Logix5000 控制器通用指令参考手册

Allen-Bradley

目录号 1756 ControlLogix、1768 CompactLogix、
1769 CompactLogix、1789 SoftLogix、1794 FlexLogix、
带 DriveLogix 的 PowerFlex 700S

LISTEN.
THINK.
SOLVE.®

Allen-Bradley • Rockwell Software

Rockwell Automation

重要用户信息

固态设备的操作特性不同于机电设备的操作特性。《固态控制设备的应用、安装与维护安全指南》(出版号 SGI-1.1，本资料可从当地罗克韦尔自动化销售处索取或从 http://www.rockwellautomation.com/literature/ 网站下载) 介绍了固态设备与硬接线机电设备之间的一些重要差异。由于存在这些差异，且固态设备的应用又非常广泛，因此，但凡负责应用此设备的人员均须保证该设备的任何预期应用都是允许的。

任何情况下，对于因使用或操作本设备造成的任何间接或连带损失，罗克韦尔自动化公司概不负责。

本手册中包含的示例和图表仅用于说明。由于任何具体安装都涉及很多变数和很多不同要求，罗克韦尔自动化公司亦不对基于这些示例和图表执行的实际操作负责。

因使用本手册中所述的信息、电路、设备或软件而引起的相关专利问题，罗克韦尔自动化公司不负任何责任。

未经罗克韦尔自动化公司书面许可，不得复制本手册的全部或部分内容。

在本手册中，必要时我们将使用以下注释来提醒用户考虑相关的安全事宜。

警 告 	标识信息，用来标识在危险环境下可能发生爆炸，从而导致人员伤亡、物品损坏或经济损失的操作或情况。
重要事项	用来标识对成功应用和了解本产品有重要作用的信息。
注 意 	标识信息，用来标识可能导致人员伤亡、物品损坏或经济损失的操作或情况。注意符号可帮助您确定危险、避免危害并了解可能的后果。
电击危险 	位于设备 (例如，驱动器或电机) 表面或内部的标签，提醒相关人员可能存在危险电压。
烧伤危险 	位于设备 (例如，驱动器或电机) 表面或内部的标签，提醒相关人员表面可能存在高温危险。

变更摘要

本手册包含一些新信息以及更新的信息。本版本中的变更内容用变更条进行了标记，如本段右侧所示。

变更	页码
SSV/GSV 指令的 IgnoreArrayFaultsDuringPostScan 属性	187

注：

目录

第 12 章

循环 / 中断指令 **(FOR、 FOR...DO、 BRK、 EXIT、 RET)**

第 13 章

特殊指令 **(FBC、 DDT、 DTR、 PID)**

第 14 章

三角函数指令 **(SIN、 COS、 TAN、 ASN、 ASIN、 ACS、 ACOS、 ATN、 ATAN)**

注：

何处查找指令

使用此定位器可查找有关 Logix 指令的详细参考信息 (灰显指令可在其它手册中找到)。此定位器还列出了指令可使用的编程语言。

如果定位器列出	指令记录在
页码	本手册
坐标	《运动控制坐标系用户手册》 (出版号 MOTION-UM002)
运动控制	《Logix5000 控制器运动控制指令参考手册》 (出版号 MOTION-RM002)
PhaseManager	《PhaseManager 用户手册》 (出版号 LOGIX-UM001)
过程控制	《Logix5000 控制器过程控制和驱动指令参考手册》 (出版号 1756-RM006)

指令	位置	语言
ABL 缓冲区行的 ASCII 测试	19-617	梯形图 结构化文本
ABS 绝对值	7-285	梯形图 结构化文本 功能块
ACB 缓冲区中的 ASCII 字符	18-581	梯形图 结构化文本
ACL ASCII 清除缓冲区	18-583	梯形图 结构化文本
ACS 反余弦	15-540	梯形图 结构化文本 功能块
ADD 加	7-260	梯形图 结构化文本 功能块
AFI 恒假指令	12-464	梯形图
AHL ASCII 握手行	18-585	梯形图 结构化文本
ALM 报警	过程控制	结构化文本 功能块
ALMA 模拟报警	2-42	梯形图 结构化文本 功能块
ALMD 数字报警	2-30	梯形图 结构化文本 功能块
AND 按位与	8-311	梯形图 结构化文本 功能块
ARD ASCII 读取	18-589	梯形图 结构化文本
ARL ASCII 读取行	18-593	梯形图 结构化文本
ASN 反正弦	15-537	梯形图 结构化文本 功能块
ATN 反正切	15-543	梯形图 结构化文本 功能块
AVE 文件平均值	9-373	梯形图
AWA ASCII 写入附加	18-597	梯形图 结构化文本

指令	位置	语言
AWT ASCII 写入	18-602	梯形图 结构化文本
BAND 布尔与	8-325	结构化文本 功能块
BNOT 布尔非	8-334	结构化文本 功能块
BOR 布尔或	8-328	结构化文本 功能块
BPT 断点	21-635	梯形图
BRK 中断	13-481	梯形图
BSL 位左移	10-394	梯形图
BSR 位右移	10-398	梯形图
BTD 位域分配	8-302	梯形图
BTDT 带目标的位域分配	8-302	结构化文本 功能块
BTR 信息	5-146	梯形图 结构化文本
BTW 信息	5-146	梯形图 结构化文本
BXOR 布尔异或	8-331	结构化文本 功能块
CC 协调控制	过程控制	结构化文本 功能块
CLR 清零	8-302	梯形图 结构化文本
CMP 比较	6-214	梯形图
CONCAT 字符串串连	19-609	梯形图 结构化文本
COP 复制文件	9-363	梯形图 结构化文本
COS 余弦	15-531	梯形图 结构化文本 功能块
CPS 同步复制文件	9-363	梯形图 结构化文本
CPT 计算	7-256	梯形图

指令	位置	语言
CTD 减计数	4-134	梯形图
CTU 增计数	4-130	梯形图
CTUD 增 / 减计数	4-138	结构化文本 功能块
D2SD 离散 2 态设备	过程控制	结构化文本 功能块
D3SD 离散 3 态设备	过程控制	结构化文本 功能块
DDT 诊断检测	14-494	梯形图
DEDT 死区时间	过程控制	结构化文本 功能块
DEG 度数	17-558	梯形图 结构化文本 功能块
DELETE 字符串删除	19-611	梯形图 结构化文本
DERV 微分	过程控制	结构化文本 功能块
DFF D 触发器	过程控制	结构化文本 功能块
DIV 除	7-269	梯形图 结构化文本 功能块
DTOS DINT 转换为字符串	20-627	梯形图 结构化文本
DTR 数据转换	14-502	梯形图
EOT 传送终止	12-466	梯形图 结构化文本
EQU 等于	6-214	梯形图 结构化文本 功能块
ESEL 增强型选择	过程控制	结构化文本 功能块
EVENT 触发事件任务	12-472	梯形图 结构化文本
FAL 文件算术和逻辑	9-343	梯形图
FBC 文件位比较	14-486	梯形图

指令	位置	语言
FFL FIFO 装载	10-402	梯形图
FFU FIFO 卸载	10-408	梯形图
FGEN 函数发生器	过程控制	结构化文本 功能块
FIND 查找字符串	19-613	梯形图 结构化文本
FLL 文件填充	9-369	梯形图
FOR 循环	13-478	梯形图
FRD 转换为整数	17-567	梯形图 功能块
FSC 文件搜寻和比较	9-354	梯形图
GEQ 大于等于	6-223	梯形图 结构化文本 功能块
GRT 大于	6-227	梯形图 结构化文本 功能块
GSV 获取系统值	5-182	梯形图 结构化文本
HLL 上限 / 下限	过程控制	结构化文本 功能块
HPF 高通滤波器	过程控制	结构化文本 功能块
ICON 输入接线器	A-649	功能块
IMC 内部模型控制	过程控制	结构化文本 功能块
INSERT 插入字符串	19-615	梯形图 结构化文本
INTG 积分器	过程控制	结构化文本 功能块
IOT 立即输出	5-209	梯形图 结构化文本
IREF 输入参数	A-649	功能块
JKFF JK 触发器	过程控制	结构化文本 功能块
JMP 跳转至标签	12-442	梯形图

指令	位置	语言
JSR 跳转至子例程	12-444	梯形图 结构化文本 功能块
JXR 跳转至外部例程	12-455	梯形图
LBL 标签	12-442	梯形图
LDL2 二阶超前滞后	过程控制	结构化文本 功能块
LDLG 超前 - 滞后	过程控制	结构化文本 功能块
LEQ 小于等于	6-231	梯形图 结构化文本 功能块
LES 小于	6-235	梯形图 结构化文本 功能块
LFL LIFO 装载	10-414	梯形图
LFU LIFO 卸载	10-420	梯形图
LIM 限制	6-239	梯形图 功能块
LN 自然对数	16-548	梯形图 结构化文本 功能块
LOG 以 10 为底的对数	16-551	梯形图 结构化文本 功能块
LOWER 小写	20-633	梯形图 结构化文本
LPF 低通滤波器	过程控制	结构化文本 功能块
MAAT 运动应用轴调整	运动控制	梯形图 结构化文本
MAFR 运动轴故障复位	运动控制	梯形图 结构化文本
MAG 运动轴齿轮	运动控制	梯形图 结构化文本
MAHD 运动应用连接诊断	运动控制	梯形图 结构化文本
MAH 运动轴归零	运动控制	梯形图 结构化文本
MAJ 运动轴点动	运动控制	梯形图 结构化文本
MAM 运动轴移动	运动控制	梯形图 结构化文本
MAOC 运动臂输出凸轮	运动控制	梯形图 结构化文本
MAPC 运动轴位置凸轮	运动控制	梯形图 结构化文本
MAR 运动臂定位	运动控制	梯形图 结构化文本
MASD 运动轴关闭	运动控制	梯形图 结构化文本
MAS 运动轴停止	运动控制	梯形图 结构化文本
MASR 运动轴关闭复位	运动控制	梯形图 结构化文本
MATC 运动轴时间凸轮	运动控制	梯形图 结构化文本
MAVE 移动平均值	过程控制	结构化文本 功能块
MAW 运动臂观察	运动控制	梯形图 结构化文本
MAXC 最大值捕捉	过程控制	结构化文本 功能块
MCCD 运动坐标变化动态	坐标	梯形图 结构化文本
MCCM 运动坐标环形移动	坐标	梯形图 结构化文本
MCCP 运动计算凸轮轮廓	运动控制	梯形图 结构化文本
MCD 运动变化动态	运动控制	梯形图 结构化文本
MCLM 运动坐标线性移动	坐标	梯形图 结构化文本
MCR 主控复位	12-460	梯形图
MCSD 运动坐标关闭	坐标	梯形图 结构化文本
MCS 运动坐标停止	坐标	梯形图 结构化文本
MCSR 运动坐标关闭复位	坐标	梯形图 结构化文本

指令	位置	语言
MCT 运动坐标转换	坐标	梯形图 结构化文本
MCTP 运动计算转换位置	坐标	梯形图 结构化文本
MDF 运动直接驱动关闭	运动控制	梯形图 结构化文本
MDOC 运动解除输出凸轮	运动控制	梯形图 结构化文本
MDO 运动直接驱动开启	运动控制	梯形图 结构化文本
MDR 运动解除定位	运动控制	梯形图 结构化文本
MDW 运动解除观察	运动控制	梯形图 结构化文本
MEQ 屏蔽码等于	6-245	梯形图 结构化文本 功能块
MGSD 运动组关闭	运动控制	梯形图 结构化文本
MGS 运动组停止	运动控制	梯形图 结构化文本
MGSP 运动组选通位置	运动控制	梯形图 结构化文本
MGSR 运动组关闭复位	运动控制	梯形图 结构化文本
MID 中间字符串	19-617	梯形图 结构化文本
MINC 最小值捕捉	过程控制	结构化文本 功能块
MMC 模块多变量控制	过程控制	结构化文本 功能块
MOD 模	7-274	梯形图 结构化文本 功能块
MOV 移动	8-291	梯形图
MRAT 运动运行轴调整	运动控制	梯形图 结构化文本
MRHD 运动运行轴连接诊断	运动控制	梯形图 结构化文本
MRP 运动重定义位置	运动控制	梯形图 结构化文本
MSF 运动伺服关闭	运动控制	梯形图 结构化文本

指令	位置	语言
MSG 信息	5-146	梯形图 结构化文本
MSO 运动伺服开启	运动控制	梯形图 结构化文本
MSTD 移动标准偏差	过程控制	结构化文本 功能块
MUL 乘	7-266	梯形图 结构化文本 功能块
MUX 多路选择器	过程控制	功能块
MVM 屏蔽移动	8-293	梯形图
MVMT 带目标屏蔽移动	8-296	结构化文本 功能块
NEG 取反	7-282	梯形图 结构化文本 功能块
NEQ 不等于	6-250	梯形图 结构化文本 功能块
NOP 空操作	12-465	梯形图
NOT 按位非	8-322	梯形图 结构化文本 功能块
NTCH 陷波滤波器	过程控制	结构化文本 功能块
OCON 输出接线器	A-649	功能块
ONS 单脉冲触发	3-90	梯形图
OR 按位或	8-314	梯形图 结构化文本 功能块
OREF 输出参数	A-649	功能块
OSFI 带输入的下降沿单脉冲触发	3-101	结构化文本 功能块
OSF 下降沿单脉冲触发	3-96	梯形图
OSRI 带输入的上升沿单脉冲触发	3-93	结构化文本 功能块

指令	位置	语言
OSR 上升沿单脉冲触发	3-93	梯形图
OTE 输出激励	3-84	梯形图
OTL 输出锁存	3-86	梯形图
OTU 输出解锁	3-88	梯形图
PATT 连接到设备阶段	PhaseManager	梯形图 结构化文本
PCLF 设备阶段清零失败	PhaseManager	梯形图 结构化文本
PCMD 设备阶段命令	PhaseManager	梯形图 结构化文本
PDET 从设备阶段断开	PhaseManager	梯形图 结构化文本
PFL 设备阶段失败	PhaseManager	梯形图 结构化文本
PIDE 增强型 PID	过程控制	结构化文本 功能块
PID 比例积分微分	14-505	梯形图 结构化文本
PI 比例 + 积分	过程控制	结构化文本 功能块
PMUL 脉冲乘法器	过程控制	结构化文本 功能块
POSP 位置比例	过程控制	结构化文本 功能块
POVR 设备阶段覆盖命令	PhaseManager	梯形图 结构化文本
PPD 设备阶段已暂停	PhaseManager	梯形图 结构化文本
PRNP 设备阶段新参数	PhaseManager	梯形图 结构化文本
PSC 阶段状态完成	PhaseManager	梯形图 结构化文本
PXRQ 设备阶段外部请求	PhaseManager	梯形图 结构化文本
RAD 弧度	17-561	梯形图 结构化文本 功能块
RESD 优先复位	过程控制	结构化文本 功能块

指令	位置	语言
RES 复位	4-143	梯形图
RET 返回	12-444 和 13-482	梯形图 结构化文本 功能块
RLIM 速率限制器	过程控制	结构化文本 功能块
RMPS 斜坡 / 保持	过程控制	结构化文本 功能块
RTO 保持型接通计时器	4-114	梯形图
RTOR 带复位的保持型接通计时器	4-126	结构化文本 功能块
RTOS REAL 转换为字符串	20-629	梯形图 结构化文本
SBR 子例程	12-444	梯形图 结构化文本 功能块
SCL 定标	过程控制	结构化文本 功能块
SCRV S 曲线	过程控制	结构化文本 功能块
SEL 选择	过程控制	功能块
SETD 优先置位	过程控制	结构化文本 功能块
SFP SFC 暂停	12-468	梯形图 结构化文本
SFR SFC 复位	12-470	梯形图 结构化文本
SIN 正弦	15-528	梯形图 结构化文本 功能块
SIZE 以元素计的尺寸	9-389	梯形图 结构化文本
SNEG 选择取反	过程控制	结构化文本 功能块
SOC 二阶控制器	过程控制	结构化文本 功能块
SQI 顺序器输入	11-428	梯形图
SQL 顺序器装载	11-436	梯形图

