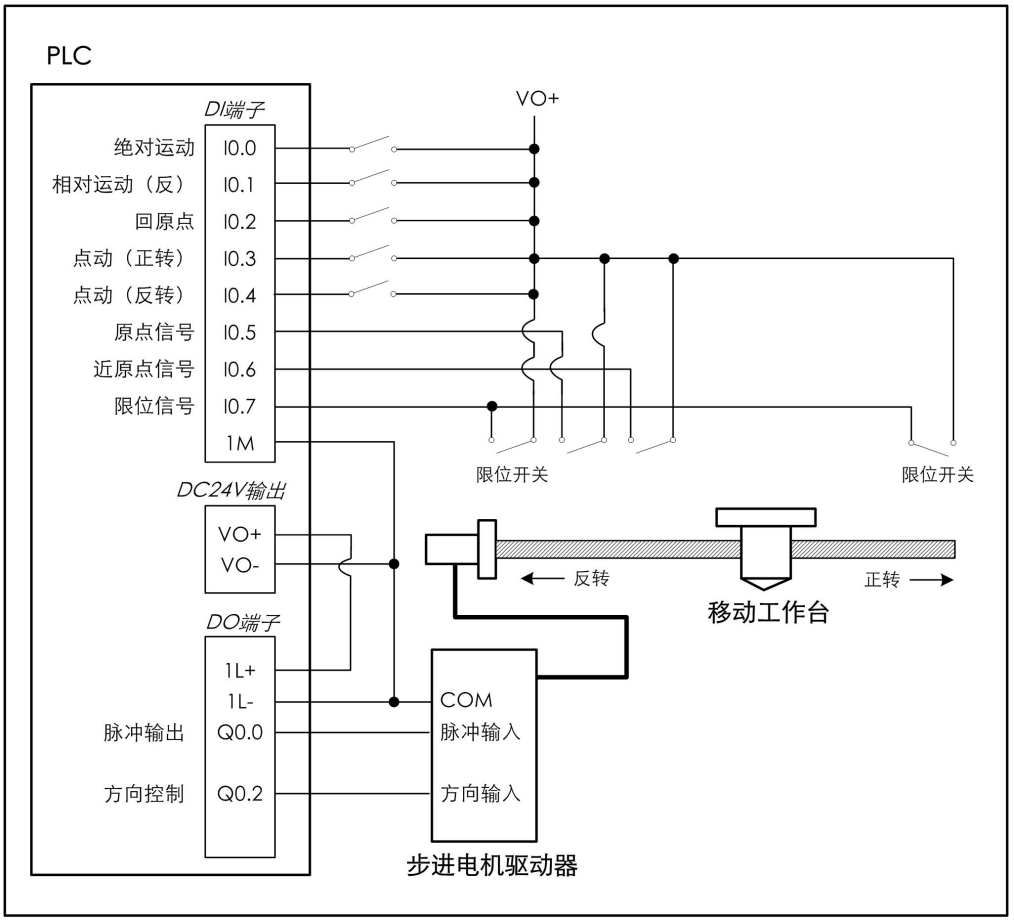
  当 *EN* 为 1 时，，则该指令被执行：

- IL

  当 CR 值为 1 时，则该指令被执行。该指令的执行不影响 CR 值。

6.16.10 定位控制指令示例

➢ 接线示意图



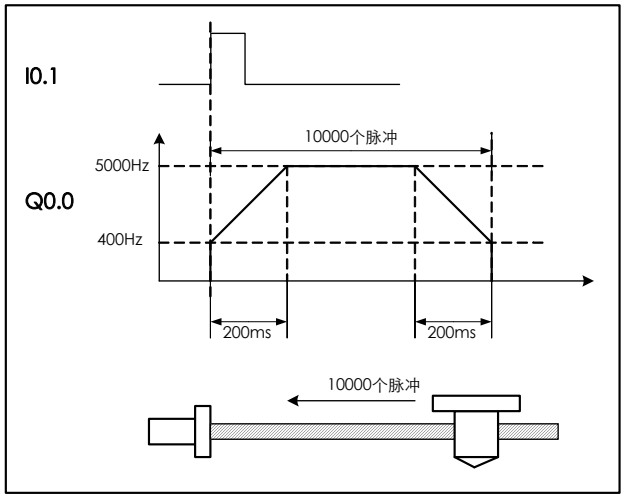
下面我们将依据这个系统来举例描述 PREL、PABS、PHOME、PJOG 和 PSTOP 指令的使用方式。

- 相对运动（反转）

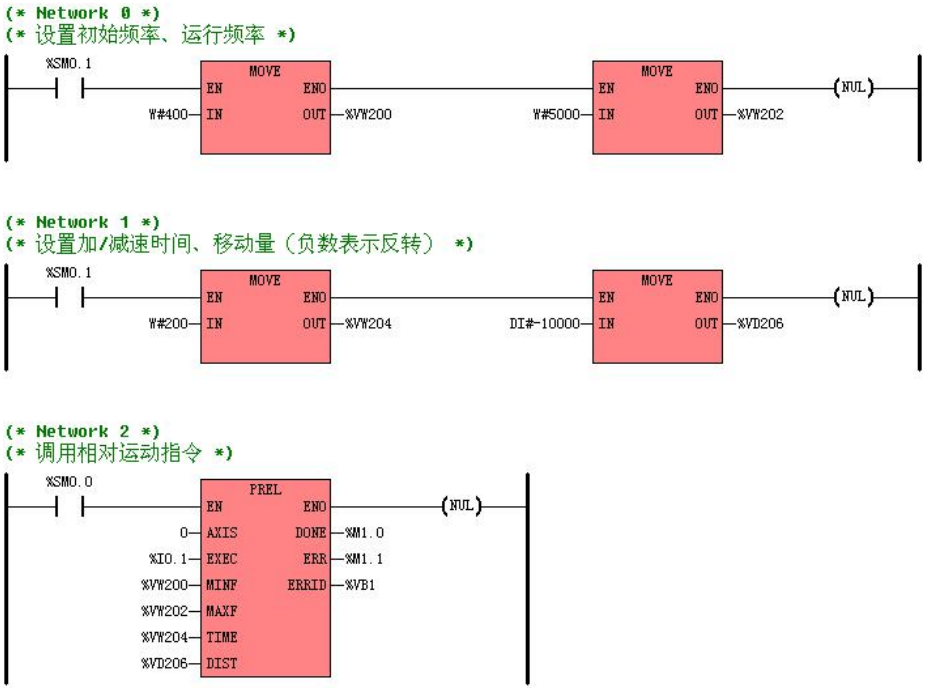
  I0.1 所接的“相对运动（反）”信号用于启动相对运动（反转）。

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。



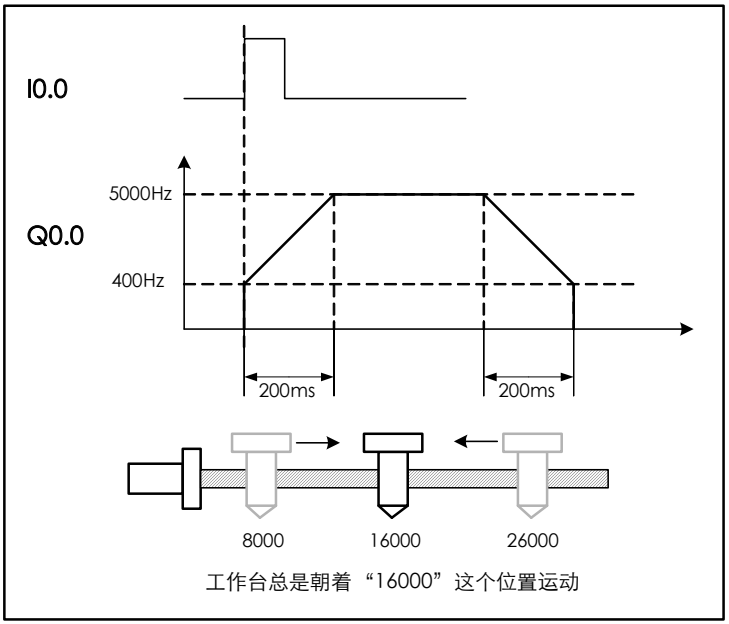
LD



IL	(* Network 0 *)
	(*设置初始频率、运行频率*)
	LD %SM0.1
	MOVE W#400, %VW200
	MOVE W#5000, %VW202
	(* Network 1 *)
	(*设置加/减速时间、移动量（负数表示反转）*)
	LD %SM0.1
	MOVE W#200, %VW204
	MOVE DI#-10000, %VD206
	(* Network 2 *)
	(*调用相对运动指令*)
	LD %SM0.0
	PREL 0, %I0.1, %VW200, %VW202, %VW204, %VD206, %M1.0, %M1.1, %VB1

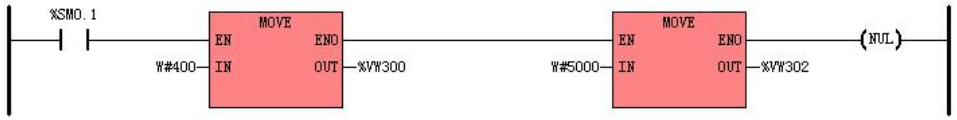
➢ 绝对运动

I0.0 所接的“绝对运动”信号用于启动绝对运动。

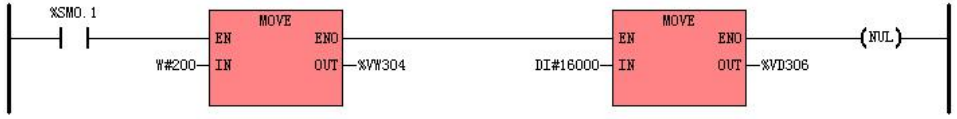
描述	若检测到 I0.0 的上升沿，则在 Q0.0 输出脉冲。据 Q0.2 输出为 0 表示正转，输出为 1 表示反转。  工作台总是朝着“16000”这个位置运动

LD

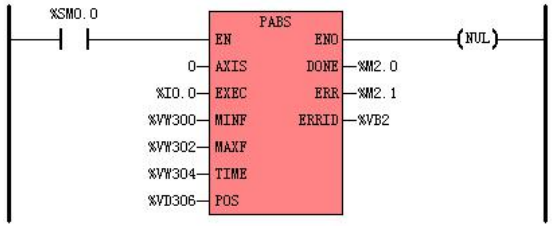
(* Network 0 *)
(* 设置初始频率、运行频率 *)



(* Network 1 *)
(* 设置加/减速时间、目标位置 *)



(* Network 2 *)
(* 调用绝对运动指令 *)