指令	位置	语言
SQO 顺序器输出	11-432	梯形图
SQR 平方根	7-278	梯形图 功能块
SQRT 平方根	7-278	结构化文本
SRT 文件排序	9-378	梯形图 结构化文本
SRTP 分程时间比例	过程控制	结构化文本 功能块
SSUM 选择加法器	过程控制	结构化文本 功能块
SSV 设置系统值	5-182	梯形图 结构化文本
STD 文件标准偏差	9-383	梯形图
STOD 字符串转换为 DINT	20-621	梯形图 结构化文本
STOR 字符串转换为 REAL	20-624	梯形图 结构化文本
SUB 减	7-263	梯形图 结构化文本 功能块
SWPB 交换字节	8-307	梯形图 结构化文本
TAN 正切	15-534	梯形图 结构化文本 功能块
TND 临时结束	12-458	梯形图
TOD 转换为 BCD	17-564	梯形图 功能块
TOFR 带复位的关断延时计时器	4-122	结构化文本 功能块
TOF 关断延时计时器	4-110	梯形图
TONR 带复位的接通延时计时器	4-118	结构化文本 功能块
TON 接通延时计时器	4-106	梯形图
TOT 累加器	过程控制	结构化文本 功能块
TPT 追踪点	21-639	梯形图
TRN 截断	17-569	梯形图 功能块
TRUNC 截断	17-569	结构化文本
UID 禁止用户中断	12-462	梯形图 结构化文本
UIE 允许用户中断	12-462	梯形图 结构化文本
UPDN 增 / 减累加器	过程控制	结构化文本 功能块
UPPER 大写	20-631	梯形图 结构化文本
XIC 检查是否闭合	3-78	梯形图
XIO 检查是否断开	3-81	梯形图
XOR 按位异或	8-318	梯形图 结构化文本 功能块
XPY X 的 Y 次幂	16-554	梯形图 结构化文本 功能块

注：

简介

本手册为程序员提供有关 Logix 控制器每条指令的详细信息。您应当熟悉 Logix 控制器存储和处理数据的方法。

初级程序员在使用指令前应阅读与该指令相关的所有详细信息。经验丰富的程序员可参考指令信息来进行确认。

本手册是介绍 Logix5000 控制器编程与操作的通用步骤的一系列相关手册之一。有关通用步骤手册的完整列表，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。

Logix5000 控制器指基于 Logix5000 操作系统的控制器，例如：

- CompactLogix 控制器。
- ControlLogix 控制器。
- DriveLogix 控制器。
- FlexLogix 控制器。
- SoftLogix5800 控制器。

指令格式说明

部分	信息
指令名称	识别指令。 确定指令是输入指令还是输出指令。
操作数	列出指令的所有操作数。  如果可用于梯形图，则介绍操作数。  如果可用于结构化文本，则介绍操作数。  如果可用于功能块，则介绍操作数。 显示在默认功能块上的引脚仅为默认引脚。操作数表列出所有可能的功能块引脚。
指令结构	列出指令的控制状态位和值 (如果有)。
说明	说明指令的使用。 说明使能和不使能指令时的区别 (如果合适)。
算术状态标志	说明指令是否影响算术状态标志。

指令格式说明

部分	信息
故障条件	说明指令是否产生主要故障或次要故障。如果产生，定义故障类型和代码。
执行	说明指令运行方式的详情。
示例	提供每种编程语言的至少一个编程示例。 包括对每个示例加以解释的说明。

以下图标帮助识别语言特定信息。

图标	编程语言
	梯形图
	结构化文本
	功能块

所有指令的通用信息

Logix5000 指令集有一些通用属性。

信息	附录
通用属性	通用属性定义： • 算术状态标志 • 数据类型 • 关键字
功能块属性	功能块属性定义： • 程序和操作员控制 • 计时模式

惯例和相关术语

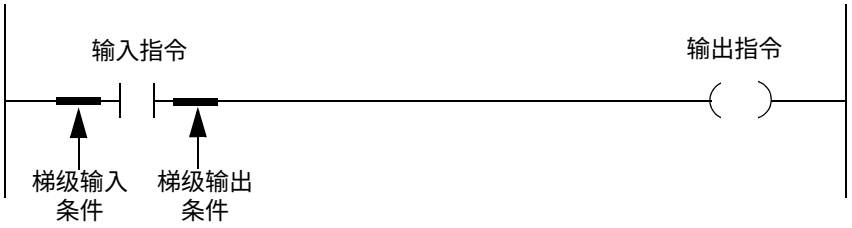
本手册使用置位和清零定义位的状态 (布尔型) 和值 (非布尔型)。

术语	含义
置位	该位置 1 (ON)。 值被置为任意非零数。
清零	该位清 0 (OFF)。 值的所有位均清 0。

如果操作数或参数支持多种数据类型，粗体数据类型表示最佳数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

梯形图梯级条件

控制器根据指令前的梯级条件 (梯级输入条件) 评估梯形图指令。根据梯级输入条件和指令，控制器将指令后的梯级条件 (梯级输出条件) 置位，进而影响后续指令。



如果输入指令的梯级输入条件为真，控制器评估指令并根据指令结果将梯级输出条件置位。如果指令评估为真，则梯级输出条件为真；如果指令评估为假，则梯级输出条件为假。

控制器还会预扫描指令。预扫描是对控制器中所有例程的特殊扫描。控制器在预扫描期间扫描所有主例程和子例程，但忽略可跳过指令执行的跳转。控制器执行所有 FOR 循环和子例程调用。如果子例程被多次调用，则每次调用时该子例程都会执行。控制器使用梯形图指令预扫描来复位非保持 I/O 和内部值。

预扫描期间，输入值不是当前值，输出值不写入。以下条件会发生预扫描：

- 从程序模式切换到运行模式。
- 开机时自动进入运行模式。

以下情况下，程序不进行预扫描：

- 在控制器运行期间程序转变为预定状态。
- 在控制器进入运行模式时程序处在未预定状态。

功能块状态

重要事项

在功能块中编程时，将工程单位的范围限制为 +/-$10^{+/-15}$，这是因为内部浮点计算是通过使用单精度浮点来完成的。如果结果接近单精度浮点 (+/-$10^{+/-38}$) 的限度，此范围以外的工程单位可能造成精度降低。

控制器根据不同情况的状态评估功能块指令。 .

可能条件	说明
预扫描	功能块例程预扫描与梯形图例程预扫描相同。惟一的区别在于，预扫描期间每条功能块指令的 EnableIn 参数都清零。
指令第一次扫描	指令第一次扫描是指指令在预扫描后第一次执行。控制器使用指令第一次扫描读取当前输入和确定将要处于的合适状态。
指令第一次运行	指令第一次运行是指指令第一次与新的数据结构实例共同执行。控制器使用指令第一次运行生成系数，以及在初次下载后对于功能块来说不发生变化的其它数据存储。

每条功能块指令还包括 EnableIn 和 EnableOut 参数。

- EnableIn 置位后，功能块指令正常执行。
- EnableIn 清零后，功能块指令执行预扫描逻辑、后扫描逻辑或跳过正常的算法执行。
- EnableOut 反映出 EnableIn，但如果执行功能块时检测到溢出条件， EnableOut 也会清零。
- EnableIn 从清零切换到置位时，功能块从先前停止的位置继续执行。但在 EnableIn 从清零切换到置位时，也有一些功能块指令会指定特殊的功能，如重新初始化。对于具有时基参数的功能块指令， EnableIn 从清零切换到置位时，只要计时模式为密集采样，指令便会总是从先前停止的位置继续执行。

如果未连接 EnableIn 参数，则指令始终正常执行， EnableIn 保持置位。如果将 EnableIn 清零，它在指令下次执行时变为置位。

注：

FactoryTalk 报警和事件 基于 Logix 的指令 (ALMD、 ALMA)

简介

这些基于 Logix 的报警指令在梯形图、结构文本和功能块图中可用。与 FactoryTalk View SE 软件 (V5.0 和更高版本) 配合使用时，这些指令借助可视化软件包创建系统。控制器检测到报警条件并将事件发布到 FactoryTalk View 报警和事件服务器，服务器将报警传送到预订接收通知的 Factory Talk View SE 客户端。

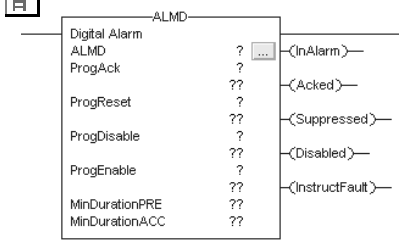
如果要	使用以下指令	在以下语言中可用	页码
根据布尔 (真 / 假) 条件检测报警	ALMD	梯形图 结构化文本 功能块	30
根据值的变化率或变化水平检测报警	ALMA	梯形图 结构化文本 功能块	42

数字报警 (ALMD)

ALMD 指令根据布尔 (真 / 假) 条件检测报警。程序 (Prog) 和操作员 (Oper) 控制参数为报警命令提供接口。

操作数：

梯形图



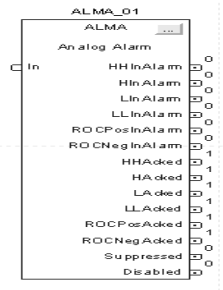
在梯形图中，报警条件输入 (In) 从梯级条件获取。

操作数	类型	格式	说明
ALMD 标签	ALARM_DIGITAL	结构	ALMD 结构。
In	BOOL	标签 立即数	仅限结构化文本。 指令执行时数值复制到 In。对报警输入值进行对比，确定是否存在报警。
ProgAck	BOOL	标签 立即数	指令执行时数值复制到 ProgAck。从清零状态转变为置位状态时，确认报警 (如果需要确认)。
ProgReset	BOOL	标签 立即数	指令执行时数值复制到 ProgReset。从清零状态转变为置位状态时，复位报警 (如果需要)。
ProgDisable	BOOL	标签 立即数	指令执行时数值复制到 ProgDisable。置位时，禁止报警 (不会覆盖使能命令)。
ProgEnable	BOOL	标签 立即数	指令执行时数值复制到 ProgEnable。置位时，使能报警 (优先级高于禁止命令)。
MinDurationPRE	DINT	立即数	仅限梯形图。 指定满足报警条件达多长时间后报告报警 (毫秒)。
MinDurationACC	DINT	立即数	仅限梯形图。 指示从满足报警条件以来所经过的毫秒数。

结构化文本

```
ALMD(ALMD, In, ProgAck,
ProgReset, ProgDisable,
ProgEnable);
```

操作数与梯形图 ALMD 指令的相同，但存在一些差异，具体情况如上所述。



功能块

操作数	类型	格式	说明
ALMD 标签	ALARM_DIGITAL	结构	ALMD 结构

ALARM_DIGITAL 结构

输入参数	数据类型	说明
EnableIn	BOOL	梯形图 与梯级状态相对应。不影响处理。 功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。始终执行该指令。
In	BOOL	指令的数字信号输入。 默认为清零状态。 梯形图 遵从梯级条件。如果梯级条件为真，则置位。如果梯级条件为假，则清零。 结构化文本 从指令操作数复制。
InFault	BOOL	指示输入状况不良的情况。用户应用项目可以置位 InFault 来指示输出信号有错误。置位时，指令将置位 InFaulted (Status.1)。清零时，指令将清零 InFaulted (Status.1)。无论在哪种情况下，指令都继续评估 In 是否满足报警条件。 默认为清零状态 (状况良好)。
Condition	BOOL	指定激活报警的条件。如果 Condition 置位，则在 In 置位时激活报警条件。如果 Condition 清零，则在 In 清零时激活报警条件。 默认为置位状态。

输入参数	数据类型	说明
AckRequired	BOOL	指定是否需要确认报警。置位时，需要确认。清零时，无需确认， Acked 始终置位。 默认为置位状态。
Latched	BOOL	指定是否锁定报警。锁定的报警在报警条件变为假时仍保持为 InAlarm，直到收到复位命令。置位时，锁定报警。清零时，不锁定报警。 仅当报警条件为假时，才能复位锁定的报警。 默认为清零状态。
ProgAck	BOOL	通过用户程序置位，进而确认报警。如果报警未确认，则需要从清零状态转变为置位状态。 默认为清零状态。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
OperAck	BOOL	通过操作员界面置位，进而确认报警。如果报警未确认，则需要从清零状态转变为置位状态。该指令将清零此参数。 默认为清零状态。
ProgReset	BOOL	通过用户程序置位，进而复位报警。如果报警为 InAlarm 且 In 条件不是处于报警状态，则需要从清零状态转变为置位状态。 默认为清零状态。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
OperReset	BOOL	通过操作员界面置位，进而复位报警。如果报警为 InAlarm 且 In 条件不是处于报警状态，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
ProgSuppress	BOOL	通过用户程序置位，进而抑制报警。 默认为清零状态。
OperSuppress	BOOL	通过操作员界面置位，进而抑制报警。该报警指令将清零此参数。 默认为清零状态。
ProgUnsuppress	BOOL	通过用户程序置位，进而取消抑制报警。其优先级高于抑制命令。 默认为清零状态。

输入参数	数据类型	说明
OperUnsuppress	BOOL	通过操作员界面置位，进而取消抑制报警。其优先级高于抑制命令。该报警指令将清零此参数。 默认为清零状态。
ProgDisable	BOOL	通过用户程序置位，进而禁止报警。 默认为清零状态。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
OperDisable	BOOL	通过操作员界面置位，进而禁止报警。该报警指令将清零此参数。 默认为清零状态。
ProgEnable	BOOL	通过用户程序置位，进而使能报警。其优先级高于禁止命令。 默认为清零状态。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
OperEnable	BOOL	通过操作员界面置位，进而使能报警。其优先级高于禁止命令。该报警指令将清零此参数。 默认为清零状态。
AlarmCountReset	BOOL	通过用户程序置位，进而复位报警计数。从清零状态转变为置位状态会将报警计数复位为零。 默认为清零状态。
UseProgTime	BOOL	指定是使用控制器时钟还是使用 ProgTime 值来给报警状态变化事件提供时间戳。置位时， ProgTime 值提供时间戳。清零时，控制器时钟提供时间戳。 默认为清零状态。

输入参数	数据类型	说明
ProgTime	LINT	如果 UseProgTime 置位，则此值将用来为所有事件提供时间戳值。这样，应用项目将应用从报警源 (例如，事件顺序输入模块) 获取的时间戳。
Severity	DINT	报警的严重程度。它不会影响控制器处理报警，但可在报警订阅者上用于排序和过滤功能。 有效值 = 1...1000(1000 表示严重程度最高； 1 表示严重程度最低)。 默认值 = 500。
MinDurationPRE	DINT	报警条件保持为真的最小持续时间预设值 (毫秒)，在此时间过后报警将标记为 InAlarm 并且将向客户端发送报警通知。控制器在检测到报警条件后立即收集报警数据，所以在等待满足最小持续时间时不会丢失任何数据。 有效值 = 0...2,147,483,647。 默认值 = 0。

输出参数	数据类型	说明
EnableOut	BOOL	使能输出。
InAlarm	BOOL	报警激活状态。报警激活时置位。报警未激活时 (正常状态) 清零。
Acked	BOOL	报警确认状态。报警已确认时置位。报警未确认时清零。 AckRequired 清零时， Acked 始终置位。
InAlarmUnack	BOOL	组合式报警激活和确认状态。报警激活 (InAlarm 置位) 且未确认 (Acked 清零) 时置位。报警处于正常 (未激活) 状态、已确认或这两种情况时清零。
Suppressed	BOOL	报警的受抑制状态。报警受抑制时置位。报警未受抑制时清零。
Disabled	BOOL	报警的禁止状态。报警禁止时置位。报警启用时清零。
MinDurationACC	DINT	从检测到报警以来所经历的时间。当此值达到 MinDurationPRE 时，报警将激活 (InAlarm 置位) 并向客户端发送通知。
AlarmCount	DINT	报警激活 (InAlarm 置位) 的次数。如果达到最大值，则计数器将保持该值为最大计数值。
InAlarmTime	LINT	报警检测的时间戳。
AckTime	LINT	报警确认的时间戳。如果报警无需确认，则此时间戳等于报警时间。
RetToNormalTime	LINT	报警返回到正常状态的时间戳。
AlarmCountReset Time	LINT	指示报警计数复位时间的时间戳。
DeliveryER	BOOL	报警通知信息传输错误。存在传输错误时置位，传输错误可能是没有预定任何报警订阅者，或者至少一个订阅者没有收到最新的报警变化状态信息。传输成功或正在传输时清零。