IL

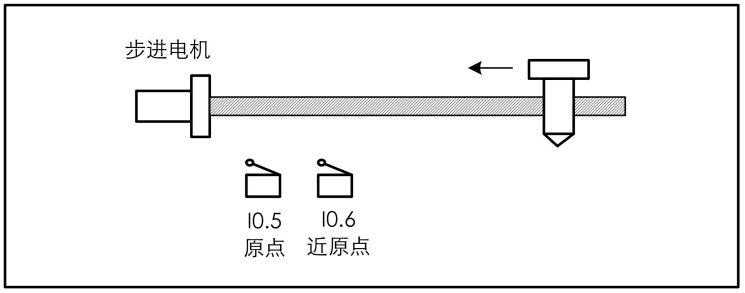
```
(* Network 0 *)
(*设置初始频率、运行频率*)
LD          %SM0.1
MOVE     W#400, %VW300
MOVE     W#5000, %VW302
(* Network 1 *)
(*设置加/减速时间、目标位置*)
LD          %SM0.1
MOVE     W#200, %VW304
MOVE     DI#16000, %VD306
(* Network 2 *)
(*调用绝对运动指令*)
LD        %SM0.0
PABS      0, %I0.0, %VW300, %VW302, %VW304, %VD306, %M2.0, %M2.1, %VB2
```

➢ 回原点

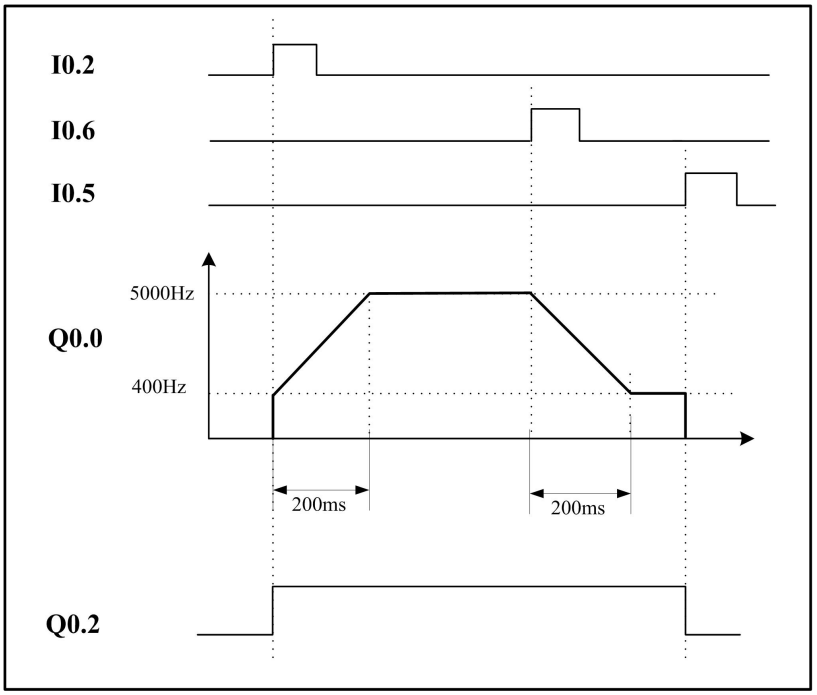
I0.2 所接的“回原点”信号用于启动回原点运动。需要注意的是：当回原点运动完成后，当前值寄存器（SMD212/SM242）并不自动清零，用户需要根据实际要求自行改变当前值。

描述

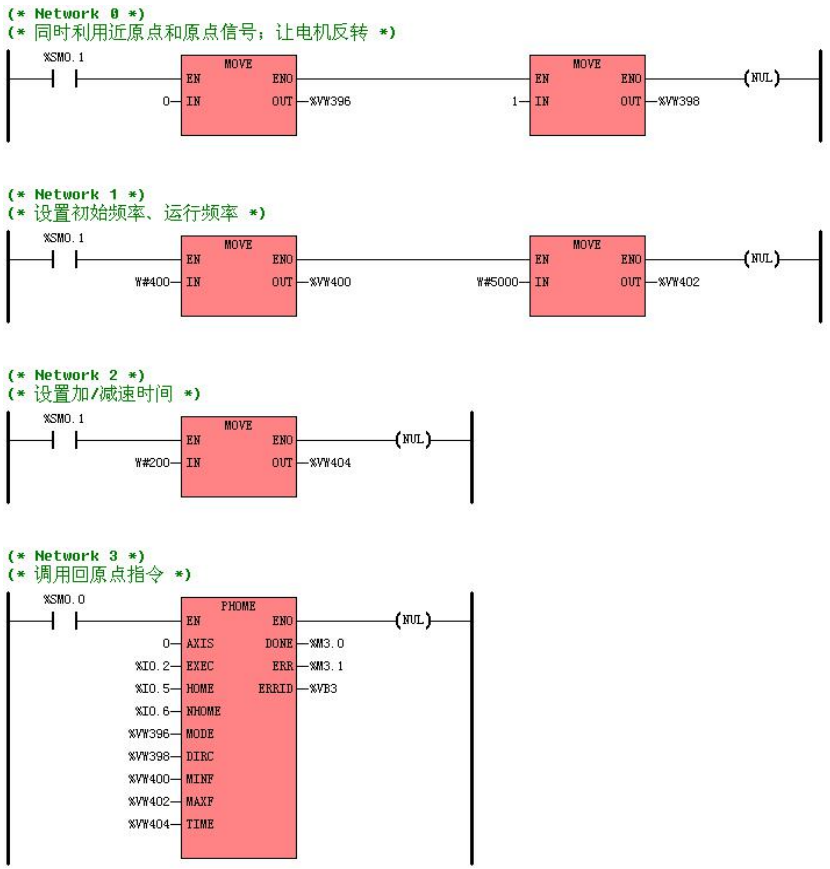
假定当前工作台处于如下状态：



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。



LD



IL

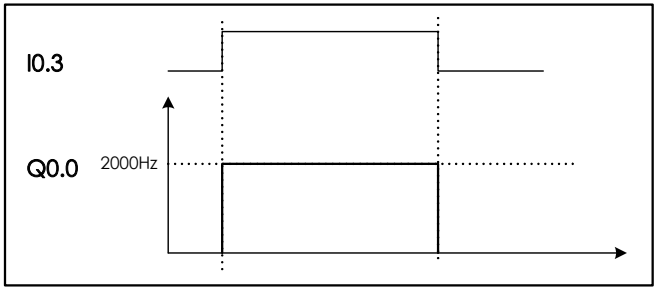
```
(* Network 0 *)
LD          %SM0.1
MOVE     0, %VW396          (* 同时利用近原点和原点信号 *)
MOVE     1, %VW398          (* 让电机反转 *)
MOVE     W#400, %VW400      (* 设置初始频率 *)
MOVE     W#5000, %VW402     (* 设置运行频率 *)
MOVE     W#200, %VW404      (* 设置加/减速时间 *)
(* Network 2 *)
LD      %SM0.0
PHOME
0, %I0.2, %I0.5, %I0.6, %VW396, %VW398, %VW400, %VW402, %VW404, %M3.0, %M3.1, %VB3
```

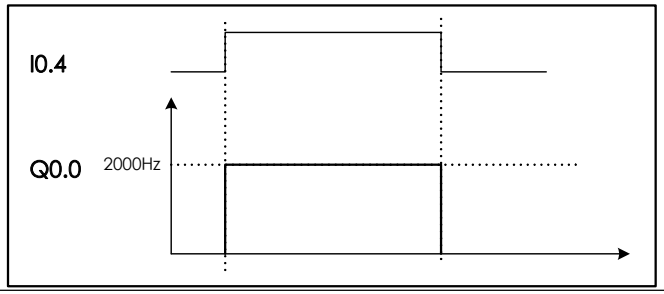
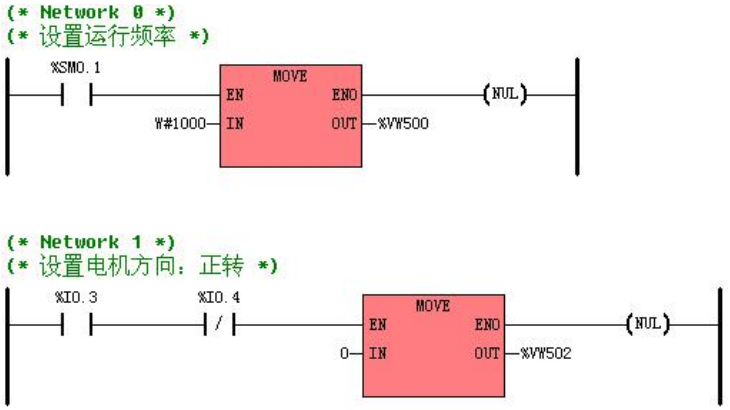
➢ 点动

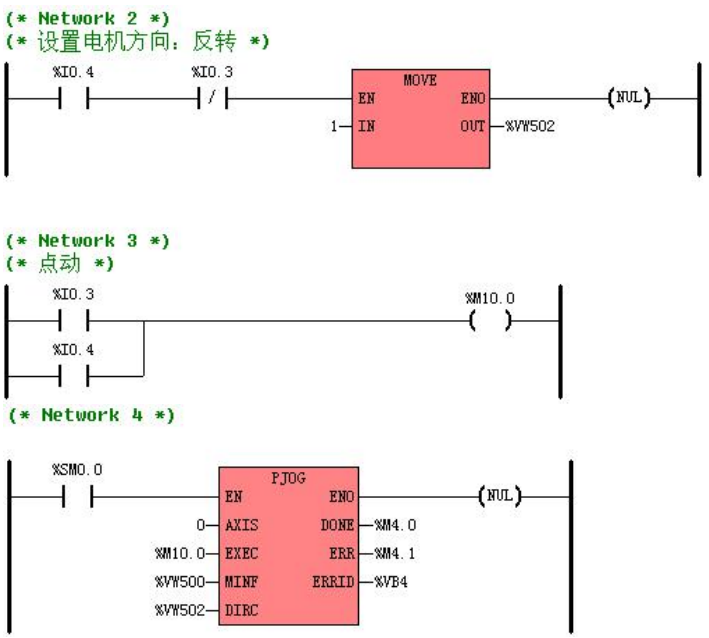
I0.3 所接的“点动（正转）”信号用于点动（电机正转）。

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。

<table>
<tr><td>描述</td><td>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 I0.3 为 0，则 Q0.0 立即停止输出。Q0.2 输出为 1 表示电机反转。
 
</td></tr>
<tr><td>LD</td><td> 
</td></tr>
</table>

<table>
<tr><td></td><td>


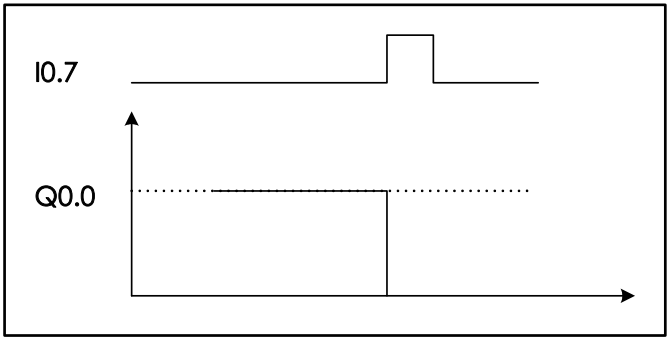
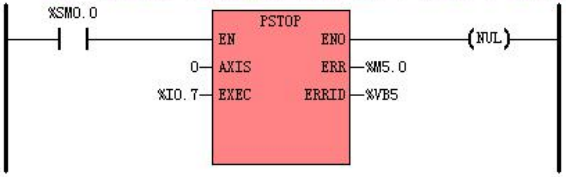
</td></tr>
<tr><td>IL</td><td>

```
(* Network 0 *)
(*设置运行频率*)
LD      %SM0.1
MOVE    W#1000, %VW500
(* Network 1 *)
(*设置电机方向：正转*)
LD      %I0.3
ANDN    %I0.4
MOVE    0, %VW502
(* Network 2 *)
(*设置电机方向：反转*)
LD      %I0.4
ANDN    %I0.3
MOVE    1, %VW502
(* Network 3 *)
(*点动*)
LD      %I0.3
OR      %I0.4
ST      %M10.0
(* Network 4 *)
LD      %SM0.0
PJOG    0, %M10.0, %VW500, %VW502, %M4.0, %M4.1, %VB4
```

</td></tr>
</table>

➢ 急停

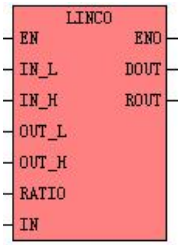
在丝杠两端有两个限位开关，这两个开关并联接入 I0.7 作为急停信号。

<table>
<tr><td>描述</td><td>若检测到 I0.7 的上升沿，则 Q0.0 立即停止输出。
 
</td></tr>
<tr><td>LD</td><td>(* 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。 *)
 
</td></tr>
<tr><td>IL</td><td>(* 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
PSTOP 0, %I0.7, %M5.0, %VB5</td></tr>
</table>

6.17 附加指令

6.17.1 LINCO（线性变换）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	LINCO			☑ K5 ☑ K2
IL	LINCO	LINCO IN_L, IN_H, OUT_L, OUT_H, RATIO, IN, DOUT, ROUT	U	

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×(k×IN + b)

DOUT ＝ TRUNC(ROUT)

其中 $k = \frac{OUT\_H - OUT\_L}{IN\_H - IN\_L}$ ； $b = OUT\_L - k \times IN\_L$ 。

- LD

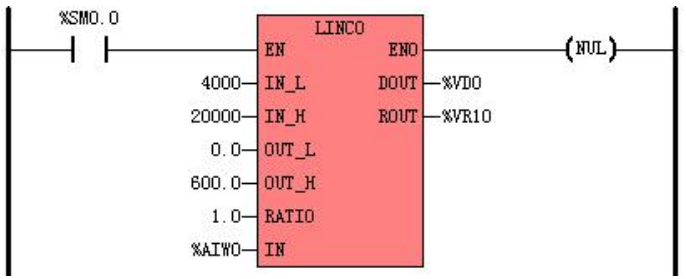
  如果 *EN* 为 1，则该指令被执行。

- 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。

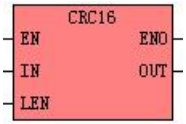
➢ 指令使用举例

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	LD %SM0.0 LINCO 4000, 20000, 0.0, 600.0, 1.0, %AIW0, %VD0, %VR10

6.17.2 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	适用于
LD	CRC16			☑ K5 ☑ K2
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。

注意所用内存区域不能超出有效的内存范围，否则执行结果不可预期。

- LD

  若 *EN* 值为 1，则该指令被执行，否则不执行。

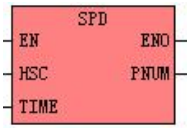
- IL

  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。

6.17.3 SPD（脉冲密度）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	SPD			☑ K5 ☑ K2
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

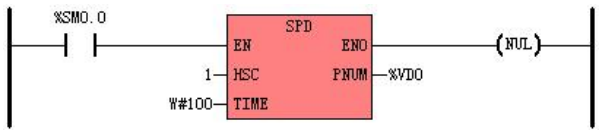
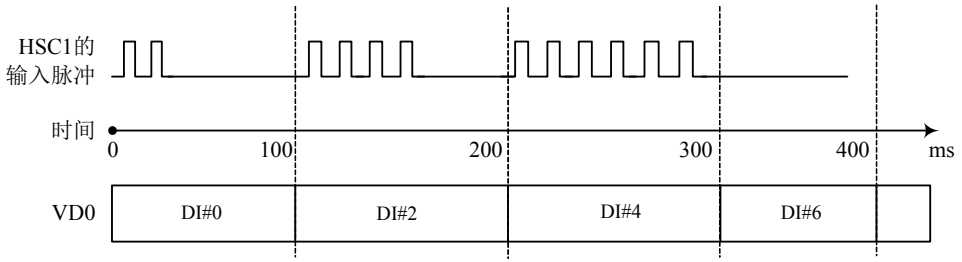
  如果 *EN* 为 1，则该指令被执行。

- IL

  如果 CR 值为 1，则该指令被执行。

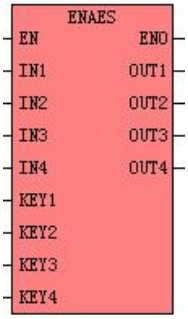
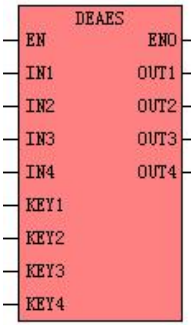
  该指令的执行不影响 CR 值。

➢ 指令使用举例

<table>
<tr><td>LD</td><td> 
</td><td>SM0.0 恒为 1，因此 SPD 指令总是执行：对 HSC1 的脉冲计数值进行计算，每到 100ms 就将这个时间内计算的脉冲个数输出至 VD0。</td></tr>
<tr><td>IL</td><td colspan="2">LD %SM0.0 (* 建立 CR，其值为 1 *)
SPD 1, W#100, %VD0 (* 计算 HSC1 每 100ms 接收到的脉冲个数并赋给 VD0 *)</td></tr>
<tr><td>结果</td><td colspan="2">结果举例如下：
 
</td></tr>
</table>

6.17.4 ENAES（AES-128 加密） DEAES（AES-128 解密）

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	ENAES			☑ K5 ☑ K2
	DEAES			
	ENAES	ENAES IN1, IN2, IN3, IN4, KEY1, KEY2, KEY3, KEY4, OUT1, OUT2, OUT3, OUT4	U	
	DEAES	DEAES IN1, IN2, IN3, IN4, KEY1, KEY2, KEY3, KEY4, OUT1, OUT2, OUT3, OUT4		

参数	输入/输出	数据类型	允许的内存区
IN1	输入	DWORD	I、Q、L、M、V、SM、常量
IN2	输入	DWORD	I、Q、L、M、V、SM、常量
IN3	输入	DWORD	I、Q、L、M、V、SM、常量

IN4	输入	DWORD	I、Q、L、M、V、SM、常量
KEY1	输入	DWORD	I、Q、L、M、V、SM、常量
KEY2	输入	DWORD	I、Q、L、M、V、SM、常量
KEY3	输入	DWORD	I、Q、L、M、V、SM、常量
KEY4	输入	DWORD	I、Q、L、M、V、SM、常量
OUT1	输出	DWORD	Q、SM、L、M、V
OUT2	输出	DWORD	Q、SM、L、M、V
OUT3	输出	DWORD	Q、SM、L、M、V
OUT4	输出	DWORD	Q、SM、L、M、V

IN1，IN2，IN3，IN4，KEY1，KEY2，KEY3，KEY4 必须同时为常量类型或同时为内存类型。

ENAES、DEAES 指令是 AES-128 的加密、解密指令。

其中，参数 *IN1*、*IN2*、*IN3*、*IN4* 是待加密/解密的数据，*KEY1*、*KEY2*、*KEY3*、*KEY4* 是用户指定的密钥，*OUT1*、*OUT2*、*OUT3*、*OUT4* 是加密/解密后的数据。

- 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。

6.17.5 特殊数据区读写指令

K 系列 PLC 在永久存储器中提供了一片 128 字节长度的特殊数据区域，并以 4 字节为单位划分为 32 个相互独立的块，用户可以对任意块进行读写。

该区域内的数据在 PLC 出厂时已经被全部清零，用户只能对每个块写入一次非 0 的数据。PLC 会将具有非 0 数据的块锁定，不允许再次写入，但可以任意读取。注意：在写入时，待写入的 4 个字节数据不可以全部为 0，否则会忽略此次写入。

在 KincoBuillder 中执行【PLC】->【清除…】菜单命令，可以将 PLC 内的所有数据清零，包括用户工程、特殊区域内的数据等。**特殊数据区无法单独清除。**

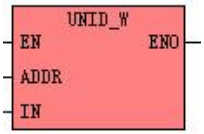
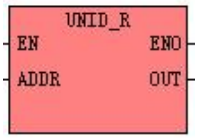
设备制造商可以利用此区域，在 PLC 中写入 S/N 之类的数据。

UNID_W 用于将参数 IN 指定的数据写入特殊数据区域内某个块中。参数 ADDR 指定了将要写入的块编号，范围是 0-31。

UNID_R 用于读取特殊数据区域内某个块的数据并存放在参数 OUT 中。参数 ADDR 指定了将要读取的块编号，范围是 0-31

⚠ **特殊数据区只能使用清除命令清除，无法单独清除。**

➢ 指令及其操作数说明

	名称	指令格式	影响 CR 值	适用于
LD	UNID_W			☑ K5 ☑ K2
	UNID_R			
	UNID_W	UNID_W ADDR, IN	U	
	UNID_R	UNID_W ADDR, OUT		

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

- 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。

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，K 系列 PLC 的 CPU 采用 Modbus RTU 协议进行通讯并作为从站。

1、PLC 内存区说明

1.1 可访问的内存区

Modbus RTU 主站可访问的内存区域分类如下：

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
错误记录（16 位无符号整数）	03,04	PLC 错误记录区

一次命令最大访问的寄存器数量如下：

1. 读取位，一次最大读取 1600 个位(200 字节)。(01，02 功能码)
2. 写入位，一次最大写入 800 个位。(15 功能码)
3. 读取字，一次最大读取 100 个字。(03, 04 功能码)
4. 写入字，一次最大写入 100 个字。(16 功能码)
5. 当操作的内存区小于上面的最大值时， 只允许一次操作整个内存区。比如报文中读取 AI 区 90 个字， 这是错误的，因为 AI 区最大 32 个字。

1.2　Modbus 寄存器编号

由于各种规格的 CPU 的内存区域不同，所以允许访问的范围也有限制。

若 Modbus RTU 主站的寄存器从 1 开始编号，那么将将下表中的寄存器号直接加 1 即可。

➢ 适用于 CPU205

内存区域	范围	类型	对应的 Modbus 寄存器号*
I	I0.0 --- I1.1	DI	0 --- 9
Q	Q0.0 --- Q1.1	DO	0 --- 9
M	M0.0 --- M1023.7	DI/DO	320 -- 8511
AI	---	AI	---
AQ	---	AO	---
V	VW0 ---VW4094	AI/AO	100 -- 2147

➢ 适用于 CPU504

内存区域	范围	类型	对应的 Modbus 寄存器号*
I	I0.0 --- I0.7	DI	0 --- 7
Q	Q0.0 --- Q0.5	DO	0 --- 5
M	M0.0 --- M1023.7	DI/DO	320 -- 8511
AI	---	AI	---
AQ	---	AO	---
V	VW0 ---VW4094	AI/AO	100 -- 2147