输出参数	数据类型	说明
DeliveryDN	BOOL	报警通知信息传输完成。传输成功时置位，传输成功是指至少预定一个订阅者且所有订阅者均成功接收最新的报警变化状态信息。传输未成功完成或正在传输时清零。
DeliveryEN	BOOL	报警通知信息的传输状态。正在传输时置位。未进行传输时清零。
NoSubscriber	BOOL	尝试传输最新信息时报警没有任何订阅者。没有任何订阅者时置位。至少有一个订阅者时清零。
NoConnection	BOOL	尝试传输最新信息时未连接报警订阅者。与所有订阅者的连接断开时置位。至少连接一个订阅者或没有任何订阅者时清零。
CommError	BOOL	传输报警信息时的通信错误。存在通信错误且所有重试次数均已用完时置位。也就是说，预定了订阅者且其具有连接，但控制器未收到确认传输信息的信息。所有已连接的订阅者都确认接收报警信息时清零。
AlarmBuffered	BOOL	由于存在通信错误 (CommError 置位) 或连接丢失 (NoConnection 置位) 而缓冲报警信息。至少为一个订阅者缓冲报警信息时置位。非缓冲报警信息时清零。
Subscribers	DINT	此报警的订阅者数。
SubscNotified	DINT	成功接到最新的报警状态变化通知的订阅者的数量。
Status	DINT	组合状态指示器： Status.0 = InstructFault。 Status.1 = InFaulted。 Status.2 = SeverityInv。
InstructFault (Status.0)	BOOL	存在指令错误情况。这不是次要或主要控制器错误。检查其余状态位，确定发生的情况。
InFaulted (Status.1)	BOOL	用户程序已置位 InFault，这说明输入数据质量不良。报警将继续评估 In 是否满足报警条件。
SeverityInv (Status.2)	BOOL	报警严重程度组态无效。 如果严重程度 < 1，则指令使用 Severity = 1。 如果严重程度 > 1000，则指令使用 Severity = 1000。

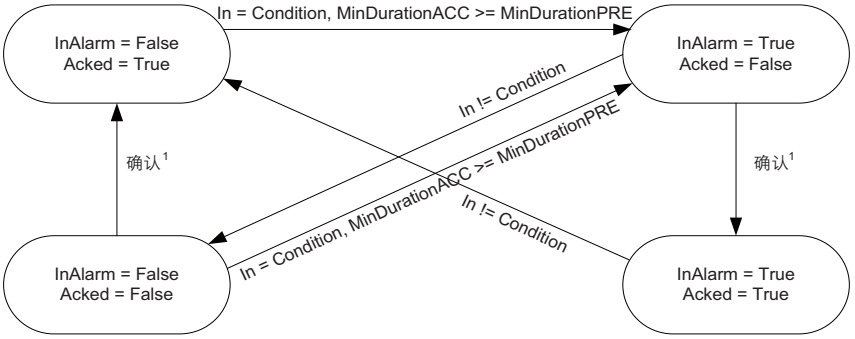
说明　ALMD 指令根据布尔 (真 / 假) 条件检测报警。

当与 RSLinx Enterprise 和 FactoryTalk View SE 软件配合使用时，ALMD 指令可提供附加功能。可在 FactoryTalk View SE 软件中的“ 报警摘要 ”(Alarm Summary)、“ 报警显示条 ”(Alarm Banner)、“ 报警状态浏览器 ”(Alarm Status Explorer) 和 “ 报警记录查看器 ”(Alarm Log Viewer) 显示画面中显示报警。

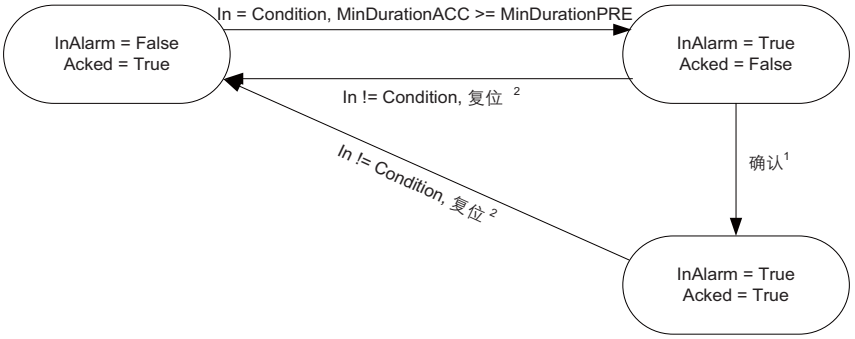
RSLinx Enterprise 软件订阅控制器中的报警。使用输出参数可监视指令，以查看报警订阅状态和显示报警状态变化。如果与 RSLinx Enterprise 软件的连接中断，控制器可暂时缓冲报警数据直到恢复连接。

需要确认时的状态图

Latched = False



Latched = True

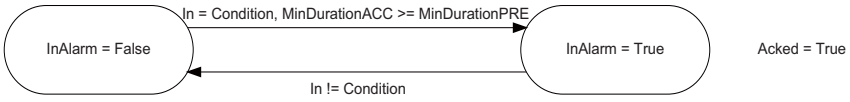


[1] 可以通过多种不同的方式来确认报警：ProgAck、OperAck、客户机(RSLogix 5000 软件、RSView 软件)。

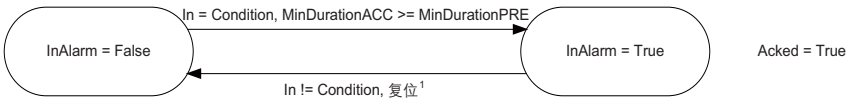
[2] 可以通过多种不同的方式来复位报警ProgReset、OperReset、客户机(RSLogix 5000 软件、RSView 软件)。

不需要确认时的状态图

Latched = False



Latched = True



[1] 可以通过多种不同的方式来复位报警：ProgReset、OperReset、客户机(RSLogix 5000 软件、RSView 软件)

算术状态标志： 无

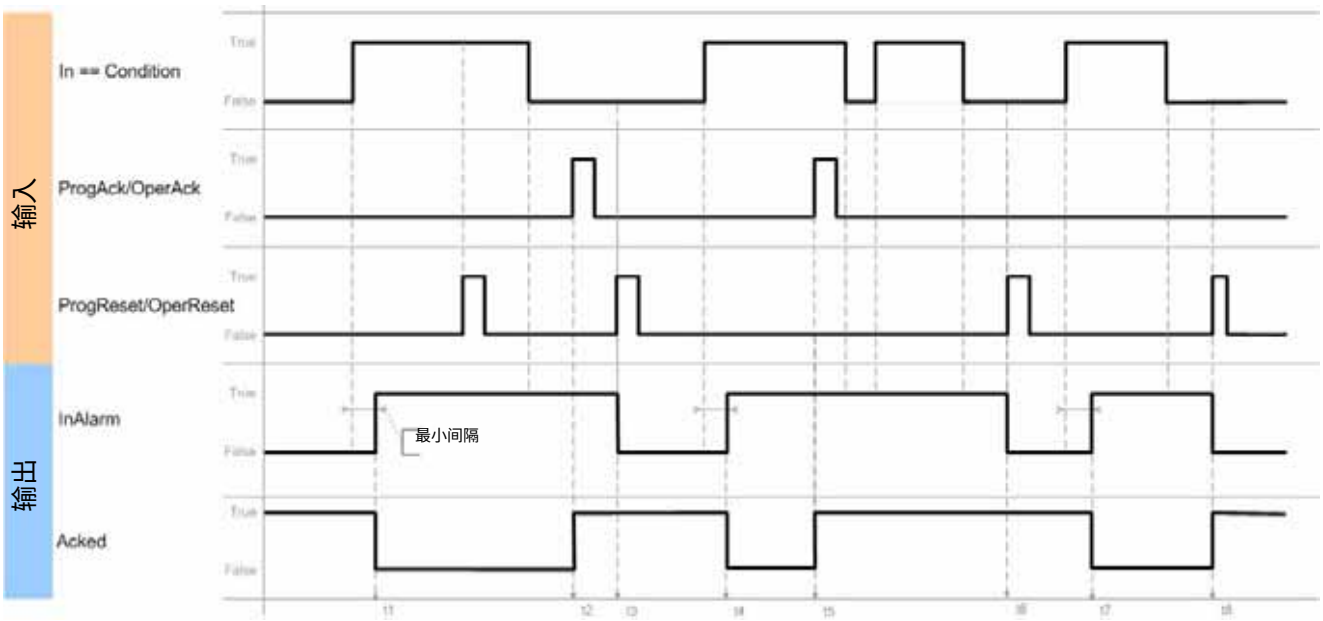
故障条件： 无

执行：

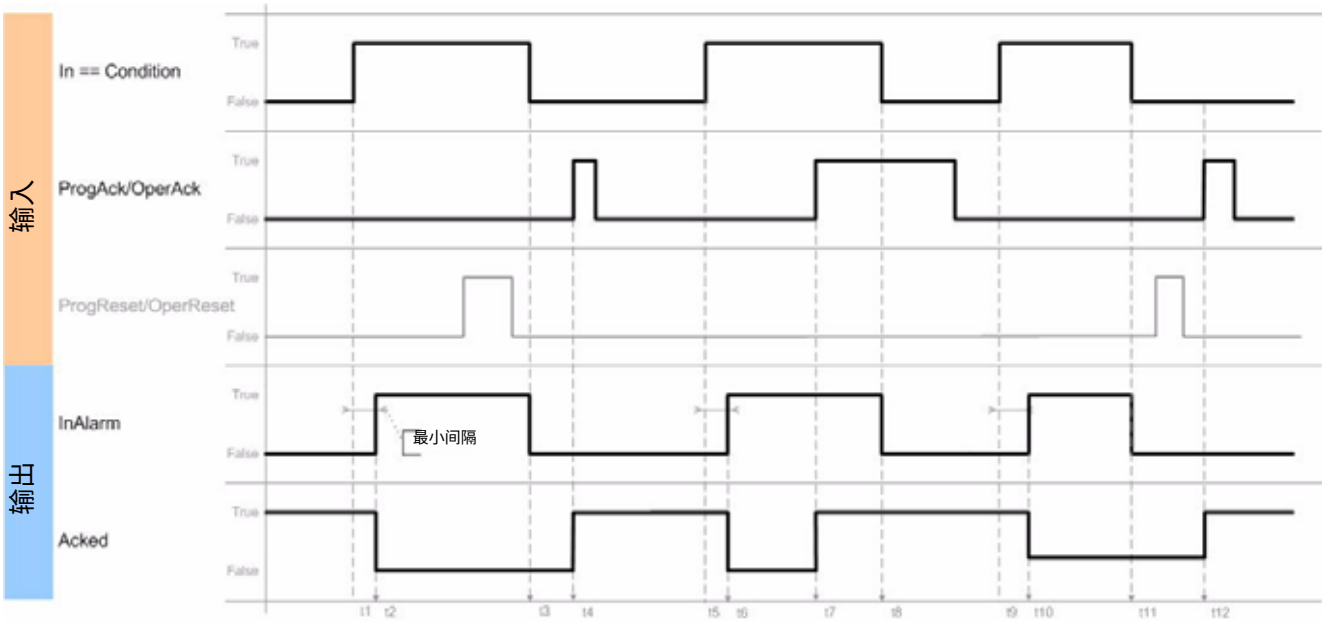
条件	梯形图操作
预扫描	梯级输出条件设置为假。 清零 InAlarm 并置位 Acked。 清除所有操作员请求、时间戳和传输标记。
梯级输入条件为假	梯级输出条件设置为假。 清零 EnableIn 和 EnableOut。 清零 In 参数，指令通过评估确定报警状态。
梯级输入条件为真	梯级输出条件设置为真。 置位 EnableIn 和 EnableOut。 置位 In 参数，指令通过评估确定报警状态。
后扫描	梯级输出条件设置为假。

条件	功能块操作	结构化文本操作
预扫描	清除所有操作员请求、时间戳和传输标记。 清零 InAlarm 并置位 Acked。	清除所有操作员请求、时间戳和传输标记。 清零 InAlarm 并置位 Acked。
指令第一次扫描	不执行任何操作。	不执行任何操作。
指令第一次运行	不执行任何操作。	不执行任何操作。
EnableIn 处于清零状态	不执行该指令。 清零 EnableOut。	执行该指令。 EnableOut 始终置位。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。	执行该指令。 EnableOut 始终置位。
后扫描	不执行任何操作。	不执行任何操作。