➢ 适用于 CPU504EX 和 CPU205

内存区域	范围	类型	对应的 Modbus 寄存器号*
I	I0.0 --- I4.7	DI	0 --- 39
Q	Q0.0 --- Q4.7	DO	0 --- 39
M	M0.0 --- M1023.7	DI/DO	320 -- 8511
AI	AIW0 --- AIW14	AI	0 --- 7
AQ	AQW0 --- AQW14	AO	0 --- 7
V	VW0 ---VW4094	AI/AO	100 -- 2147

➢ 适用于 CPU506、CPU506EA 和 CPU508

内存区域	范围	类型	对应的 Modbus 寄存器号*
I	I0.0 --- I31.7	DI	0 --- 255
Q	Q0.0 --- Q31.7	DO	0 --- 255

M	M0.0 --- M1023.7	DI/DO	320 -- 8511
AI	AIW0 --- AIW62	AI	0 --- 31
AQ	AQW0 --- AQW62	AO	0 --- 31
V	VW0 ---VW4094	AI/AO	100 -- 2147

- 除以上内存区域支持 MODBUS 访问外，K 系列 PLC 还提供了错误记录的内存区域供用户通过 MODBUS 读取查看，详见附录 D 错误诊断功能

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”的描述指的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		写入个数		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		写入个数		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 CKINCOSerialCom::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　动态修改 RS485 通信口参数

K 系列默认需要在【PLC 硬件配置】中修改各个通信口的参数并下载到 PLC 中才能生效。

同时，K5 也提供了在用户程序中利用特定的 SM 寄存器来动态修改本体 RS485 通信口（PORT1 和 PORT2）参数的功能。PORT0 作为编程口可能会经常使用，因此不允许动态调整通信参数。

1、概述

- 允许动态修改【PLC 站号】、【波特率】和【奇偶校验】这三项参数。
- 动态修改的通信参数值存放在永久存储器中，永久有效。
- 动态修改的通信参数优先级要高于【PLC 硬件配置】中的通信参数。即使用户重新下载了新工程，PLC 也会优先采用存储的动态通信参数。用户可以使用【PLC】->【清除...】菜单命令或者下文提到的清除功能，来清除动态通信参数。
- 用户在程序中修改通信参数后，【PLC 站号】会立即生效，但是【波特率】和【奇偶校验】则不一定：若通信口当时正处于空闲状态，则这 2 个参数会立即生效；若通信口当时正处于通信状态，则这 2 个参数会被保存但不会立即生效。

  在 PLC 下一次上电启动时，所有修改过的通信参数都会生效。

2、寄存器说明

SMB20--SMB25 用于动态修改通信参数。下面详细描述了各个寄存器的含义。

➢ **参数值：SMB23、SMB24 和 SMB25**

SMB	描 述
SMB23	PLC 站号值。数值有效范围：1-31。 若执行写操作，则 SMB23 是将要写入的新 PLC 站号值；若执行读操作，则 SMB23 是读取到的当前使用的 PLC 站号值；若执行清除操作，则 SMB23 被忽略。
SMB24	波特率值。数值有效范围 0-5：0 表示 2400, 1 表示 4800, 2 表示 9600, 3 表示 19200。 若执行写操作，则 SMB24 是将要写入的新波特率值；若执行读操作，则 SMB24 是

	读取到的当前使用的波特率值；若执行清除操作，则 SMB24 被忽略。
SMB25	奇偶校验值。数值有效范围 0-2，0 表示无检验，1 表示奇检验，2 表示偶检验。 若执行写操作，则 SMB25 是将要写入的新奇偶校验值；若执行读操作，则 SMB25 是读取到的当前使用的奇偶校验值；若执行清除操作，则 SMB25 被忽略。

- **控制字节：SMB20 和 SMB21**

位	描 述
SMB20：指定将要操作的端口号和操作方式。	
SM20.7	值为 1 表示启动写操作。PLC 写入新参数完成后会自动将该位清 0。
SM20.6	值为 1 表示启动读操作。PLC 读取参数完成后会自动将该位清 0。
SM20.5	值为 1 表示启动清除操作。PLC 清除参数完成后会自动将该位清 0。
SM20.4	保留，必须赋值为 0。
SM20.3 ∽ SM20.0	这 4 位的组合值表示将要操作的端口号。 1 表示 PORT1，2 表示 PORT2，若设置为其它无效值则 PLC 将报错并退出操作。
SMB21：指定将要操作的通信参数。	
SM21.7 ∽ SM21.3	备用。必须赋值为 0。
SM21.2	值为 1 表示修改或者清除指定通信口的奇偶校验值。
SM21.1	值为 1 表示修改或者清除指定通信口的波特率值。
SM21.0	值为 1 表示修改或者清除指定通信口的 PLC 站号。

在同一时刻，SM20.5、SM20.6、SM20.7 只允许有一个位的值为 1，否则 PLC 将报错并退出操作。

当执行读操作时，SMB21 的值被忽略，PLC 将一次性读取所有的通信参数值。

当执行清除操作时，PLC 允许分别或者同时清除站号、波特率、奇偶校验值。某个参数被清除后，PLC 将会自动采用硬件配置信息中相应的通信参数值。

- **状态字节：SMB22**

SMB22 存放了本次动态调整通信参数的操作结果。

位（只读）	描 述
SM22.7	值为 1 表示本次操作完成。 PLC 完成用户指定的操作后，无论成功还是失败，都会自动将 SM22.7 置 1。只有

	当 SM22.7 的值为 1 时，SMB22 中其它位的值才有实际意义。
SM22.6	若 SM22.7 值为 1，那么若 SM22.6 值为 1，则表示本次操作成功；若 SM22.6 值为 0，则表示本次操作因发生错误而失败。
SM20.5 ∽ SM20.0	若本次操作失败，那么这 6 位的组合值就表示发生的错误代码，含义如下表。

错误代码	错误描述
1	错误的操作命令。比如，SM20.7 和 SM20.6 被同时置为 1。
2	错误的端口号
3	SMB21（将要操作的通信参数）值错误。
4	SMB23（新 PLC 站号）值错误。
5	SMB24（新波特率）值错误。
6	SMB25（新奇偶校验）值错误。
10	从永久存储器中读取存放的 PORT1 动态 PLC 站号失败。
11	尚未设置过 PORT1 的动态 PLC 站号。
12	从永久存储器中读取存放的 PORT1 动态波特率值失败。
13	尚未设置过 PORT1 的动态波特率。
14	从永久存储器中读取存放的 PORT1 动态奇偶校验值失败。
15	尚未设置过 PORT1 的动态奇偶校验值。
20	从永久存储器中读取存放的 PORT2 动态 PLC 站号失败。
21	尚未设置过 PORT2 的动态 PLC 站号。
22	从永久存储器中读取存放的 PORT2 动态波特率值失败。
23	尚未设置过 PORT2 的动态波特率。
24	从永久存储器中读取存放的 PORT2 动态奇偶校验值失败。
25	尚未设置过 PORT2 的动态奇偶校验值。
61	动态通信参数值写入永久存储器失败。

3、使用方法

- 修改PLC的通信参数

1) 将SMB20的低4位设置为将要操作的端口号。

比如，SMB20=B#1表示将要修改PORT1的参数。

2) 根据要修改的参数类型，将相应的值赋给 SMB21。

比如，SMB21=B#16#03 表明了将要修改 PLC 站号和波特率值。

3) 将期望的新参数值赋给相应寄存器：SMB23 是新 PLC 站号值，SMB24 是新波特率值，SMB25 是新奇偶校验值。

比如，SMB23=B#03 表示要将 PLC 站号修改为 3，SMB24=B#3 表示要将波特率修改为 19200。

4) （可选）若已经启动过一次通信参数的操作（读、写或者清除），那么必须先检查完成标志位SM22.7，只有SM22.7为1才可以启动本次的操作。

5) 将SM20.7赋值为1，启动本次的写操作。PLC在完成写入新参数后会自动将SM20.7清零。

6) （可选）检查标志位SM22.7和SM22.6，这两个标志位的值都为1表示修改参数成功。

- 读取PLC的通信参数

1) 将SMB20的低4位设置为将要操作的端口号。

比如，SMB21=B#1表示将要读取PORT1的参数。

2) （可选）若已经启动过一次通信参数的操作（读、写或者清除），那么必须先检查完成标志位SM22.7，只有SM22.7为1才可以启动本次的操作。

3) 将SM20.6赋值为1，启动本次的读操作。PLC在完成读取新参数后会自动将SM20.6清零。

4) 检查标志位SM22.7和SM22.6，这两个标志位的值都为1表示读取参数成功。读取成功后，参数值存放在如下寄存器中：SMB23是PLC站号值，SMB24新波特率值，SMB25是奇偶校验值。

- 清除PLC的通信参数

1) 将SMB20的低4位设置为将要操作的端口号。

比如，SMB21=B#1表示将要修改PORT1的参数。

2) 根据要清除的参数类型，将相应的值赋给 SMB21。

比如，SMB21=B#16#03 表明了将要清除永久存储器中存放的动态 PLC 站号和波特率值。

3) （可选）若已经启动过一次通信参数的操作（读、写或者清除），那么必须先检查完成标志位SM22.7，只有SM22.7为1才可以启动本次的操作。

4) 将SM20.5赋值为1，启动本次的清除操作。PLC在完成清除后会自动将SM20.5清零。

5) （可选）检查标志位SM22.7和SM22.6，这两个标志位的值都为1表示清除参数成功。

4、示例

下面例子是在HMI上修改PORT1和PORT2的动态站号。程序采用了IL语言，用户可以直接将其复制到KincoBuilder的编辑器中并执行【工程】->【LD语言】菜单命令转换为梯形图。

VW48是新站号值，可以在HMI上修改，同时VW48的值也被置于到VW3690中永久存储。PLC程序检查VW48的实时值并与VW3690中存储的值进行比较，若数值发生变化且VW48是允许的合法值，则将VW48作为PORT1和PORT2的新站号值并启动修改。

```
(* Network 0 *)
(*上电启动时，利用永久存储的值来初始化 VW48。*)
LD      %SM0.1
MOVE    %VW3690, %VW48
(* Network 1 *)
(*判断 VB48 值是否变化，且值是否有效。然后保存 VW48 并启动修改。*)
LD      %SM0.0
GE      %VB48, B#1
LE      %VB48, B#31
NE      %VW48, %VW3690
MOVE    %VW48, %VW3690
ST      %M999.7
(* Network 2 *)
(*开始修改 PORT1 站号。*)
LD      %M999.7
R_TRIG
MOVE    B#1, %SMB20
MOVE    B#1, %SMB21
MOVE    %VB48, %SMB23
S       %SM20.7
S       %M999.6
(* Network 3 *)
(*PORT1 参数修改成功后，再修改 PORT2 站号。*)
LD      %M999.6
```

```
AND     %SM22.7
R_TRIG
AND     %SM22.6
MOVE    B#2, %SMB20
S       %SM20.7
R       %M999.6
```

附录 C　数据保持与数据备份

数据保持是指在 CPU 模块 RAM 中的数据在断电后保持为断电瞬间的状态，并供 CPU 在下一次上电的时候使用。CPU 模块内部均提供一个后备锂电池（不可充电）用于数据保持功能。在断电时，后备电池为 RAM 供电并保持 RAM 中的数据。用户需要使用 KincoBuilder 软件在用户工程的【PLC 硬件配置】中选择需保持的数据区类型（如 V

区、C 区等）及起止范围，最大可以保持所有 V 区。常温下，电池典型寿命为 5 年,断电电保持的时间累计不小于 3 年。PLC 提供一个特殊系统内存区 SMW10，存放后备电池的电压值，单位：0.01V。若后备电池的电源持续低于 2.6V 时，PLC 会产生“后备电池电量低”报警。可以通过错误读取对话读取.

数据备份是指 CPU 模块在永久存储器中开辟一个区域，用于存放用户数据，该区域内的数据断电永久不会丢失，并供 CPU 在下一次上电的时候使用。**CPU 模块提供了 E2PROM 存储器用于数据备份功能，由于 E2PROM 只有 100 万次的写入寿命，因此用户注意尽量避免永久备份那些变化频繁的数据!**

K 系列 PLC 在 V 区中提供了数据备份区，该区域中的数据会自动写入永久存储器中，用户可以直接使用这些内存区域。下表列出了 K 系列中的数据备份区。

长 度	448 字节
范 围	VB3648 - VB4095。

需要注意的是对于上表的永久数据备份区域，若同时在【PLC 硬件配置】中也设置为数据保持，那么永久数据备份功能是优先的，也就是说铁电中的数据会覆盖电池供电的 RAM 中的数据。

为了与最早的 Kinco-K3 兼容，K 系列 PLC 在新建工程时会将数据备份区中的 VB3648-VB3902 这 255 字节长度的区域生效，而 VB3903 之后的区域默认不生效，即默认状态下 VB3903 之后的数据不能断电备份。若用户要将 VB3903 之后的区域也生效作为备份区，那么需要在【PLC 硬件配置】中 CPU 模块的【其它】页面进行设置，如下图：

- 【将 VB3648-4095 永久保持】

选中此项则表示 VB3648-4095 将生效作为数据备份区，该区域中的数据会自动写入永久存储器中。

- 【将整个用户工程备份到 PLC 的永久存储中】

K 系列默认在 PLC 中只保存用户工程中的硬件配置和用户程序信息，而变量名称（全局和局部）、程序名称、注释等均不保存。

若选中此项，则在 PLC 中保存完整的用户工程信息，包括硬件配置、用户程序、程序名称、所有变量名称、注释等。用户从 PLC 中上载后得到的工程将与下载的工程完全相同。

附录 D　K5 的错误诊断功能

K5 将运行时发生的错误分为三个等级：致命错误、严重错误、一般错误。当 PLC 在运行时发生错误时，会根据错误的等级采取不同的处理措施，同时将错误码根据发生的先后次序依次存储下来，以供用户读取并进行分析。对于不停发生的同一种错误，PLC 最多连续记录 4 次。

无论曾经发生过的错误等级是多少，我们都建议用户在发现有错误指示之后，及时进行分析、检查，以免造成不必要的损失！

1、错误类型

- **致命错误**

致命错误发生的原因是 PLC 检测到硬件芯片发生了未知的、可能危及正确运行的故障，或者 PLC 的看门狗被触发（比如程序中 FOR 循环量过大、JMP 指令造成了死循环等等）。致命错误可能会导致 PLC 无法继续工作，并且我们也无法确认 PLC 是否会再次发生不可预期的、危险的错误，因此，处理致命错误的目标是立即让 PLC 进入安全状态。

当发生致命错误后，PLC 会自动采取如下措施：立即退出正常的扫描状态，并根据 SM2.0 的值来直接复位或者进入独立的安全子系统运行。SM2.0 的值决定了当发生致命错误时 PLC 的动作：若值为 0，则发生致命错误时，PLC 会进入独立的安全子系统运行；若值为 1，则发生致命错误时，PLC 直接复位重新启动。

下面是关于安全状态的描述。

- 所有的输出点（DO、AO）立即输出用户在【PLC 硬件配置】中定义的“停机输出”值。
- STOP 指示灯常亮，ERR 指示灯闪烁，提示用户发生了致命错误。（**注意如果 SM2.0==1， PLC 发生致命错误时，PLC 会直接重启，用户可能看不到 STOP 常亮， ERR 指示灯闪烁**）

- 记录故障点和故障代码，并允许用户使用专用软件读取这些记录信息。注：由于致命错误会导致 PLC 不能正常运行，因此这些故障信息有可能记录不下来。

➢ **严重错误**

严重错误会导致 PLC 无法执行某项或者几项重要功能，不能继续正确运行用户程序，但是其结果是可以预期的。当发生严重错误时，PLC 会自动采取如下措施：

1) 立即将 PLC 置于 STOP 状态，所有输出点（DO、AO）立即输出用户设置的“停机输出”值。
2) ERR、STOP 指示灯长亮。
3) 依次记录故障代码，并允许用户通过 KincoBuilder 和 Modbus RTU 协议读取这些记录。

➢ **一般错误**

一般错误是 PLC 执行某项功能时发生错误，但 PLC 能够容忍这种错误，可以继续正确运行其它部分用户程序，其结果是可以预期的。当发生一般错误时，PLC 会自动采取如下措施：

1) PLC 继续运行。
2) ERR 指示灯长亮。
3) 依次记录故障代码，并允许用户通过 KincoBuilder 和 Modbus RTU 协议读取这些记录。

2、错误代码

代码	描述
严重错误	
20	在“PLC 硬件配置”中的 CPU 类型与实际连接的 CPU 类型不一致。
21	在“PLC 硬件配置”中存在错误的扩展模块（使用了其它系列的模块）。
25	上电时，读取 PLC 保护类型失败。
26	上电时，读取目标文件（明文）失败。
27	上电时，读取目标文件（密文）失败。
28	上电时，目标文件 CRC 校验错误。
29	上电时，检查 PLC 程序中有未知指令。
30	上电时，检查 PLC 程序的参数个数超过限制。

35	上电时，读取永久存储区数据失败。
40	运行时，JMP 指令跳转失败。
41	运行时，子程序调用失败。
42	运行时，中断子程序调用失败。
60	上电时，第 1 个扩展模块超时没有响应。
61	上电时，第 1 个模块响应错误。
62	第 1 个扩展模块与硬件配置中的类型不一致。
65	上电时，第 2 个扩展模块超时没有响应。
66	上电时，第 2 个模块响应错误。
67	第 2 个扩展模块与硬件配置中的类型不一致。
70	上电时，第 3 个扩展模块超时没有响应。
71	上电时，第 3 个模块响应错误。
72	第 3 个扩展模块与硬件配置中的类型不一致。
75	上电时，第 4 个扩展模块超时没有响应。
76	上电时，第 4 个模块响应错误。
77	第 4 个扩展模块与硬件配置中的类型不一致。
80	上电时，第 5 个扩展模块超时没有响应。
81	上电时，第 5 个模块响应错误。
82	第 5 个扩展模块与硬件配置中的类型不一致。
85	上电时，第 6 个扩展模块超时没有响应。
86	上电时，第 6 个模块响应错误。
87	第 6 个扩展模块与硬件配置中的类型不一致。
90	上电时，第 7 个扩展模块超时没有响应。
91	上电时，第 7 个模块响应错误。
92	第 7 个扩展模块与硬件配置中的类型不一致。
100	上电时，第 8 个扩展模块超时没有响应。
101	上电时，第 8 个模块响应错误。
102	第 8 个扩展模块与硬件配置中的类型不一致。
105	上电时，第 9 个扩展模块超时没有响应。
106	上电时，第 9 个模块响应错误。
107	第 9 个扩展模块与硬件配置中的类型不一致。
110	上电时，第 10 个扩展模块超时没有响应。

111	上电时，第 10 个模块响应错误。
112	第 10 个扩展模块与硬件配置中的类型不一致。
115	上电时，第 11 个扩展模块超时没有响应。
116	上电时，第 11 个模块响应错误。
117	第 11 个扩展模块与硬件配置中的类型不一致。
120	上电时，第 12 个扩展模块超时没有响应。
121	上电时，第 12 个模块响应错误。
122	第 12 个扩展模块与硬件配置中的类型不一致。
95	上电时，CPU 模块发送扩展通信报文失败。
96	上电时，CPU 模块扩展总线进入错误被动状态。
97	上电时，CPU 模块扩展总线进入总线关闭状态。
一般错误	
136	上电时，AI 通道的校准值读取失败。
137	上电时，AO 通道的校准值读取失败。
138	校准时，AI 通道的校准值写入失败。
139	校准时，AO 通道的校准值写入失败。
150	运行时，第 1 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
151	运行时，收到第 1 个扩展模块的故障报文。
154	运行时，第 2 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
155	运行时，收到第 2 个扩展模块的故障报文。
158	运行时，第 3 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
159	运行时，收到第 3 个扩展模块的故障报文。
162	运行时，第 4 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
163	运行时，收到第 4 个扩展模块的故障报文。
166	运行时，第 5 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
167	运行时，收到第 5 个扩展模块的故障报文。
170	运行时，第 6 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
171	运行时，收到第 6 个扩展模块的故障报文。
174	运行时，第 7 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
175	运行时，收到第 7 个扩展模块的故障报文。
178	运行时，第 8 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
179	运行时，收到第 8 个扩展模块的故障报文。

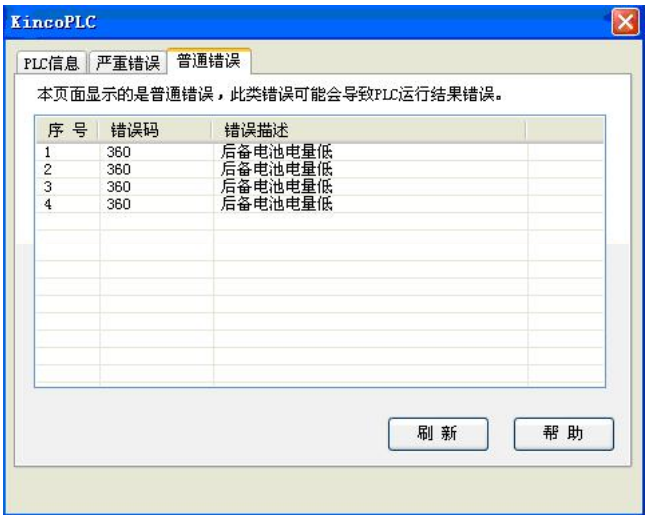
182	运行时，第 9 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
183	运行时，收到第 9 个扩展模块的故障报文。
186	运行时，第 10 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
187	运行时，收到第 10 个扩展模块的故障报文。
190	运行时，第 11 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
191	运行时，收到第 11 个扩展模块的故障报文。
194	运行时，第 12 个扩展模块心跳超时，该模块的扩展总线通信可能中断。
195	运行时，收到第 12 个扩展模块的故障报文。
300	运行时，本体 AI 通道曾经发生过 DMA 错误。
301	运行时，本体 AI 通道的采样转换过程曾经停止过。
320	运行时，扩展总线通信曾经发生过帧格式错误。
321	运行时，扩展总线通信曾经进入错误错误主动状态。
322	运行时，扩展总线通信曾经进入错误错误被动状态。
323	运行时，扩展总线曾经关闭过。注：总线关闭是由于总线上通信错误过多造成的。
324	运行时，扩展通信故障：接收缓冲区满。
325	运行时，扩展通信故障：发送缓冲区满。
326	运行时，扩展通信故障：发送报文失败。
327	运行时，检测到 CANOpen 从站故障（心跳超时或者节点保护超时、SDO 无回应等）。
329	运行时，发生过如下错误：被 0 除。
330	运行时，发生过如下错误：类型转换指令（I_TO_B、DI_TO_I）溢出。
331	运行时，发生过如下错误：LN 指令输入值为 0 或者负数。
332	运行时，发生过如下错误：LOG 指令输入值为 0 或者负数。
333	运行时，发生过如下错误：SQRT 指令输入值为负数。
334	运行时，发生过如下错误：I_TO_BCD 指令无效的输入值。
335	运行时，发生过如下错误：A_TO_H 指令无效的输入值。
336	运行时，发生过如下错误：R_TO_A 指令无效的输入值。
341	运行时，发生过如下错误：FOR 指令无效的输入值。
350	运行时，发生过如下错误：保存永久存储数据失败。
351	上电时，检测到 RAM 中的掉电保持数据丢失。
360	后备电池电量低报警。