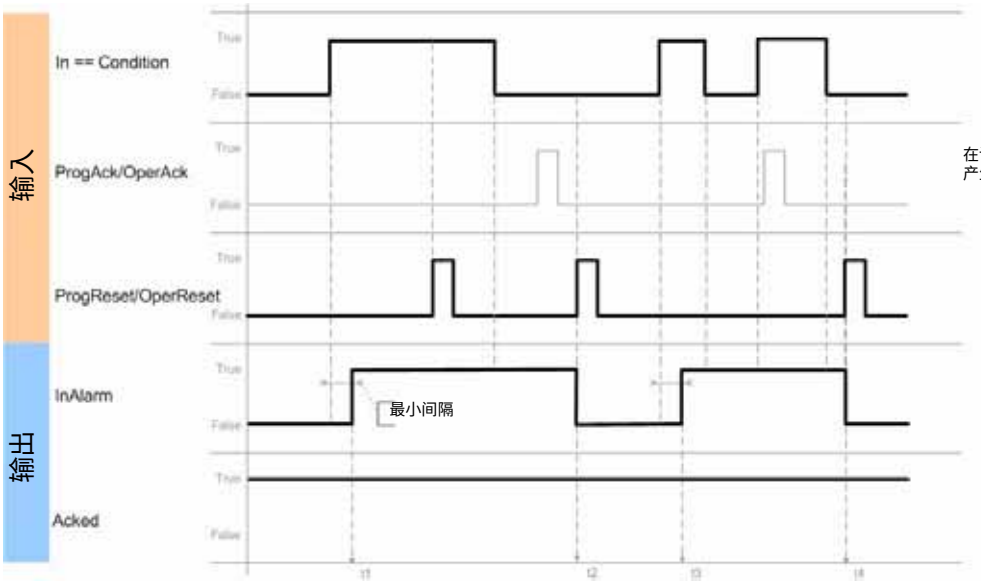
ALMD 报警需要确认且已锁定



ALMD 报警需要确认且未锁定

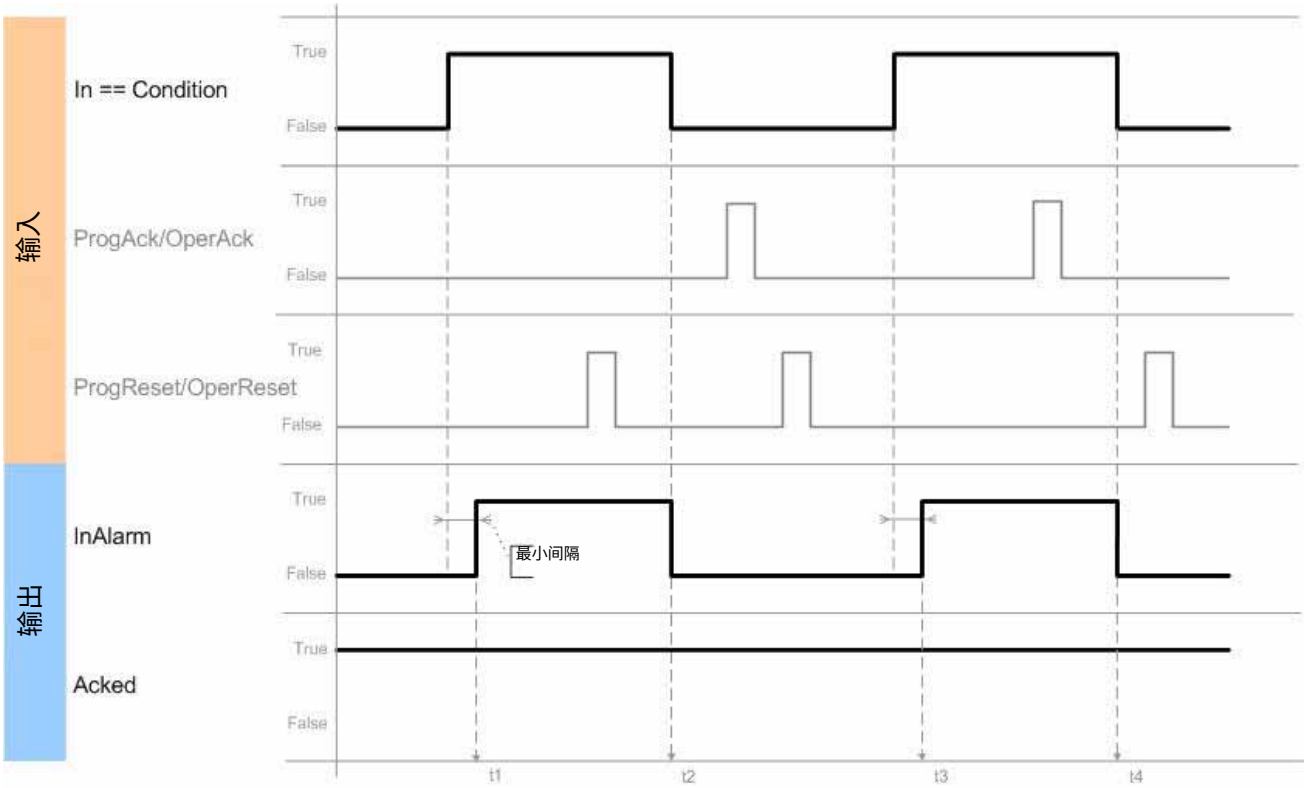


ALMD 报警无需确认且已锁定



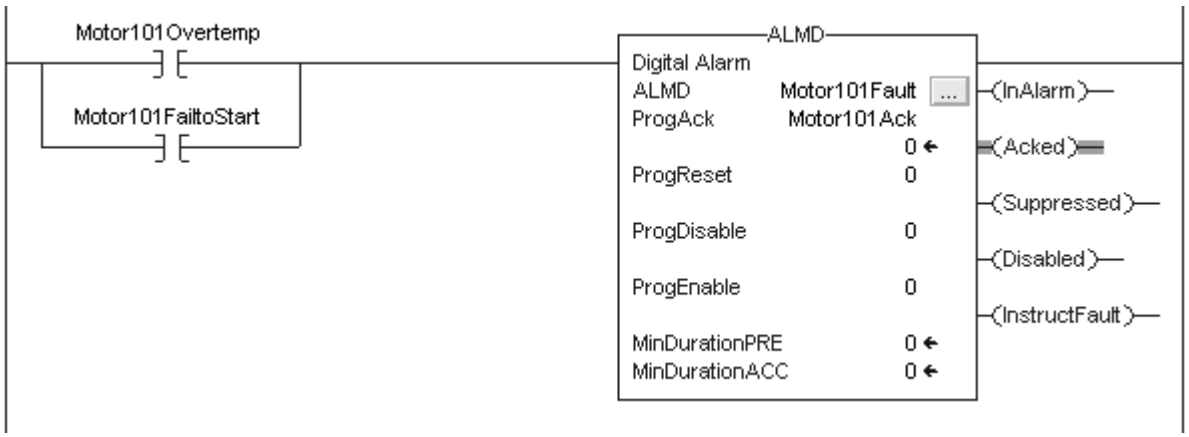
在该组态情况下，此信号不会对报警状态变化产生任何影响。

ALMD 报警无需确认且未锁定



示例： 两个电机故障信号组合为只要出现一个报警信号，就激活电机故障报警。通过程序将 Motor101Ack 标签值从清零状态转变为置位状态来确认报警。应用程序逻辑必须清零 Motor101Ack。

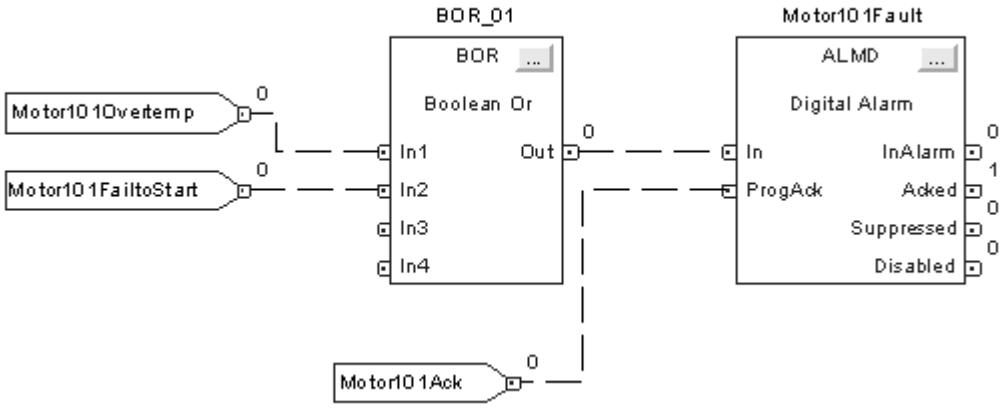
梯形图



结构化文本

```
Motor101FaultConditions := Motor101Overtemp OR
Motor101FailToStart;

ALMD(Motor101Fault,Motor101FaultConditions,
Motor101Ack,0,0,0 );
```

功能块

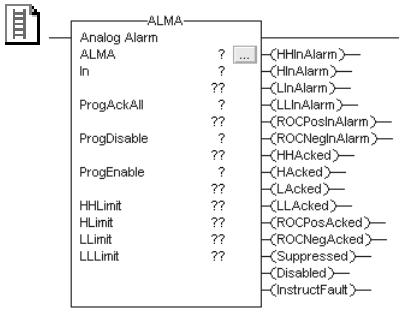


模拟报警 (ALMA)

ALMA 指令根据模拟值的变化率或变化水平来检测报警。程序 (Prog) 和操作员 (Oper) 控制参数为报警命令提供接口。

操作数：

梯形图



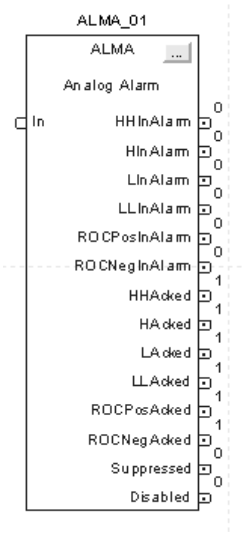
操作数	类型	格式	说明
ALMA 标签	ALARM_ANALOG	结构	ALMA 结构。
In	REAL DINT INT SINT	标签 立即数	指令执行时数值复制到 In。报警输入值，其与报警限值比较以检测报警条件。
ProgAckAll	BOOL	标签 立即数	指令执行时数值复制到 ProgAckAll。从清零状态转变为置位状态时，确认所有需要确认的报警条件。
ProgDisable	BOOL	标签 立即数	指令执行时数值复制到 ProgDisable。置位时，禁止报警 (不会覆盖使能命令)。
ProgEnable	BOOL	标签 立即数	指令执行时数值复制到 ProgEnable。置位时，使能报警 (优先级高于禁止命令)。
HHlimit	REAL	立即数	仅限梯形图。 报警上上限。
HLimit	REAL	立即数	仅限梯形图。 报警上限。
LLimit	REAL	立即数	仅限梯形图。 报警下限。
LLLimit	REAL	立即数	仅限梯形图。 报警下下限。

结构化文本

```
ALMA(ALMA, In, ProgAckAll,
ProgDisable, ProgEnable);
```

操作数与梯形图 ALMD 指令的相同，但存在一些差异，具体情况如上所述。

功能块



操作数	类型	格式	说明
ALMA 标签	ALARM_ANALOG	结构	ALMA 结构

ALARM_ANALOG 结构

输入参数	数据类型	说明
EnableIn	BOOL	梯形图 与梯级状态相对应。如果为清零状态，则不执行该指令，并且不更新输出。 结构化文本 无影响。始终执行该指令。 功能块 如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
In	REAL	报警输入值，其与报警限值比较以检测报警条件。 默认值 = 0.0。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
InFault	BOOL	指示输入状况不良的情况。用户应用项目可以置位 InFault 来指示输出信号有错误。置位时，指令将置位 InFaulted (Status.1)。清零时，指令将清零 InFaulted (Status.1)。无论在哪种情况下，指令都继续评估 In 是否满足报警条件。 默认为清零状态 (状况良好)。
HHEnabled	BOOL	上上限报警条件检测。置位时使能对上上限报警条件进行检测。清零时禁止对上上限报警条件进行检测。 默认为置位状态。
HEnabled	BOOL	上限报警条件检测。置位时使能对上限报警条件进行检测。清零时禁止对上限报警条件进行检测。 默认为置位状态。
LEnabled	BOOL	下限报警条件检测。置位时使能对下限报警条件进行检测。清零时禁止对下限报警条件进行检测。 默认为置位状态。
LLEnabled	BOOL	下下限报警条件检测。置位时使能对下下限报警条件进行检测。清零时禁止对下下限报警条件进行检测。 默认为置位状态。

输入参数	数据类型	说明
AckRequired	BOOL	指定是否需要确认报警。置位时，需要确认。清零时，无需确认， HHAcked、HAcked、 LAcked、 LLAcked、 ROCPosAcked 和 ROCNegAcked 始终置位。 默认为置位状态。
ProgAckAll	BOOL	通过用户程序置位，进而确认此报警的全部条件。如果报警条件未确认，则需要从清零状态转变为置位状态。 默认为清零状态。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
OperAckAll	BOOL	通过操作员界面置位，进而确认此报警的全部状态。如果报警条件未确认，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
HHProgAck	BOOL	上上限报警程序确认。通过用户程序置位，进而确认上上限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。 默认为清零状态。
HHOperAck	BOOL	上上限操作员确认。通过操作员界面置位，进而确认上上限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
HProgAck	BOOL	上限报警程序确认。通过用户程序置位，进而确认上限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。 默认为清零状态。
HOperAck	BOOL	上限操作员确认。通过操作员界面置位，进而确认上限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
LProgAck	BOOL	下限报警程序确认。通过用户程序置位，进而确认下限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。 默认为清零状态。
LOperAck	BOOL	下限操作员确认。通过操作员界面置位，进而确认下限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
LLProgAck	BOOL	下下限报警程序确认。通过用户程序置位，进而确认下下限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。 默认为清零状态。

输入参数	数据类型	说明
LLOperAck	BOOL	下下限操作员确认。通过操作员界面置位，进而确认下下限报警条件。如果报警条件未确认，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
ROCPosProgAck	BOOL	正变化率程序确认。通过用户程序置位，进而确认正变化率条件。如果报警条件未确认，则需要从清零状态转变为置位状态。 默认为清零状态。
ROCPosOperAck	BOOL	正变化率操作员确认。通过操作员界面置位，进而确认正变化率条件。如果报警条件未确认，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
ROCNegProgAck	BOOL	负变化率程序确认。通过用户程序置位，进而确认负变化率条件。如果报警条件未确认，则需要从清零状态转变为置位状态。 默认为清零状态。
ROCNegOperAck	BOOL	负变化率操作员确认。通过操作员界面置位，进而确认负变化率条件。如果报警条件未确认，则需要从清零状态转变为置位状态。该报警指令将清零此参数。 默认为清零状态。
ProgSuppress	BOOL	通过用户程序置位，进而抑制报警。 默认为清零状态。
OperSuppress	BOOL	通过操作员界面置位，进而抑制报警。该报警指令将清零此参数。 默认为清零状态。
ProgUnsuppress	BOOL	通过用户程序置位，进而取消抑制报警。其优先级高于抑制命令。 默认为清零状态。
OperUnsuppress	BOOL	通过操作员界面置位，进而取消抑制报警。其优先级高于抑制命令。该报警指令将清零此参数。 默认为清零状态。
ProgDisable	BOOL	通过用户程序置位，进而禁止报警。 默认为清零状态。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
OperDisable	BOOL	通过操作员界面置位，进而禁止报警。该报警指令将清零此参数。 默认为清零状态。

输入参数	数据类型	说明
ProgEnable	BOOL	通过用户程序置位，进而使能报警。其优先级高于禁止命令。 默认为清零状态。 梯形图 从指令操作数复制。 结构化文本 从指令操作数复制。
OperEnable	BOOL	通过操作员界面置位，进而使能报警。其优先级高于禁止命令。该报警指令将清零此参数。 默认为清零状态。
AlarmCountReset	BOOL	通过用户程序置位，进而复位所有条件的报警计数。从清零状态转变为置位状态会将报警计数复位为零。 默认为清零状态。
HHLimit	REAL	报警上上限。 有效值 = HLimit < HHLimit < 最大正浮点数。 默认值 = 0.0。
HHSeverity	DINT	上上限报警条件的严重程度。它不会影响控制器处理报警，但可在报警订阅者上用于排序和过滤功能。 有效值 = 1...1000(1000 表示严重程度最高； 1 表示严重程度最低)。 默认值 = 500。
HLimit	REAL	报警上限。 有效值 = LLimit < HLimit < HHLimit。 默认值 = 0.0。

输入参数	数据类型	说明
HSeverity	DINT	上限报警情况的严重程度。这不会影响控制器对报警的处理过程，但是可用来帮助实现报警订阅者的排序和过滤功能。 有效值 = 1...1000(1000 = 严重程度最高； 1 = 验证程度最低). 默认值 = 500。
LLimit	REAL	报警下限。 有效值 = LLLimit < LLimit < HLimit。 默认值 = 0.0。
LSeverity	DINT	下限报警条件的严重程度。它不会影响控制器处理报警，但可在报警订阅者上用于排序和过滤功能。 有效值 = 1...1000(1000 表示严重程度最高； 1 表示严重程度最低)。 默认值 = 500。
LLLimit	REAL	报警下下限。 有效值 = 最大负浮点数 < LLLimit < LLimit。 默认值 = 0.0。
LLSeverity	DINT	下下限报警条件的严重程度。它不会影响控制器处理报警，但可在报警订阅者上用于排序和过滤功能。 有效值 = 1...1000(1000 表示严重程度最高； 1 表示严重程度最低)。 默认值 = 500。
MinDurationPRE	DINT	报警级别条件保持为真的最小持续时间预设值 (毫秒)，在此时间过后该条件将标记为 InAlarm 并且将向客户端发送报警通知。控制器在检测到报警条件后立即收集报警数据，所以在等待满足最小持续时间时不会丢失任何数据。这不适用于变化率条件。 MinDurationPRE 仅适用于在任一方向上从正常状态的首次变化。例如，当上限报警条件超时后，上上限报警条件将立即激活，而下限报警条件将等待超时周期。 有效值 = 0...2,147,483,647。 默认值 = 0。