3、如何读取 PLC 中曾经发生的错误

当 PLC 发生错误之后，会依次自动记录下错误代码，用户可以利用如下方法来读取一般错误和严重错误信息。致命故障信息只能使用步科公司提供的专用软件来读取。

1) 使用 KincoBuilder

用户可以通过编程口（PORT0），在 Kincobuilder 中执行【PLC】->【PLC 严重错误…】或者【PLC 普通错误…】菜单命令，在对话框窗口中查看，如右图。在对话框中，用户可以单击【刷新】按钮，刷新显示 PLC 内最新的错误记录信息。



2) 使用 Modbus RTU 通信

用户可以使用Modbus RTU通信命令（所用功能码 03、04），通过 Port0、Port1 或者 Port2 通信口来读取错误记录信息。用户读取到的是 PLC 记录的错误码，以字的形式来读取，一次可以读取一条或者多条记录。

错误记录的 Modbus 寄存器地址如下表：

Modbus 寄存器	描述
9000-9127	PLC 本次上电后，最近发生的 128 个**普通**错误的代码。 其中，9000 为最新一次的错误，9001 为次新的错误，依此类推。
9128-9255	PLC 本次上电后，最近发生的 128 个**严重**错误的代码。 其中，9128 为最新一次的错误，9129 为次新的错误，依此类推。
9256-9383	PLC 在上一次上电过程中，最后发生的 128 个**普通**错误的代码。 其中，9256 为最新一次的错误，9257 为次新的错误，依此类推。
9384-9511	PLC 在上一次上电过程中，最后发生的 128 个**严重**错误的代码。 其中，9384 为最新一次的错误，9385 为次新的错误，依此类推。

4、错误寄存器

针对错误处理，K 系列 PLC 在 SM 区提供了一个控制字节和各种错误状态寄存器。当 PLC 发生错误时会自动设置相应错误寄存器的值。用户程序可以直接读取这些寄存器值并进行相应的处理。

- SMB2：控制字节

位（可读写）	功能描述
SM2.0	其值决定了当发生致命错误时 PLC 的动作。上电后初始值为 0。 若值为 0：发生致命错误时，PLC 进入独立的安全子系统运行。 若值为 1：发生致命错误时，PLC 直接复位。

- SMB0 和 SMB1：内存、指令错误

SM	功能描述
SMB0（只读）	
SM0.2	若上电后 PLC 检测到 RAM 中的掉电保持数据丢失，则将该位置 1，否则置 0。
SMB1（只读）	
SM1.0	1 表示发生过如下错误：DIV、MOD 指令被 0 除。
SM1.1	1 表示发生过如下错误：LN、LOG、SQRT 指令参数为非法值（0 或者负数）。
SM1.2	1 表示发生过如下错误：I_TO_B、DI_TO_I 指令转换结果溢出。
SM1.3	1 表示发生过如下错误：I_TO_BCD 指令无效的输入 BCD 码。
SM1.4	1 表示发生过如下错误：A_TO_H 指令输入字符串中有不可识别的字符。
SM1.5	1 表示发生过如下错误：R_TO_A 指令转换结果溢出。
SM1.6	1 表示发生过如下错误：FOR 指令输入参数值无效。

- SMB3，SMB5 和 SMB96-SMB110：扩展模块故障(K2 系列不支持)

若 PLC 检测到扩展总线通信故障或者收到扩展模块发送的故障报文，则 PLC 会设置相应的扩展扩展总线故障标志位，同时将故障代码存放在相应的扩展总线故障寄存器中供查询。若没有检测到故障，则故障标志位和故障寄存器被设置为 0。

注意：若 PLC 在启动过程中检测到扩展总线或者扩展扩展模块故障，那么将进入 STOP 状态，

同时点亮 ERR 灯，但不设置相应的寄存器，因为此时 CPU 不执行用户程序，就无法读取寄存器值。

SM	功能描述
SMB3，SMB5（只读）：扩展总线错误标志	
SMB3.0	若第 1 个扩展模块发生故障，则该位被置 1。
SMB3.1	若第 2 个扩展模块发生故障，则该位被置 1。
SMB3.2	若第 3 个扩展模块发生故障，则该位被置 1。
SMB3.3	若第 4 个扩展模块发生故障，则该位被置 1。
SMB3.4	若第 5 个扩展模块发生故障，则该位被置 1。
SMB3.5	若第 6 个扩展模块发生故障，则该位被置 1。
SMB3.6	若第 7 个扩展模块发生故障，则该位被置 1。
SMB3.7	若第 8 个扩展模块发生故障，则该位被置 1。
SMB5.0	若第 9 个扩展模块发生故障，则该位被置 1。
SMB5.1	若第 10 个扩展模块发生故障，则该位被置 1。
SMB5.2	若第 11 个扩展模块发生故障，则该位被置 1。
SMB5.3	若第 12 个扩展模块发生故障，则该位被置 1。
SMB5.7	若 CPU 检测到扩展总线通信故障，则该位被置 1。
SMB96 - SMB110（只读）：扩展总线错误代码	
SMB96	若第 1 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB97	若第 2 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB98	若第 3 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB99	若第 4 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB100	若第 5 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB101	若第 6 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB102	若第 7 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB103	若第 8 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB104	若第 9 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB105	若第 10 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB106	若第 11 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB107	若第 12 个扩展模块发生故障，则存放其故障码。含义见表 1。
SMB110	若 CPU 端的扩展总线发生故障，则存放其故障码。含义见表 2。

故障码	描 述

0	无故障。
6	在运行过程中，扩展模块心跳报文超时。扩展模块会定时向 CPU 模块发送心跳报文，若 CPU 超时未收到该模块的心跳报文，则说明该模块可能通信异常。
10	AI 通道（电压、电流输入）的 ADC 转换出现一次错误。
11	模块保存校准值错误。
12	模块读取校准值错误。
14	模拟量输入模块第 1 个通道输入信号超出设定的测量范围。
15	模拟量输入模块第 2 个通道输入信号超出设定的测量范围。
16	模拟量输入模块第 3 个通道输入信号超出设定的测量范围。
17	模拟量输入模块第 4 个通道输入信号超出设定的测量范围。

表 1　扩展模块故障码含义

故障码	描 述
0	无故障。
1	通信帧格式错误。
2	扩展总线进入错误报警状态。
3	扩展总线进入错误被动状态。
4	扩展总线进入了总线关闭状态，并刚刚自动恢复。
5	扩展总线接收缓冲区满。
6	扩展总线发送缓冲区满。
7	CPU 发送报文失败。

表 2 CPU 模块扩展总线通信故障码含义

5、如何将 CPU 恢复至出厂的初始状态？

PLC 提供了如下方法将 CPU 恢复至出厂的初始状态：

下面的操作会清除 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。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**【PC 机通讯设置】**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 3.7 如何连接计算机与 Kinco-K5 。

③ 执行清除命令。

进入 KincoBuilder，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。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。

6、故障现象：PLC 上电后，RUN 或 STOP 灯闪烁。

可能的原因和处理方法：

- 用户将 PLC 的 SM2.0 位置 1。

在这种前提下， 每当 PLC 发生致使错误， PLC 会自动重新上电重启， 如果致命错误不停发生， 则 PLC 会不停重启， 表面现象就是 RUN 灯不停闪烁。 这种原因一般不会导致 STOP 灯闪烁，用户可以在 STOP 档下， 重新下载 PLC 程序，或者清除 PLC， 或做其它处理。

- 某些 OEM 定制的 PLC， 在 PLC 内有专用的用户程序时， 如果更新固件到标准 PLC 固件。

上述情况下，可能会发生 RUN, STOP 闪烁， 解决方法， 在更新固件之前，先用 Kincobuilder 清除 PLC。 如果 PLC 在 STOP 情况下， 灯也闪烁， 则只能将 PLC 先刷新成与 PLC 内用户程序相匹配的固件， 让 PLC 先正常运行， 然后清除 PLC。STOP 灯闪烁的情况， 可以直接联系我们的技术支持。

- 硬件损坏。

硬件损坏分为好多种，表现会各有不同，且不可预期。 比如 EEPROM 损坏， 会导致致命错误，但大多情况下不会导致 RUN,STOP 灯闪烁。 内存芯片损坏， 则有很大可能会导致 RUN, STOP 灯闪烁。 MCU 主芯片损坏， 则可能所有灯都不亮。 一般正常使用， 没有雷击或者在野外使用电压不稳， 很少会出现硬件损坏， 应该优先查找上面的软件原因。

7、故障现象：PLC 上电后，RUN STOP COMM ERR 灯全亮。

原因是 PLC 的上电安全自检失败：

- PLC 的 EEPROM 完全损坏。

EEPROM 中存放着 PLC 的各种数据， 包括允许 PLC 启动的数据区， 当 EEPROM 损坏时， PLC 会无法读取到启动代码。

- 用户自己更换过 EEPROM。

EEPROM 中存放着 PLC 的各种数据， 包括允许 PLC 启动的数据区， 用户更换的新 EEPROM 中，数据缺失， PLC 会无法读取到启动代码。

- 用户自己更换过 MCU。

禁止用户自己更换 MCU, 或者 EEPROM, 这是为了保护设备制造厂商的知识产权。

- 处理方法。

如果用户自己更换过芯片， 则把芯片换回就可以正常。

其它情况， 只能返厂维修。

附录 E　常用系统存储器 SM 区的定义

本附录详细描述了常用的系统存储区 SM 区中的相关定义。系统存储区是用来辅助 Kinco-K 系列 PLC 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。 需要注意的是不是所有 CPU 都支持下边所有的 SM 去，比如某些 CPU 本体不自带模拟量，自然也就不支持 SM2.1 的功能

1、SMB0：系统状态字节

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	首次扫描位。 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	若 RAM 中的掉电保持数据丢失，则在首次扫描中 PLC 将该位置 1，之后清 0。
SM0.3	周期为 1s 的连续脉冲串，占空比为 50%。
SM0.4	周期为 2s 的连续脉冲串，占空比为 50%。
SM0.5	周期为 4s 的连续脉冲串，占空比为 50%。
SM0.6	周期为 60s 的连续脉冲串，占空比为 50%。