输入参数	数据类型	说明
Deadband	REAL	死区用于检测上上限、上限、下限和下下限报警级别是否返回到正常状态。 如果 In 值连续变化，但仍保持在级别条件阈值附近，则非零死区可降低报警条件波动的情况。 Deadband 值不会影响向 InAlarm(激活) 状态的转换。当级别条件激活后，但在该条件返回到未激活 (正常) 状态前， In 值必须： • 下降到低于阈值与死区之差 (仅限上限报警和上上限报警条件)。 • 上升到高于阈值与死区之和 (仅限下限报警和下下限报警条件)。 Deadband 不能用于调整最小持续时间测量。 有效值 = 0 ≤ Deadband < 从首个使能的下限报警到首个使能的上限报警这一范围。 默认值 = 0.0。
ROCPosLimit	REAL	用单位数 / 秒表示的增加变化率的限值。如果 ROCPeriod 也 > 0.0，则将对任何 > 0.0 值使能检测。 有效值 = 0.0... 允许的最大浮点数。 默认值 = 0.0。
ROCPosSeverity	DINT	增加变化率条件的严重程度。它不会影响控制器处理报警，但可在报警订阅者上用于排序和过滤功能。 有效值 = 1...1000(1000 表示严重程度最高； 1 表示严重程度最低)。 默认值 = 500。
ROCNegLimit	REAL	用单位数 / 秒表示的减低变化率的限值。如果 ROCPeriod 也 > 0.0，则将对任何 > 0.0 值使能检测。 有效值 = 0.0... 允许的最大浮点数。 默认值 = 0.0。
ROCNegSeverity	DINT	减低变化率条件的严重程度。它不会影响控制器处理报警，但可在报警订阅者上用于排序和过滤功能。 有效值 = 1...1000(1000 表示严重程度最高； 1 表示严重程度最低)。 默认值 = 500。
ROCPeriod	REAL	计算变化率值的时间周期 (采样间隔)，以秒为单位。每次采样时间间隔到期后，都将存储 In 中的新采样，并重新计算 ROC。 对任何 > 0.0 的值使能变化率检测。 有效值 = 0.0... 允许的最大浮点数。 默认值 = 0.0。

输出参数	数据类型	说明
EnableOut	BOOL	使能输出。
InAlarm	BOOL	报警激活状态。只要有报警条件激活，就置位。所有报警条件都未激活时 (正常状态) 清零。
AnyInAlarmUnack	BOOL	组合式报警激活和确认状态。检测到任何报警条件但未对其进行确认时置位。所有报警条件均处于正常 (未激活) 状态、已确认或者这两种情况时清零。
HHInAlarm	BOOL	上上限报警条件状态。存在上上限报警条件时置位。不存在上上限报警条件时清零。
HInAlarm	BOOL	上限报警条件状态。存在上限报警条件时置位。不存在上限报警条件时清零。
LInAlarm	BOOL	下限报警条件状态。存在下限报警条件时置位。不存在下限报警条件时清零。
LLInAlarm	BOOL	下下限报警条件状态。存在下下限报警条件时置位。不存在下下限报警条件时清零。
ROCPosInAlarm	BOOL	正变化率报警条件状态。存在正变化率条件时置位。不存在正变化率条件时清零。
ROCNegInAlarm	BOOL	负变化率报警条件状态。存在负变化率条件时置位。不存在负变化率条件时清零。
ROC	REAL	计算出的 In 值变化率。每次 ROCPeriod 到期后扫描指令时，更新此值。ROC 值用于评估 ROCPosInAlarm 和 ROCNegInAlarm 条件。 ROC = (In 的当前采样 - In 的上次采样)/ROCPeriod。
HHAcked	BOOL	上上限报警条件已确认状态。上上限报警条件已确认时置位。AckRequired 清零时始终置位。上上限报警条件未确认时清零。
HAcked	BOOL	上限报警条件已确认状态。上限报警条件已确认时置位。 AckRequired 清零时始终置位。上限报警条件未确认时清零。
LAcked	BOOL	下限报警条件已确认状态。下限报警条件已确认时置位。 AckRequired 清零时始终置位。下限报警条件未确认时清零。
LLAcked	BOOL	下下限报警条件已确认状态。下下限报警条件已确认时置位。AckRequired 清零时始终置位。下下限报警条件未确认时清零。
ROCPosAcked	BOOL	正变化率条件已确认状态。正变化率条件已确认时置位。 AckRequired 清零时始终置位。正变化率条件未确认时清零。
ROCNegAcked	BOOL	负变化率条件已确认状态。负变化率条件已确认时置位。 AckRequired 清零时始终置位。负变化率条件未确认时清零。
HHInAlarmUnack	BOOL	组合式上上限报警条件激活且已确认状态。上上限报警条件激活 (HHInAlarm 置位) 且已确认时置位。上上限报警条件处于正常 (未激活) 状态、已确认或这两种情况时清零。
HInAlarmUnack	BOOL	组合式上限报警条件激活且已确认状态。上限报警条件激活 (HInAlarm 置位) 且已确认时置位。上限报警条件处于正常 (未激活) 状态、已确认或这两种情况时清零。

输出参数	数据类型	说明
LInAlarmUnack	BOOL	组合式下限报警条件激活且已确认状态。下限报警条件激活 (LInAlarm 置位) 且已确认时置位。下限报警条件处于正常 (未激活) 状态、已确认或这两种情况时清零。
LLInAlarmUnack	BOOL	组合式下下限报警条件激活且已确认状态。下下限报警条件激活 (LLInAlarm 置位) 且已确认时置位。下下限报警条件处于正常 (未激活) 状态、已确认或这两种情况时清零。
ROCPosInAlarmUnack	BOOL	组合式正变化率条件激活但未确认状态。正变化率条件激活 (ROCPosInAlarm 置位) 但未确认时置位。正变化率条件处于正常 (未激活) 状态、已确认或这两种情况时清零。
ROCNegInAlarmUnack	BOOL	组合式负变化率条件激活但未确认状态。负变化率条件激活 (ROCNegInAlarm 置位) 但未确认时置位。负变化率条件处于正常 (未激活) 状态、已确认或这两种情况时清零。
Suppressed	BOOL	报警的受抑制状态。报警受抑制时置位。报警未受抑制时清零。
Disabled	BOOL	报警的禁止状态。报警禁止时置位。报警启用时清零。
MinDurationACC	DINT	从检测到报警条件以来所经历的时间。当此值达到 MinDurationPRE 时，所有检出的报警级别条件将激活 (*x*InAlarm 置位)，并向各客户端发送通知。
HHInAlarmTime	LINT	ALMA 指令最近一次检测到 In 值超出上上限报警条件限值而转换到激活状态时的时间戳。
HHAlarmCount	DINT	上上限报警条件激活的次数。如果达到最大值，则计数器将保持该值为最大计数值。
HInAlarmTime	LINT	ALMA 指令最近一次检测到 In 值超出上限报警条件限值而转换到激活状态时的时间戳。
HAlarmCount	DINT	上限报警条件激活的次数。如果达到最大值，则计数器将保持该值为最大计数值。
LInAlarmTime	LINT	ALMA 指令最近一次检测到 In 值超出下限报警条件限值而转换到激活状态时的时间戳。
LAlarmCount	DINT	下限报警条件激活的次数。如果达到最大值，则计数器将保持该值为最大计数值。
LLInAlarmTime	LINT	ALMA 指令最近一次检测到 In 值超出下下限报警条件限值而转换到激活状态时的时间戳。
LLAlarmCount	DINT	下下限报警条件激活的次数。如果达到最大值，则计数器将保持该值为最大计数值。
ROCPosInAlarmTime	LINT	ALMA 指令最近一次检测到 In 值超出正变化率条件限值而转换到激活状态时的时间戳。
ROCPosInAlarmCount	DINT	正变化率条件激活的次数。如果达到最大值，则计数器将保持该值为最大计数值。
ROCNegInAlarmTime	LINT	ALMA 指令最近一次检测到 In 值超出负变化率条件限值而转换到激活状态时的时间戳。
ROCNegAlarmCount	DINT	负变化率条件激活的次数。如果达到最大值，则计数器将保持该值为最大计数值。

输出参数	数据类型	说明
AckTime	LINT	最新条件确认的时间戳。如果报警无需确认，则此时间戳等于最新报警条件的时间。
RetToNormalTime	LINT	报警返回到正常状态的时间戳。
AlarmCountResetTime	LINT	指示报警计数复位时间的时间戳。
DeliveryER	BOOL	报警通知信息传输错误。存在传输错误时置位，传输错误可能是没有预定任何报警订阅者，或者至少一个订阅者没有收到最新的报警变化状态信息。传输成功或正在传输时清零。
DeliveryDN	BOOL	报警通知信息传输完成。传输成功时置位，传输成功是指至少预定一个订阅者且所有订阅者均成功接收最新的报警变化状态信息。传输未成功完成或正在传输时清零。
DeliveryEN	BOOL	报警通知信息的传输状态。正在传输时置位。未进行传输时清零。
NoSubscriber	BOOL	尝试传输最新信息时报警没有任何订阅者。没有任何订阅者时置位。至少有一个订阅者时清零。
NoConnection	BOOL	尝试传输最新信息时未连接报警订阅者。与所有订阅者的连接断开时置位。至少连接一个订阅者或没有任何订阅者时清零。
CommError	BOOL	传输报警信息时的通信错误。存在通信错误且所有重试次数均已用完时置位。也就是说，预定了订阅者且其具有连接，但控制器未收到确认传输信息的信息。所有已连接的订阅者都确认接收报警信息时清零。
AlarmBuffered	BOOL	由于存在通信错误 (CommError 置位) 或连接丢失 (NoConnection 置位) 而缓冲报警信息。至少为一个订阅者缓冲报警信息时置位。非缓冲报警信息时清零。
Subscribers	DINT	此报警的订阅者数。
SubscNotified	DINT	成功接到最新的报警状态变化通知的订阅者的数量。
Status	DINT	组合状态指示器： Status.0 = InstructFault。 Status.1 = InFaulted。 Status.2 = SeverityInv。 Status.3 = AlarmLimitsInv。 Status.4 = DeadbandInv。 Status.5 = ROCPosLimitInv。 Status.6 = ROCNegLimitInv。 Status.7 = ROCPeriodInv。
InstructFault (Status.0)	BOOL	存在指令错误情况。这不是次要或主要控制器错误。检查其余状态位，确定发生的情况。

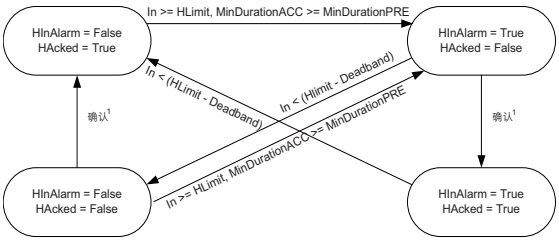
输出参数	数据类型	说明
InFaulted (Status.1)	BOOL	用户程序已置位 InFault，这说明输入数据质量不良。报警将继续评估 In 是否满足报警条件。
SeverityInv (Status.2)	BOOL	报警严重程度组态无效。 如果严重程度 < 1，则指令使用 Severity = 1。 如果严重程度 > 1000，则指令使用 Severity = 1000。
AlarmLimitsInv (Status.3)	BOOL	报警限值组态无效 (例如， LLimit < LLLimit)。如果无效，指令将清零所有级别条件激活位。在排除该故障前，无法检测任何新的级别条件。
DeadbandInv (Status.4)	BOOL	死区组态无效。如果无效，指令将使用 Deadband = 0.0。 有效值 = 0 ≤ Deadband < 从首个使能的下限报警到首个使能的上限报警这一范围。
ROCPosLimitInv (Status.5)	BOOL	正变化率限值无效。如果无效，指令将使用 ROCPosLimit = 0.0，从而禁止正变化率检测。
ROCNegLimitInv (Status.6)	BOOL	负变化率限值无效。如果无效，指令将使用 ROCNegLimit = 0.0，从而禁止负变化率检测。
ROCPeriodInv (Status.7)	BOOL	变化率周期无效。如果无效，指令将使用 ROCPeriod = 0.0，从而禁止变化率检测。

说明　ALMA 指令根据值的变化率或变化水平来检测报警。

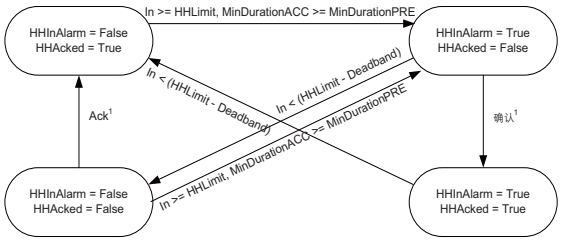
当与 RSLinx Enterprise 和 FactoryTalk View SE 软件配合使用时，ALMA 指令可提供附加功能。可在 FactoryTalk View SE 软件中的“ 报警摘要 ” (Alarm Summary)、“ 报警显示条 ” (Alarm Banner)、“ 报警状态浏览器 ” (Alarm Status Explorer) 和 “ 报警记录查看器 ” (Alarm Log Viewer) 显示画面中显示报警。

RSLinx Enterprise 软件订阅控制器中的报警。使用输出参数可监视指令，以查看报警订阅状态和显示报警状态变化。如果与 RSLinx Enterprise 软件的连接中断，控制器可暂时缓冲报警数据直到恢复连接。

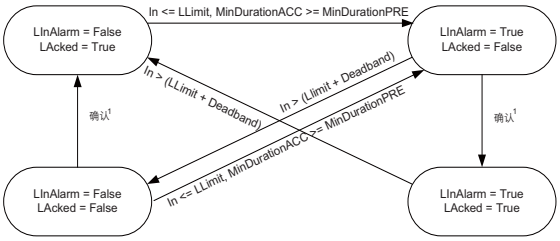
需要确认时的状态图



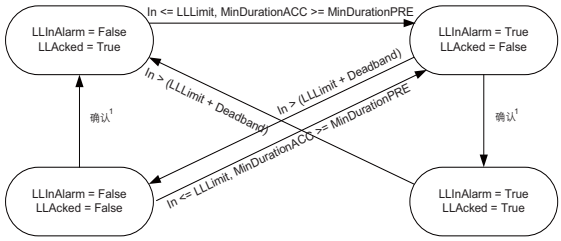
[1] 可以通过多种不同的方式来确认上限报警条件：HProgAck、HOperAck、ProgAckAll、OperAckAll、客户机 (RSLogix 5000 软件、RSView 软件)。



[1] 可以通过多种不同的方式来确认上上限报警条件： HHProgAck、HHOperAck、ProgAckAll、OperAckAll、客户机 (RSLogix 5000 软件、RSView 软件)。



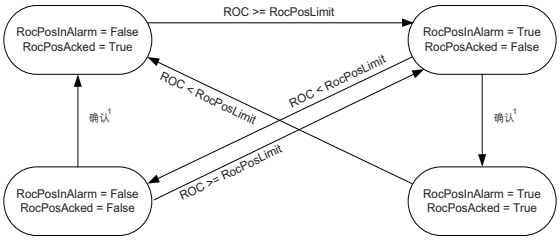
[1] 可以通过多种不同的方式来确认下限报警条件： LProgAck、LOperAck、ProgAckAll、OperAckAll、客户机 (RSLogix 5000 软件、RSView 软件)。



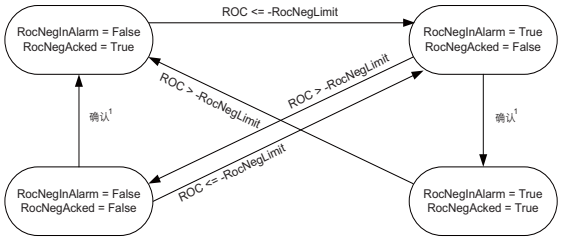
[1] 可以通过多种不同的方式来确认下下限报警条件： LLProgAck、LLOperAck、ProgAckAll、OperAckAll、客户机 (RSLogix 5000 软件、RSView 软件)。

$$ROC = \frac{\ln(CurrentSample) - \ln(PreviousSample)}{ROCPeriod}$$

其中，在 ROCPeriod 时间到达后的下一次扫描中采集到新样本。

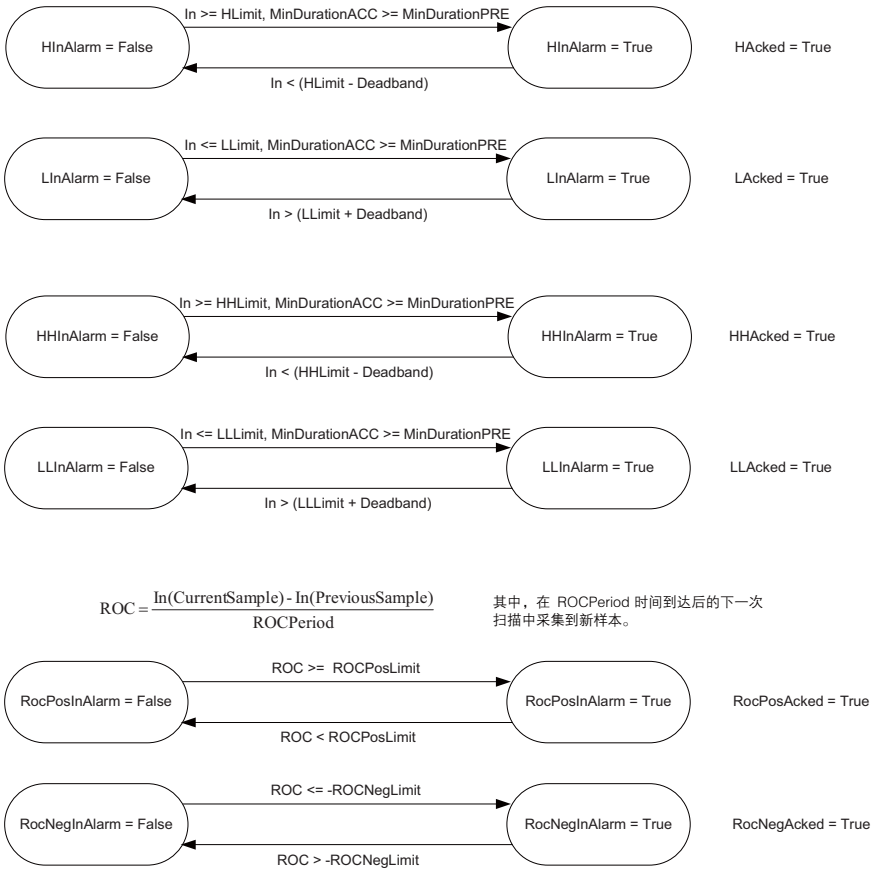


[1] 可以通过多种不同的方式来确认 ROCPos 报警条件： RocPosProgAck、RocPosOperAck、ProgAckAll、OperAckAll、客户机（RSLogix 5000 软件、RSView 软件）



[1] 可以通过多种不同的方式来确认 ROCNeg 报警条件： RocNegProgAck、RocNegOperAck、ProgAckAll、OperAckAll、客户机 (RSLogix 5000 软件、RSView 软件)。

不需要确认时的状态图



算术状态标志： 将为 ROC 输出设置算术状态标志。

故障条件：

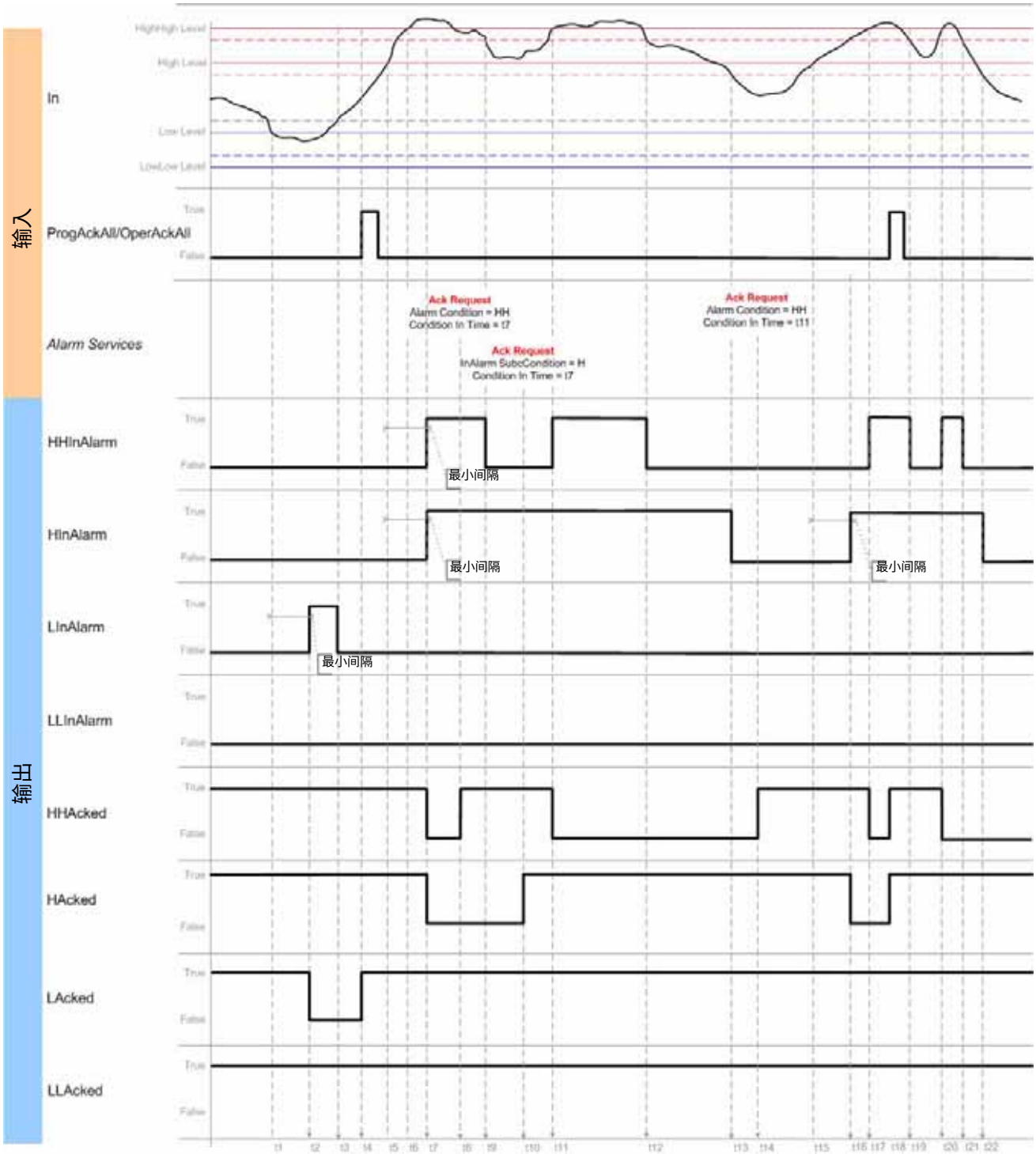
次要故障	故障类型	故障代码
ROC 溢出	4	4

执行：

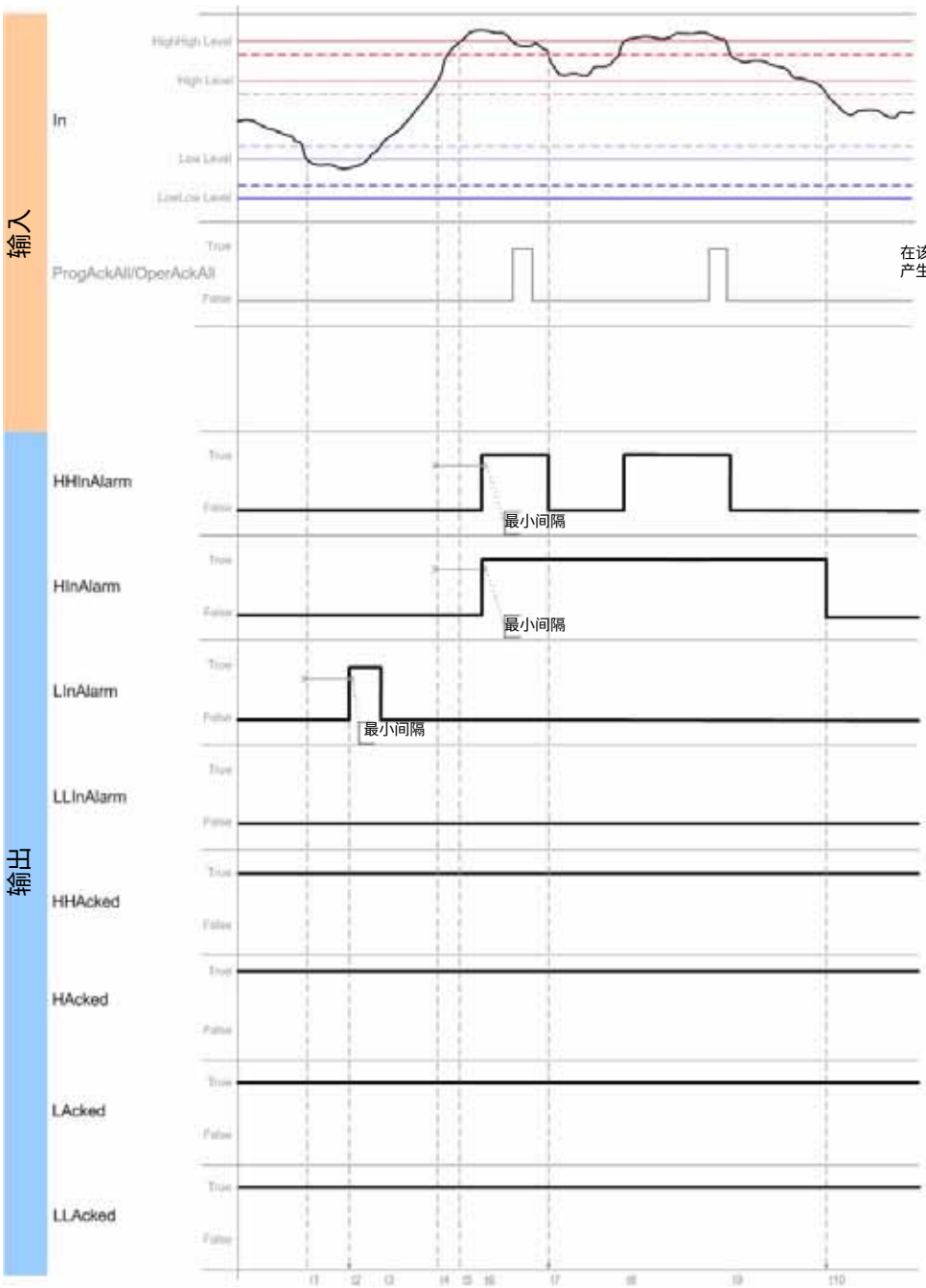
条件	梯形图操作
预扫描	梯级输出条件设置为假。 清零所有 *x*InAlarm 参数，确认所有报警条件。 清除所有操作员请求、时间戳和传输标记。
梯级输入条件为假	不执行该指令。 清零 EnableOut。
梯级输入条件为真	执行该指令。 置位 EnableOut。
后扫描	梯级输出条件设置为假。

条件	功能块操作	结构化文本操作
预扫描	清除所有操作员请求、时间戳和传输标记。 清零所有 *x*InAlarm 参数，确认所有报警条件。	清除所有操作员请求、时间戳和传输标记。 清零所有 *x*InAlarm 参数，确认所有报警条件。
指令第一次扫描	不执行任何操作。	不执行任何操作。
指令第一次运行	不执行任何操作。	不执行任何操作。
EnableIn 处于清零状态	不执行该指令。 清零 EnableOut。	执行该指令。 EnableOut 始终置位。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。	执行该指令。 EnableOut 始终置位。
后扫描	不执行任何操作。	不执行任何操作。