2、SMB2：系统控制字节

位（可读写）	描 述
SM2.0	其值决定了当发生致命错误时 PLC 的动作。上电后初始值为 0。 若值为 0：发生致命错误时，PLC 进入独立的安全子系统运行。 若值为 1：发生致命错误时，PLC 直接复位。

SM2.1	其值决定了系统中 AI、AO 通道的工作状态。上电后初始值为 0。 若值为 0：本体的 AI、AO 通道正常工作。 若值为 1：本体的 AI、AO 通道进入校准状态。

3、通信口复位功能

K 系列 PLC 提供了复位通信口（PORT0、PORT1 和 PORT2）的功能。执行复位功能，K5 会清除通信口用到的内部缓冲区并重新初始化、启动该通信口。复位之后，通信口的参数、功能，比如站号、波特率、主/从站选项等，均保持原来的值不变。

➢ 控制寄存器和状态寄存器

位			值	描 述
PORT 0	PORT 1	PORT 2		
SM87.0	SM187.0	SM287.0	1	将这 2 位设置为前面列中的值后，再调用 RCV 指令（参数 PORT 指定了通信口），就会复位指定的通信口。
SM87.7	SM187.7	SM287.7	0	
SM4.0	SM4.1	SM4.2	-	复位成功后，K5 将该位赋值为 1。该位需用户自己去复位。

➢ 复位方法（以 PORT0 为例）

1) （可选）将SM4.0赋值为0。

2) 将SM87.7赋值为0，将SM87.0赋值为1。

3) 调用RCV指令，其PORT参数赋值为待操作的通信口编号，本例中需要赋值为0。

4) （可选）检测标志位SM4.0，若为1则表示复位成功，可按需要对该通信口进行其它操作。

➢ 示例

下面举例说明如何复位PORT1。示例程序采用了IL语言，用户可以将其复制到KincoBuilder的编辑器中并执行【工程】->【LD语言】菜单命令转换为梯形图。

```
(* Network 0 *)
(*利用I0.0的上升沿来触发PORT1的复位。*)
(*RCV指令中的TBL参数对复位无影响，可以使用任意值。*)
LD      %I0.0
```

```
R_TRIG
MOVE    B#0, %SMB4
AND     B#16#7F, %SMB187
OR      B#16#1, %SMB187
RCV     %VB222, 1
(* Network 1 *)
(*复位成功后，用户可以延时让PORT1稳定后再进行其它操作。理论上，不延时也可以。*)
(*在本例子中，复位PORT1之后就再次让PORT1进入了接收状态。*)
LD      %SM4.1
TON     T4, 3
OR      B#16#80, %SMB187
RCV     %VB222, 1
R       %SM4.1
```

4、其它常用功能变量

SM	描 述
SMB6	只读。存放最近一次的 PLC 扫描时间，单位：ms。
SMW10	只读。存放后备电池的电压值，单位：0.01V。 若后备电池的电源持续低于 2.6V 时，PLC 会产生“后备电池电量低”报警。
SMB274-SMB285	这 12 个字节的组合值表示本 CPU 模块的 ID 值。 出厂时每个 CPU 模块均被赋予一个唯一的 ID 值，各个模块 ID 值不会相同。

5、SMD12 和 SMD16：定时中断的周期

K 系列 PLC 提供了两个 0.1ms 时基的定时中断：定时中断 0，中断号为 3；定时中断 1，中断号为 4。

SMD12 用于指定定时中断 0 的周期值，单位 0.1ms。若将 SMD12 设置为 0，则定时中断 0 被禁止。SMD12 的缺省值为 0。

SMD16 用于指定定时中断 1 的周期值，单位 0.1ms。若将 SMD16 设置为 0，则定时中断 1 被禁止。SMD16 的缺省值为 0。

定时中断会周期性的产生，可以利用它来完成周期性的任务。定时中断不受 PLC 扫描周期的影响，可以用于精确的定时。

附录 F　CANOpen 主站功能的使用

KINCO K5 系列 PLC 支持 CANopen 主站功能，CANOpen 总线具有开放性好、可靠性高、实时性较好、抗干扰能力强、成本低等优势，是工业控制中一种常用的现场总线，目前应用越来越广泛。

1、CANOpen 通信对象简介

CANOpen 应用层和通信规范（CiA DS301）是 CANOpen 协议的核心，适用于所有的 CANopen 设备。 在 DS301 中定义了多种 CANOpen 通信对象，同时也详细描述了这些对象的服务和协议。为了方便用户的应用，下面我们将介绍几种关键的对象及其通信协议。

1.1 网络管理（NMT）

网络管理（NMT）面向 CANOpen 设备，采用了“主站-从站”结构。NMT 服务可以初始化、启动、监视、复位或者停止 CANOpen 设备。在一个网络内必须存在一个 NMT 主站，主站拥有整个网络的控制权，即网络管理类（NMT）功能。下面介绍几种常用的 NMT 服务。

1.1.1 NMT 节点控制（NMT Node Control）

NMT 主站通过 NMT Node Control 报文来控制各从站的 NMT 状态（包括停止、预操作、操作和初始化）。从站设备必须支持 NMT 节点控制服务。NMT 节点控制报文格式如下：

COB-ID	Byte 0	Byte 1
0x000	CS(Command Specifier)	Node ID

其中， Node ID：目标从站的 ID。若 Node ID 为 0，则表示网络上所有的从站都需要执行本命令。

CS： 命令字，不同数值的含义为：1　表示　启动目标节点；

2　表示　停止目标节点；

128　表示　目标节点进入预操作状态；

129　表示　复位目标节点

130　表示　目标节点复位通信参数

1.1.2 NMT 错误控制（NMT Error Control）

错误控制服务用于检测CANOpen网络故障,包括节点保护(Node Guarding)和心跳(Heartbeat)两种方式。在实际应用中，必须为一个节点选择一种错误控制方式。顺便提一下，心跳服务是在DS301 后期的版本中新增加的，CiA 推荐使用。

- **NMT 节点保护（NMT Node Guarding）**

NMT 主站发送远程帧（无数据）：

COB-ID
0x700 + Node ID

NMT 从站发送如下应答报文：

COB-ID	Byte 0
0x700 + Node ID	Bit7：触发位，必须在每次节点保护应答中交替置“0”或者“1”。 Bit0-6：组合的数值表示从站状态。其中，0 表示 Boot-up，4 表示 STOPPED；5 表示 Operational；127 表示 Pre-Operational。

- **心跳（NMT Node Guarding）**

若一个节点被配置为心跳生产者，它会周期性地发送心跳报文。网络中另外一个或者多个节点作为心跳消费者，来处理各生产者的心跳报文。通常，主站作为心跳消费者，其它从站作为心跳生产者。心跳报文格式如下：

COB-ID	Byte 0
0x700 + Node ID	本节点的状态值。其中，0 表示 Boot-up，4 表示 STOPPED；5 表示 Operational；127 表示 Pre-Operational。

1.2 服务数据对象（SDO，Service Data Object）

通过使用索引（index）和子索引（sub-index），SDO 使一个 CANOpen 设备（作为客户机）可以直接访问其它 CANOpen 设备（作为服务器）的对象字典中的对象。通常，主站作为客户机。

SDO 有两种传输机制：加速传输，每次最多传输 4 字节数据；分段传输，允许分段传输超过 4 个字节的数据。下面简单介绍一下加速传输机制 SDO 的报文格式。

请求报文，Client -> Server：

COB-ID	Byte 0	Byte 1-2	Byte 3	Byte 4-7
0x600 + Node ID	SDO 命令字	对象索引	对象子索引	数据

应答报文，Server -> Client：

COB-ID	Byte 0	Byte 1-2	Byte 3	Byte 4-7
0x580 + Node ID	SDO 命令字	对象索引	对象子索引	数据

1.3 过程数据对象（PDO，Process Data Object）

PDO 用于传输实时的数据，一个 PDO 报文中最多包含 8 个字节的数据。

PDO 通信是基于“生产者-消费者”模型。以发送数据或者接收数据来区分，PDO 分为发送 PDO（TPDO）和接收 PDO（RPDO）。生产者支持 TPDO，消费者支持 RPDO。

PDO 通信没有协议规定，一个 PDO 报文中包含的内容是预先定义好的。在网络组态时，用户就定义了每个 PDO 的 COB-ID 和其中映射的对象,因此,生产者和消费者都能知道相应 PDO 的内容，从而对报文进行解析。

每个 PDO 在对象字典中由通信参数和映射参数来描述。下面介绍 PDO 的通信参数。

- **COB-ID**

  指明了该 PDO 使用的 COB-ID。

- **传输类型**

  指明了该 PDO 发送（或接收）的触发方式。它是一个 8 位无符号整数值。

  传输类型分为如下几类：

- 同步方式：根据 SYNC 对象的计数值来触发发送（或接收）。传输类型的值为 0 表示“同步，非循环”方式，值为 1-240 表示“同步，循环”方式。
- RTR-Only：仅适用于 TPDO，由接收到的 RTR 报文来触发 PDO 的发送。传输类型的值为 252 意味着接收到 SYNC 和 RTR 之后就发送 PDO。值为 253 意味着接收到 RTR 之后立即发送 PDO。
- 事件驱动：当 CANOpen 设备内部事件发生之后就立即发送 PDO。传输类型值为 254 表示是设备制造商自定义的事件。值为 255 表示是设备子协议和应用层协议定义的事件，一般是指 PDO 内的数据值改变或者定时器定时时间到。

➢ **禁止时间**

禁止时间定义了该 PDO 连续发送时的最小间隔时间。配置禁止时间是为了避免由于高优先级 PDO 发送过于频繁，始终占据总线，而使其它优先级较低的报文无法使用总线的问题。

➢ **事件定时器**

用于指定一个定时发送的周期值。它是一个 16 位无符号整数，单位是 ms。PDO 将以该定时值为周期来触发发送。若该数值为 0，则表示不使用事件定时器。

2、Kinco-K5 的 CANOpen 主站功能

在 Kinco-K5 中，除了 CPU504 之外，其它 CPU 与 K541（CAN 通信模块）配合，都能提供 CANOpen 主站功能，作为 CANOpen 主站使用。

2.1 参数介绍

- 采用 CAN2.0A 标准。符合 CANOpen 标准协议 DS301 V4.2.0。
- 支持 NMT 网络管理服务，包括 NMT Node Control 和 NMT Error Control，并作为 NMT 主站。
- 最大支持 72 个 CANOpen 从站。允许用户在 KincoBuilder 中为每个从站配置启动过程；
- 每个从站最多支持 8 个 TPDO 和 8 个 RPDO；总共最多支持 256 个 TPDO 和 256 个 RPDO。
- 支持客户端 SDO，并提供 SDO 读、写指令，SDO 指令支持标准的加速传输模式；
- 支持 CANOpen 预定义的紧急报文。

- 网络故障管理功能。

2.2 使用方法

2.2.1 CANOpen 网络配置工具

在 KincoBuilder 中，进入【PLC 硬件配置】，在窗口上部的表格中选中 CPU 模块，然后在窗口下部的页面中单击【CANOpen 主站】，就进入了 CANOpen 的网络配置页面，可以对网络和各个设备的参数进行配置。