ALMA 级别条件需要确认

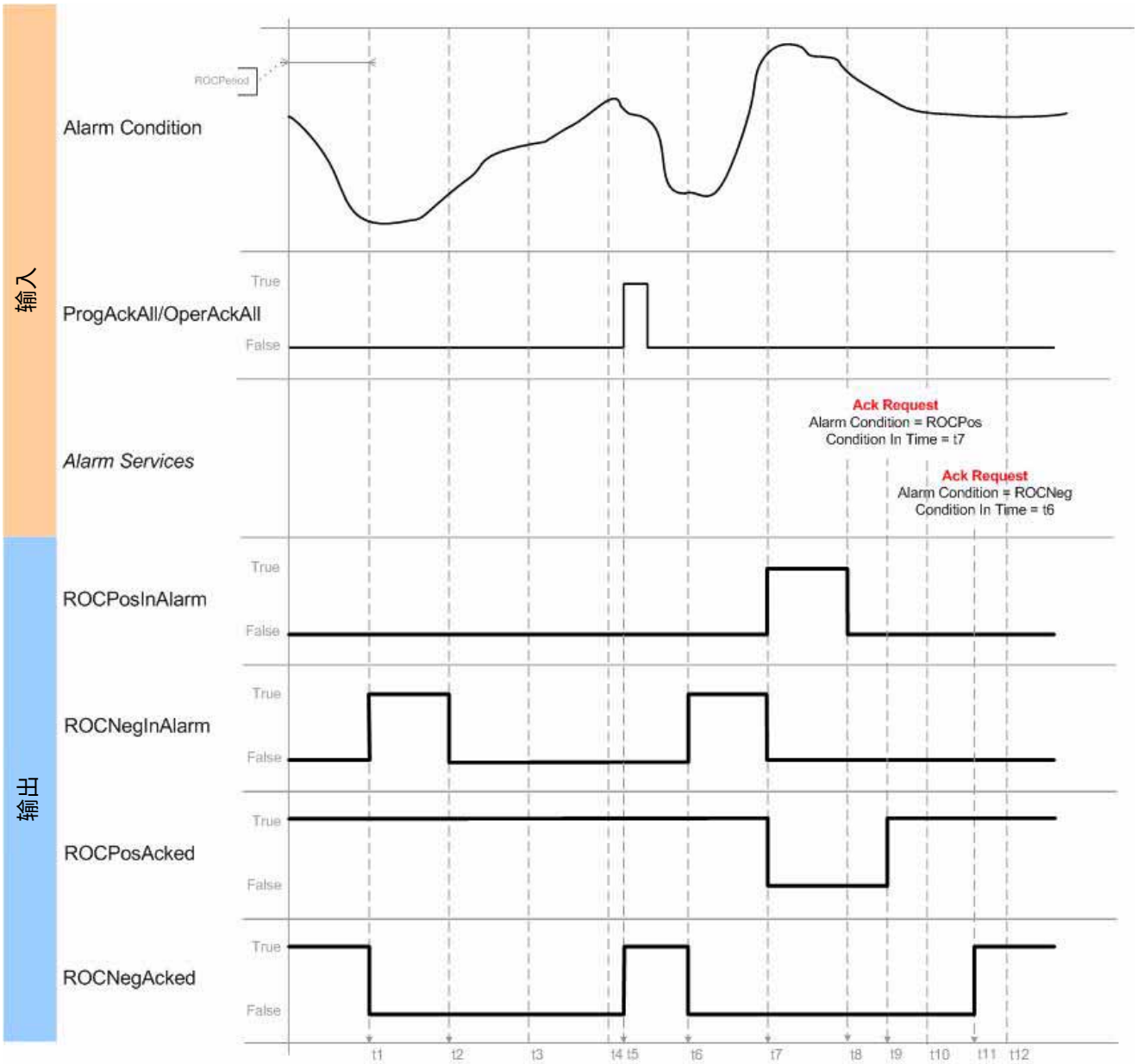


ALMA 级别条件不需要确认

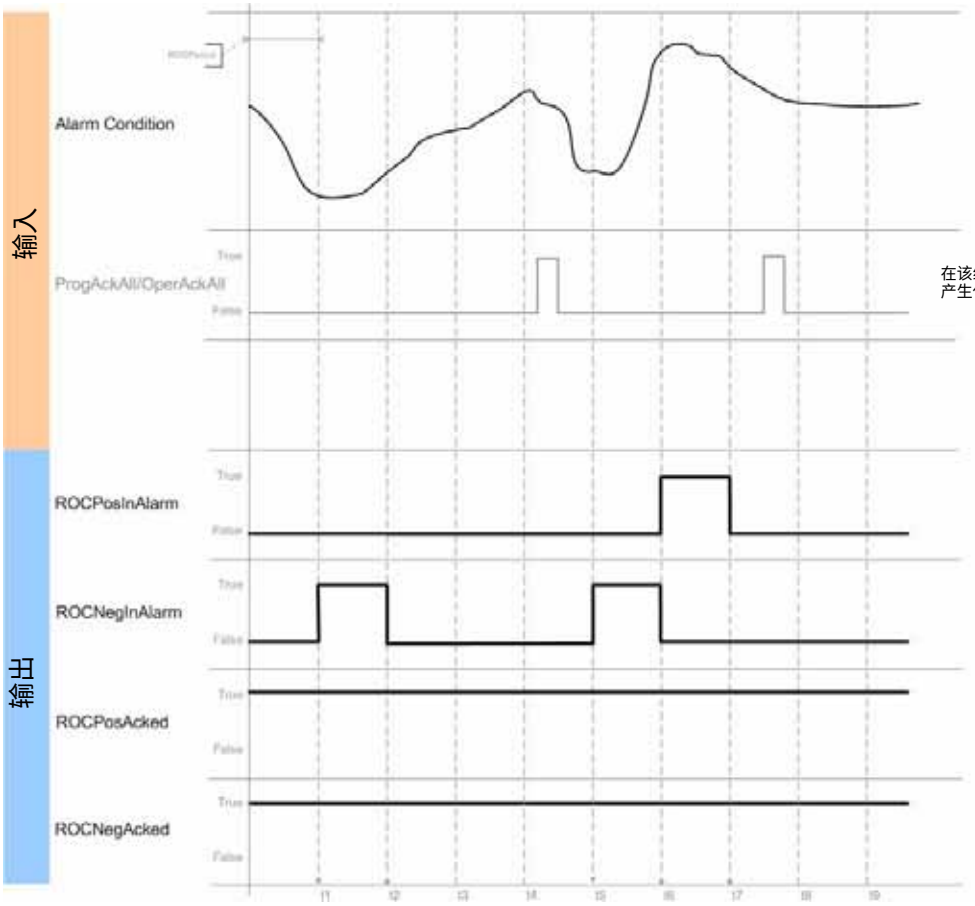


在该组态情况下，此信号不会对报警状态变化产生任何影响。

ALMA 变化率需要确认

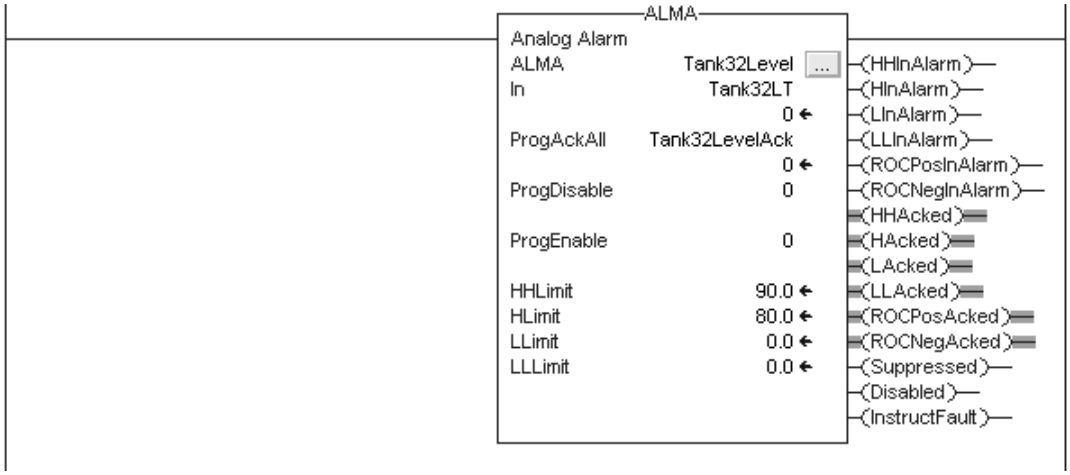


ALMA 变化率不需要确认



示例： 如果罐液位超过上限或上上限，将激活罐报警。通过程序将 Tank32LevelAck 标签值从清零状态转变为置位状态来确认所有报警条件。应用程序逻辑必须清零 Tank32LevelAck。

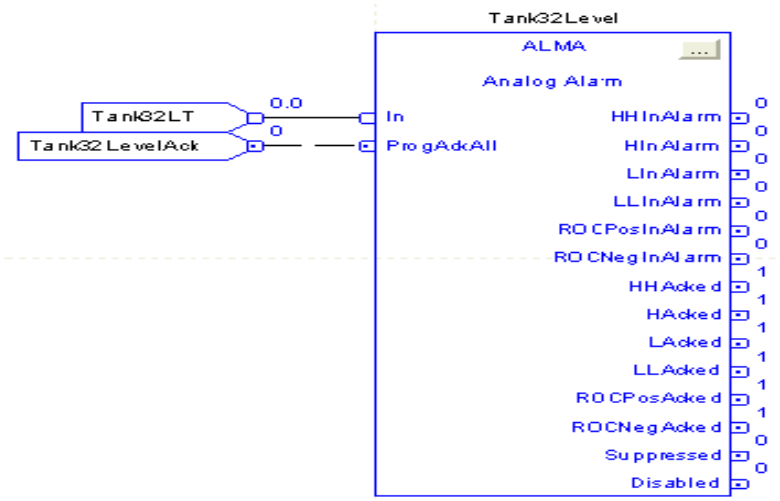
梯形图



结构化文本

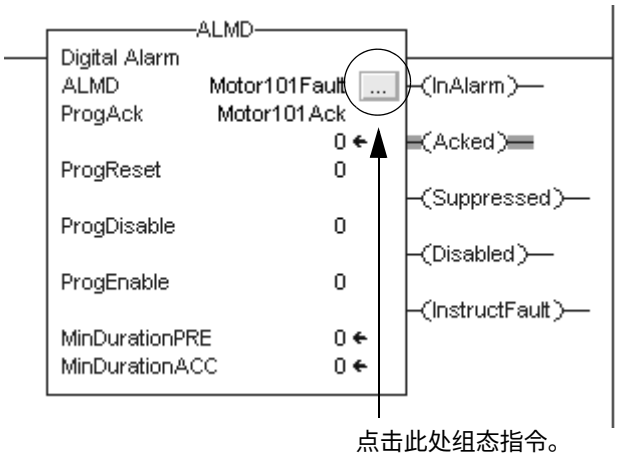
```
ALMA(Tank32Level,Tank32LT,Tank32LevelAck,0, 0);
```

功能块

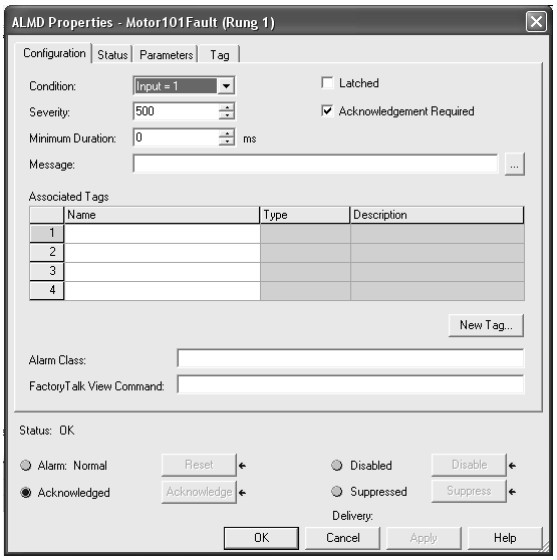


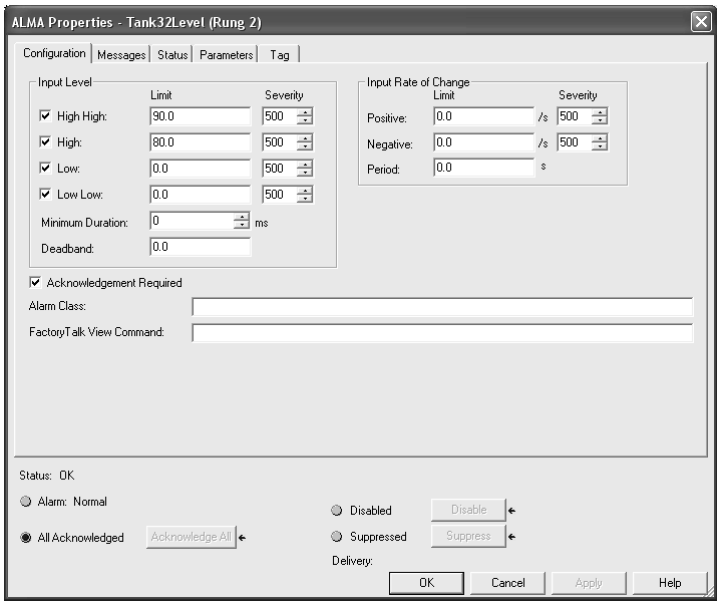
组态报警指令

输入 ALMD 或 ALMA 指令并指定报警标签名称后，可使用“ 报警组态 ”(Alarm Configuration) 对话框指定信息的详细内容。



报警指令的“ 属性 ”(Properties) 对话框包含“ 组态 ”(Configuration) 页面。





为各报警指令组态以下信息。

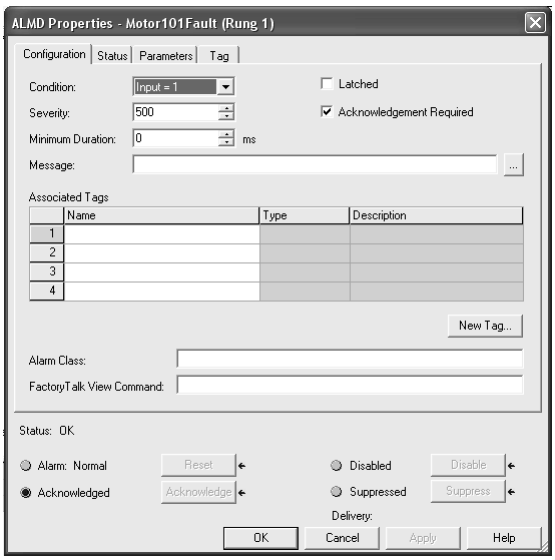
选项	说明
条件 - ALMD 指令	触发报警的条件。 对于 In = 1 时激活的报警，选择 “ 输入 =1 ”。对于 In= 0 时激活的报警，选择 “ 输入 =0 ”。
输入级别 - ALMA 指令 输入变化率 - ALMA 指令	触发报警的输入级别 (上上限、上限、下限或下下限) 或输入变化率 (正变化率或负变化率)。 选择报警条件并为这些条件输入限值。若要禁止变化率条件，可为周期或限值输入 0。
严重程度	从 1...1000 的严重程度范围内进行选择，对报警条件的重要性进行分级。严重程度为 1 表示报警优先级低；严重程度为 1000 表示是紧急条件。 默认情况下，在 FactoryTalk 报警和事件系统中，严重程度范围与优先级的映射关系如下： • 1…250 低优先级。 • 251…500 为中优先级。 • 501…750 为高优先级。 • 751…1000 为紧急优先级。 用户可在 FactoryTalk 报警和事件系统中组态严重程度到优先级的映射。有关详细信息，请参见 FactoryTalk 帮助。
最小持续时间	输入以 ms 为单位的时间值，报警条件保持激活状态达该时间后才会报告报警。
锁定 - ALMD 指令	如果要在报警条件返回到未激活 (正常) 状态后报警仍保持激活状态 (InAlarm)，则选择 “ 锁定 ” (Latched)。锁定的报警需要使用复位命令来转换到正常状态。必须在条件返回到正常状态后接收复位命令。 确认命令不会复位锁定的报警。
死区 - ALMA 指令	指定 “ 死区 ” (Deadband) 值可减少由于 In 值微小波动而导致报警条件波动的情况。 死区值并不影响导致转换到激活状态的报警限值，而且在最小持续时间间隔期间也不使用该值。 当级别条件激活 (InAlarm) 后，其保持激活状态，直到 In 值重新超出限值达指定的死区或以上。例如，如果上限为 80，下限为 20，死区为 5，则上限报警条件将在≥ 80 时激活并在≤ 75 时返回正常状态；下限报警条件在 ≤ 20 时激活，并在≥≤ 25 时返回正常状态。 死区对变化率报警条件无任何影响。

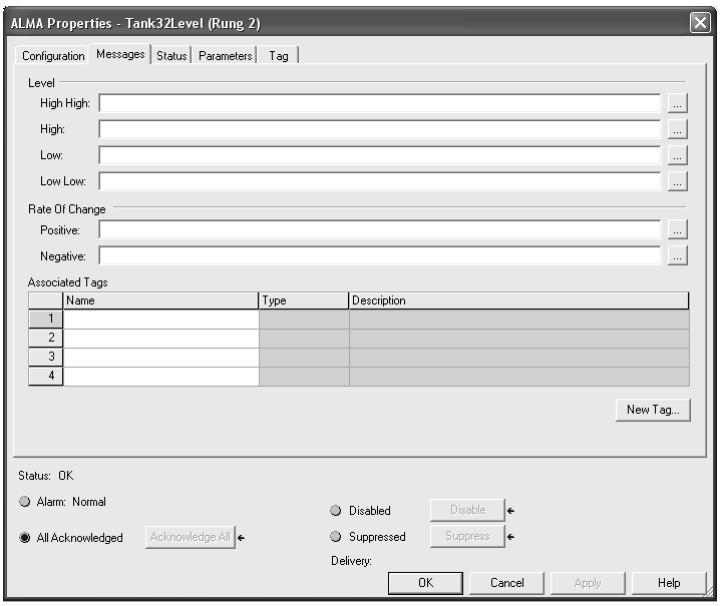
选项	说明
需要确认	报警默认组态为需要确认。确认表明操作员知道了报警条件，无论报警条件是否已返回到正常状态。 如果要报警在无操作员介入的情况下在 HMI 上的“ 报警摘要 ”(Alarm Summary) 上自出现并随后消失，则清除“ 需要确认 ”(Acknowledgement Required) 设置。 不需要确认的报警会使 Acked 状态始终为置位。 如果数字报警组态为锁定，则复位命令也可确认该报警。
报警类别	使用报警类别可对相关报警分组。对于要分为同一类的各个报警，具体指定相同的报警类别。报警类别区分大小写。 例如，指定类别“ 油罐区 A ”(Tank Farm A) 对特定区域的所有油罐报警分组。或指定类别“ 控制回路 ”(Control Loop) 对 PID 回路的所有报警分组。 然后可在 HMI 上根据类别显示和过滤报警。例如，操作员可显示所有油罐报警或所有 PID 回路报警。 报警类别不限制“ 报警摘要 ”(Alarm Summary) 对象订阅的报警。“ 报警摘要 ”(Alarm Summary) 对象接收报警后，使用报警类别可过滤显示给操作员的报警。 FactoryTalk View 软件可以过滤报警类别，支持使用通配符代替字母。
查看命令	当操作员针对特定报警提出请求时，在操作站执行命令。从而操作员可执行所有标准 FactoryTalk View 命令，例如，调用特定面板和显示画面、执行宏、访问帮助文件和启动外部应用程序。当出现报警条件并显示给操作员时，操作员可使用摘要和显示条画面中的按钮来执行相关查看命令。 注意输入正确的命令语法，并且在运行时对命令进行测试时也要多加小心，因为在输入命令时并不执行任何错误检查。

用户可以离线和在线对报警组态的所有方面进行编辑。对新报警和现有报警的在线编辑会立即发送到 FactoryTalk 订阅者 (仅轮询标签的早期 HMI 终端不自动更新)。 FactoryTalk 订阅者不必重新订阅便可接收更新信息。在线更改会自动从控制器报警结构传播到架构的其余部分。

输入报警信息文本

输入要在报警条件激活 (InAlarm) 时显示的合适信息文本。对于 ALMD 指令，在 “ 组态 ” (Configuration) 页面中输入信息内容。对于 ALMA 指令，在 “ 信息 ” (Message) 页面中输入信息内容。





要定义报警信息，请指定以下内容。

选项	说明
信息字符串	信息字符串包含显示给操作员的报警相关内容。除输入文本外，还可嵌入变量信息。在报警信息编辑器中，选择所需变量并将其添加到信息字符串中的任意所需位置。 信息字符串最多可包含 255 个字符，这包括指定任何嵌入变量的字符 (不是嵌入变量的实际值中的字符数)。例如， /*S:0 %Tag1*/ 指定一个字符串标签，其使总字符串长度增加 13 个字符，但此字符串标签的实际值可能包含 82 个字符。 用户无法在程序中通过报警标签访问报警信息字符串。要根据特定事件更改报警信息，请将某一关联标签组态为字符串数据类型，并将该关联标签嵌入到信息中。 可以使用多种语言版本的信息。通过导入 / 导出实用工具输入不同语言。有关详细信息，请参见第 67 页。