2.2.2 处理 EDS 文件

在【CANOpen 主站】->【网络配置】页面中，提供了如下按钮可以对 EDS 文件进行操作：

- 【导入 EDS】：单击此按钮，选择所需的 EDS 文件，就将其导入到 Kincobuilder 中并存储。导入一个 EDS 之后，相应的从站设备就在显示在下方的【所有从站模块列表】中，之后就可以对相关从站进行组态。
- 【删除】： 在下方的【所有从站模块列表】中选择一个从站设备，然后单击【删除】按钮，就可以将该设备从列表中删除，同时也将它的 EDS 文件从 Kincobuilder 中删除。
- 【导出所有 EDS】：可以将 Kincobuilder 中已有的从站模块 EDS 文件全部合并导出到一个文件中（扩展名为 .ALLEDS）用作备份。这个功能在卸载 Kincobuilder 时会比较有用，卸载时会删除 Kincobuilder 安装目录下的所有文件，所以用户在卸载前可以使用【导出所有 EDS】这个功能将所有从站的 EDS 文件备份，以后可以将备份的.ALLEDS 文件直接导入即可。
- 【导入所有 EDS】：可以将一个 EDS 备份文件（扩展名为 .ALLEDS ）导入到 Kincobuilder 中，导入之后该文件中包含的所有从站设备都将显示在下方的【所有从站模块列表】中。

2.2.3CANOpen 网络配置过程

1）配置全局参数

进入【主站及全局配置】页面，如下图：

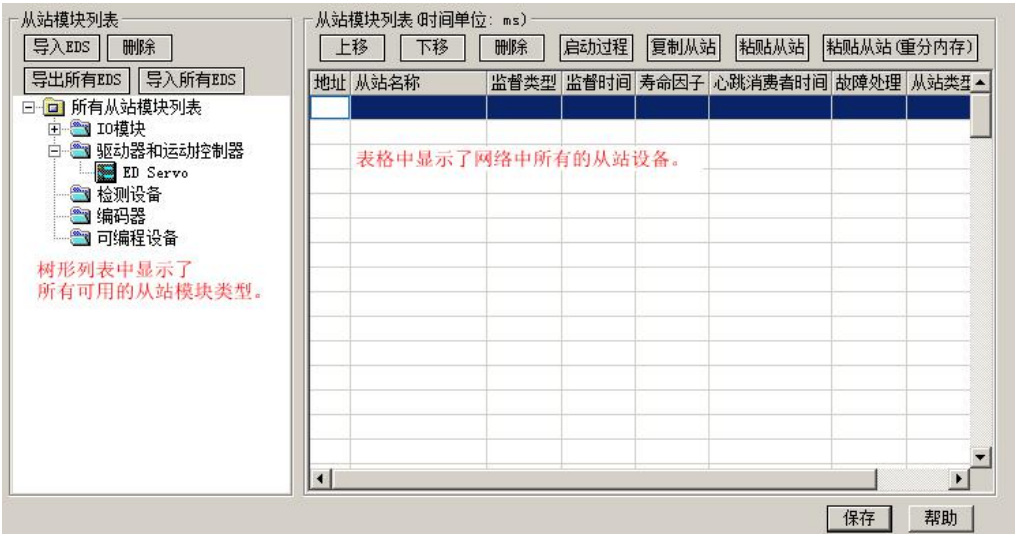
【波特率】：选择主站所用的波特率。注意网络上所有节点的波特率必须一致。

【SDO 超时】：设定主站发送 SDO 请求报文之后的超时等待时间，若超过这个时间没有收到相应从站的应答报文，则会报告错误。SDO 超时值设定一般不需要超过 100ms。

【启动时配置各从站】：若选中此项，那么主站除了控制各个从站的 NMT 状态转换外，在启动时，主站还会根据各个从站的参数组态情况依次发送相应的配置命令来对各个从站进行配置（比如从站的错误控制方式、PDO 映射等）。若不选中此项，则主站仅仅控制各个从站的 NMT 状态转换。

2）配置各从站

进入【网络配置】页面，继续配置网络上的从站节点及其参数，如下图：



页面中所有的功能按钮，都有相应的右键菜单命令。用户在相关位置单击鼠标右键，就会弹出相应的右键菜单，此时可以使用菜单命令。下面的过程中不再赘述。

a） 向网络中添加一个从站设备：

从左侧的树形列表中双击需要加入网络的从站类型，就会添加一个该类型的从站设备

到网络中去，并显示在右侧的表格中。

b) **配置从站设备的站号（ID）、监督类型等参数：**

右侧表格中的【地址】列就是从站的站号（ID）。第 1 行是 10 号站的位置。

添加一个从站设备后，就会显示出它的默认配置参数。添加时，Kincobuilder 默认是将设备从上到下依次添加到表格中。用户可以鼠标单击表格中的行来选中一个从站，然后可以单击【上移】、【下移】按钮来调整它的站号，也可以单击【删除】按钮将此设备从网络中删除。

【监督类型】用于配置该节点的 NMT Error Control 方式，包括节点保护和心跳两种方式。若从站设备同时支持这两种方式，推荐优先选择使用心跳方式。

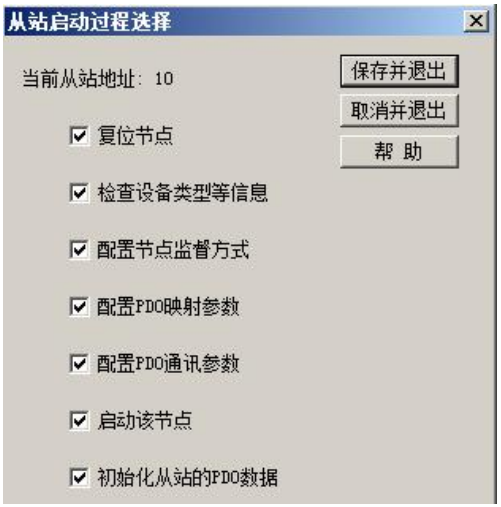
【监督时间】表示前面所选的节点保护方式或者心跳方式的周期值。建议在实际应用中，这个周期值设置不要太小，比如可以设置在 2000 以上。

【心跳消费者时间】主站会定时查询是否收到从站的心跳报文，若超过这个“心跳消费者”时间仍然没有收到，则认为该从站已经离线并进行相应的故障处理。建议在实际应用中，这个周期值设置不要太小，比如可以设置在 2000 以上。

【故障处理】用于选择当主站检测到该从站故障后采用的处理方式，包括“无”、“停止节点”和“停止网络”三种选项。主站能够检测的故障包括 SDO 命令超时没有应答、节点保护或者心跳报文超时、收到从站发送的部分类型的紧急报文等。

c) **配置从站的启动过程：**

鼠标单击表格中的一个从站，然后单击【启动过程】，就可以选择在网络启动过程中主站需要对该从站进行何种配置。



【复位节点】：主站在向从站发送配置命令之前，是否先发送“复位节点”命令。

【检查设备类型】：主站在向从站发送配置命令之前，是否先读取设备信息进行检查。

【配置节点监督方式】：主站是否需要配置从站的监督类型及其参数。

【配置 PDO 映射参数】：主站是否需要配置从站的 PDO 映射参数。

【配置 PDO 通信参数】：主站是否需要配置从站的 PDO 通信参数。

【启动该节点】：配置完成后，主站是否需要向该从站发送“启动节点”命令。

【初始化从站的 PDO 数据】：在启动该从站后，主站是否需要把该从站所有 RPDO 中的数据全部置为 0 并立即全部发送一次。

d) **配置各从站的 PDO：**

进入【对象字典映射】页面，为网络中所有的从站配置 PDO。

页面中，左侧部分【已添加从站列表】显示了已经加入到网络的所有从站，以及各从站对象字典中允许映射到 PDO 中的对象。其中,【发送 PDO】中的对象只能映射到该从站的 TPDO 中，【接收 PDO】列表中的对象只能映射到该从站的 RPDO 中。

配置 PDO 的过程如下：

i) 在【已添加从站列表】中鼠标单击一个从站，那么在右侧就会显示出该从站所有的 PDO。

ii) 在右侧的表格中，选中一个 PDO，可以修改其定时时间、禁止时间等通信参数。其中，从站中前 TPDO1-4 和前 RPDO1-4 的 COB-ID 不允许修改，采用了 DS301 中预定义连接集中的默认值。TPDO5-8 和 RPDO5-8 的 COB-ID 允许用户自己输入合法的值。

另外，在左侧的对象列表中，双击一个对象，就会将这个对象加入到了当前 PDO 中，同时 Kincobuilder 会自动为这个对象分配一个 PLC 的 V 区地址，比如 VW1006，用户在程序中操作这个 V 区地址就相当于操作相应的对象了。

iii) 重复执行上述操作，直到配置完成当前从站的所有 PDO。

3） 复制从站、粘贴从站

在【网络配置】页面中，提供了【复制从站】、【粘贴从站】和【粘贴从站（重分内存）】这 3 个按钮。如下图。

【复制从站】：选中一个已经配置好的从站，然后单击此按钮，就会复制该从站的所有信息（其中包括所有 PDO 的通信参数、映射参数等）。如果所选从站没有配置任何 PDO，那么复制失败并提示相应信息。

【粘贴从站】：复制成功一个从站后，单击选中表格中的一个空行，然后单击此按钮，就会将刚才复制的从站信息粘贴到该行并生成一个新从站。注意：新从站 PDO 中的各映射对象对应的 PLC 内存地址，还是保持与源从站中的一致，没有重新分配，用户需要自己修改。

【粘贴从站（重分内存）】：操作方法与【粘贴从站】一样，但是不同之处是新从站 PDO 中各映射对象的 PLC 内存地址会自动进行分配，无需用户修改。

2.3 K541 模块 ERR 灯含义

通常，CAN 控制器芯片自身就实现了完整的 CAN2.0 协议。根据 CAN2.0 协议中定义的错误检测机制，CAN 控制器芯片能自动检测到网络上的位错误、CRC 错误、ACK 错误等，并设置相应的错误寄存器供外部 MCU 读取。

K541 会实时读取所用 CAN 控制器芯片的错误寄存器值，若读取到错误值则会点亮 ERR 灯，若错误值为 0，则会熄灭 ERR 灯。因此，ERR 灯点亮就意味着网络通信质量比较差导致通信错误比较多，并非 K541 本身发生错误。当用户发现 ERR 灯点亮时，一般需要检查如下方面：

1） CAN 总线接线是否正确，是否有接错、虚接等情况。

2） 网络中所有节点采用的波特率是否一致。

3） 在总线型网络接法时，第 1 个节点和最后一个节点是否需要使用 120 欧姆的终端电阻。

4） 在网络或者节点附近，是否存在着强烈的干扰源。

5） PDO 中若存在频繁发送的对象（比如伺服的实时位置、实时速度等对象），那么最好为该 PDO 设置适当的“禁止时间”。

附录 G　更新系统程序

K 系列 PLC 提供用户自行更新系统程序功能，以便有用到后期开发新功能等需要的用户能及时更新到最新版本的系统程序。

在 KincoBulider 软件菜单栏-工具-更新系统程序（如下图）可找到更新系统程序的小工具，然后按照更新系统程序工具上的帮助菜单里边操作说明逐步进行操作即可