选项	说明
关联标签	最多可从控制器项目中选择四个附加标签与报警关联。这些标签的值随报警信息一起发送到报警服务器。例如，泄压阀的数字报警可能还包括泵速度和油罐温度等信息。 关联标签可以是任何基本数据类型 (BOOL、 DINT、 INT、 SINT 或 REAL) 或字符串。它们可以是 UDT 或数组中的元素。不允许变量数组引用。如果报警属于控制器区域，则关联标签也必须属于控制器区域。 还可在信息文本字符串中嵌入关联标签。 无论是否将关联标签嵌入到信息文本字符串中，关联标签值始终会随报警一起发送，操作员可以查看，并输入到历史记录中。

信息字符串变量

用户可将以下变量信息嵌入到信息字符串中。

变量	嵌入到信息字符串	添加到信息字符串的默认代码
报警名称	报警的名称，由控制器名称、程序名称和标签名称组成。例如，[Zone1Controller]Program:Main.MyAlarmTagName。	`/*S:0 %AlarmName*/`
条件名称	触发报警的条件： • 数字报警显示跳闸。 • 模拟报警显示 HiHi、 Hi、 Lo、 LoLo、 ROC_POS 或 ROC_NEG。	`/*S:0 %ConditionName*/`
输入值	报警的输入值： • 数字报警显示 0 或 1。 • 模拟报警显示报警当前监视的输入变量的值。	`/*N:5 %InputValue NOFILL DP:0*/`
限值	报警的阈值： • 数字报警显示 0 或 1。 • 模拟报警显示针对模拟报警条件的实际组态范围检查。	`/*N:5 %LimitValue NOFILL DP:0*/`
严重程度	为报警条件组态的严重程度。	`/*N:5 %Severity NOFILL DP:0*/`
关联标签的值	组态的随报警事件一起提供的标签值。	`/*N:5 %Tag1 NOFILL DP:0*/`

代码随所选标签类型的不同而不同，无论标签值中有多少个数字或字符，以及是否要在左侧的空位填充空格或零。例如：

标签	代码
BOOL 值	`/*N:1 %Tag1 NOFILL DP:0*/`
DINT 值， 9 位，左侧填充空格	`/*N:9 %Tag2 SPACEFILL DP:0*/`
REAL 输入值， 9 位 (包括小数点)，小数点后 3 位，左侧填充零	`/*N:9 %InputValue NOFILL DP:3*/`
REAL 值， 8 位 (包括小数点)，小数点后 4 位，左侧填充零	`/*N:8 %Tag3 ZEROFILL DP:4*/`
字符串值，无固定宽度	`/*S:0 %Tag4*/`
字符串值， 26 个字符，固定宽度	`/*S:26 %Tag4*/`

无论是否将所有这些变量信息都嵌入到信息文本中，它们都将随报警数据一起提供，操作员可以查看，并输入到历史记录中。

多语言版本报警信息

用户可保持报警信息为多种语言。可以在相关语言版本的 RSLogix 5000 编程软件中输入不同语言，也可在导入 / 导出 (.CSV 或 .TXT) 文件中输入不同语言。

用户可在导入 / 导出 (.CSV 或 .TXT) 文件中访问报警信息文本，并添加附加行供原始信息字符串的翻译版本使用。不同语言形式的信息使用 “ 类型 ” (TYPE) 列中的 ISO 语言代码。显示给操作员的报警信息文本 (包括嵌入的变量代码) 位于 “ 说明 ” (DESCRIPTION) 列中。“ 说明符 ” (SPECIFIER) 标识报警条件。

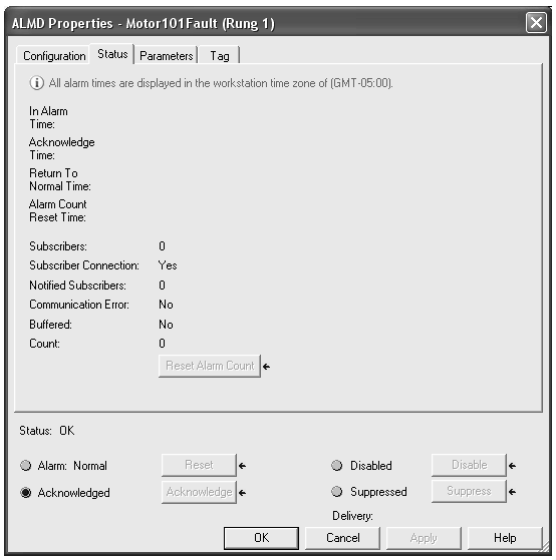
TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER
TAG		alma1		ALARM_ANALOG	
ALMMSG:en-us		alma1	HH alarm for operator in English		HH
ALMMSG:en-us		alma1	H alarm for operator in English		H
ALMMSG:en-us		alma1	L alarm for operator in English		L
ALMMSG:en-us		alma1	LL alarm for operator in English		LL
ALMMSG:en-us		alma1	ROC positive alarm for operator in English		POS
ALMMSG:en-us		alma1	ROC positive alarm for operator in English		NEG
ALMMSG:de-ch		alma1	HH Mitteilung für den Operator auf Deutsch		HH
ALMMSG:de-ch		alma1	H Mitteilung für den Operator auf Deutsch		H
TAG		almd1		ALARM_DIGITAL	
ALMMSG:en-us		almd1	digital alarm for operator in English		AM

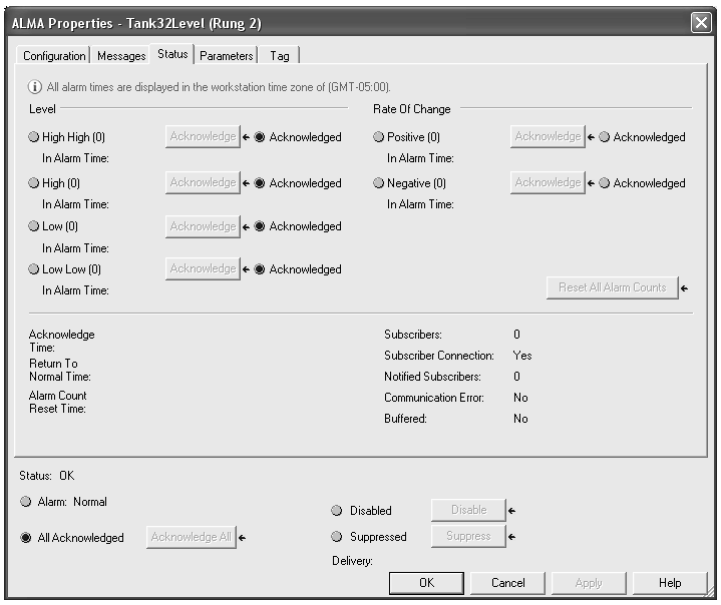
使用导入 / 导出实用工具可创建信息字符串并将其翻译为多种语言。.TXT 导入 / 导出格式支持双字节字符，因此可对所有语言 (包括中文、日文和韩语) 使用此格式。 .CSV 导入 / 导出格式不支持双字节字符。

导入和导出信息始终执行合并。在 .CSV 或 .TXT 文件中删除信息并不会删除 .ACD 文件中的对应信息。要删除信息，导入类型、名称和说明符字段已填充但说明字段空白的 .CSV 或 .TXT 文件。

监视报警状态

在报警对话框的“ 状态 ”(Status) 页面中，可以监视报警条件、确认报警、禁止报警、抑制报警或复位报警。使用此对话框选项可查看报警行为，无需运行 HMI。





缓冲报警

要接收基于控制器的报警信息，报警客户端 (例如 RSLinx Enterprise 服务器) 必须在 Logix 控制器中建立对报警的订阅。该控制器保持与各个订阅者的连接，并监视该连接的状态。

如果发生报警状态变化，控制器中的报警指令将缓存必要信息 (例如，时间戳和关联标签值) 并请求将报警信息传送到所有订阅者。发布者机构将报警信息尽可能快地传送到各个订阅者。

如果任何订阅者没有确认接收到报警信息，或者与已知订阅者的连接不良，则发布者机构会将未送达的报警信息存储在 100 KB 的缓冲区中。每个订阅者都有自己的缓冲区，因此一个订阅者发生通信问题不会干扰向其它订阅者传送报警。缓冲区已满时，将丢失新的报警信息。在订阅者首次建立连接时创建缓冲区，在订阅者断开其连接后保持可组态的时长 (0...120 分钟，默认值为 20 分钟)。

如果订阅者在缓冲超时时间间隔内重新建立连接，则将获得所有报警的当前状态，开始接收当前报警信息，还将上传可能已堆积的所有被缓冲的信息。即使缓冲区已满，信息已被丢弃，但订阅者仍可精确同步到报警的当前状态 (包括最新的 InAlarmTime、 RetToNormalTime 和 AckTime 时间戳)。

缓冲区连续工作，直到填满为止。填满后，缓冲区停止缓冲报警，直到订阅者使缓冲区中腾出空间。

通过程序访问报警信息

每个报警指令都具有用来存储报警组态和执行信息的报警结构。报警结构包括程序和操作员控制元素以及操作员元素。报警指令不通过模式设置来确定程序访问或操作员访问是否激活，因此这些元素始终激活。

共有三种对报警指令执行操作的方法 .

访问	报警结构元素	考量因素
用户程序	• ProgAck • ProgReset • ProgSuppress • ProgDisable • ProgEnable	使用通过程序访问报警系统元素的控制器逻辑。例如，控制程序可决定是否禁止与某一根原因相关的一系列报警。例如，控制程序可通过访问标签子元素 MyDigitalAlarm.ProgDisable 来禁止 ALARM_DIGITAL 数据类型的报警指令 MyDigitalAlarm。
自定义 HMI	• OperAck • OperReset • OperSuppress • OperDisable • OperEnable	创建用于访问报警系统元素的自定义 HMI 面板。例如，如果操作员需要从报警画面中删除一个工具，而不是分别手动禁止或抑制报警，那么操作员可以按下访问标签 MyDigitalAlarm.OperDisable 的禁止按键。 操作员参数与任何罗克韦尔自动化或第三方操作员界面配合可控制报警状态。 设置操作员参数后，指令评估其是否可以响应请求，然后始终复位该参数。
标准 HMI 对象	不可访问	常规操作员交互通过 FactoryTalk View 应用程序中的报警摘要、报警显示条、报警状态浏览器对象进行。此交互类似于上面介绍的自定义 HMI 选项，但不能通过程序可视化或交互。

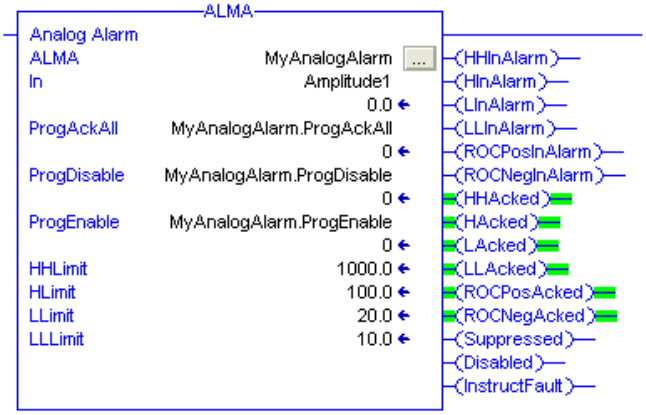
创建报警指令时，必须为该报警创建并分配具有正确报警数据类型的标签。例如，创建数据类型为 ALARM_DIGITAL 的 MyDigitalAlarm。在梯形图中，必须在指令中输入以下指令参数：

- ProgAck
- ProgReset
- ProgDisable
- ProgEnable

在梯形图和结构化文本中，给指令参数 (例如， ProgAck) 分配的值或标签在每次扫描指令时自动写入到报警标签成员 (例如， MyAnalogAlarm.ProgAck) 中。

在梯形图和结构化文本中，如果要通过程序访问报警结构，则给指令参数分配结构标签。例如，要在逻辑中使用

MyAnalogAlarm.ProgAck，则给 ProgAck 参数分配标签 MyAnalogAlarm.ProgAck。



抑制或禁止报警

抑制报警可从 HMI 中清除用户已知其存在的报警，但该报警仍保持激活状态。这样，如果您解决已知报警时无需继续查看报警信息，这可清理报警摘要。受抑制的报警不会出现在操作员摘要或显示条画面中，但受抑制的报警仍将发送到订阅者，记录到历史数据库中，可以使报警状态发生转变，具有时间戳，并且可对其它程序或操作员交互进行响应。

- 报警受到抑制时，其将继续正常工作，继续监视 In 参数是否满足报警条件，并继续对确认请求进行响应。向所有订阅者通知此事件，而且在报警处于受抑制状态时生成的所有报警信息都包括已抑制状态。报警客户端可以不同的方式来响应受抑制的报警。例如，可将受抑制的报警记录到历数数据库，但不报告给操作员。
- 报警解除抑制时，将对所有订阅者进行通知，而且提供给订阅者的报警信息将不再包括已抑制状态。

如果禁止报警，则可以像控制程序中没有报警那样来处理报警。禁止的报警不会使报警状态发生转变，也不会记录到历史数据库中。在 FactoryTalk View SE 软件的报警状态浏览器中，仍跟踪禁止的报警并且可重新使能它。

- 报警禁止时，其所有条件都将设置为初始状态 (InAlarm 清零，Acked 置位)。不监视 In 参数是否满足报警条件。向所有订阅者通知此事件。
- 报警使能时，将开始监视 In 参数是否满足报警条件。向所有订阅者通知此事件。

基于控制器的报警执行

基于控制器的报警处理来自两个来源的输入。

来源	说明
报警标签成员	多数情况下，用户应用程序扫描报警指令时处理报警标签成员。这包括： • 处理对组态参数进行的更改。 • 评估报警条件。 • 为 MinDuration 测量已经历的时间。 • 捕捉 InAlarmTime 和 RetToNormalTime 时间戳。 • 捕捉关联标签值。 • 处理 Prog 和 Oper 命令。 此外，以下报警标签状态成员在报警信息传送到各个订阅者时也会更新，与程序扫描异步发生： • DeliveryEN、 DeliveryER、 DeliveryDN • NoSubscriber、 NoConnection、 CommError、 AlarmBuffered、 SubscNotified
客户端信息	客户端信息在接收时进行处理，与程序扫描异步进行。 • 来自 RSLogix 5000 终端的复位、确认、禁止 / 使能和抑制 / 解除抑制命令 • 来自 FactoryTalk View SE 报警订阅者的复位、确认、禁止 / 使能和抑制 / 解除抑制命令

确定在应用程序中放置报警指令的位置时应小心。报警条件更改状态后扫描指令的速度影响时间戳的精确度。 MinDuration 时间累计和变化率计算需要重复扫描，扫描操作在由用户应用程序决定的时间间隔内执行。报警条件变为假后必须继续扫描报警指令，以便可以检测出 ReturnToNormal 转换。例如，如果希望时间戳的精度为 10 ms，则可将需要该精度的报警指令置于周期为 10 ms 的周期性任务中。

控制器存储器使用

作为指导准则，可使用以下报警大小粗略计算控制器存储器的使用情况：

- 一个不带关联标签的数字报警通常需要 1 KB

数字报警示例	近似大小
不带关联标签并采用以下组态的数字报警： • 报警信息：接触器故障 • 报警类别：油罐区 A	1012 字节
带两个关联标签并采用以下组态的数字报警： • 报警信息：接触器故障 • 报警类别：油罐区 A • 关联标签 1 = DINT 数据类型 • 关联标签 2 = DINT 数据类型	1100 字节
带两个关联标签并采用以下组态的数字报警： • 报警信息：接触器故障 • 报警类别：油罐区 A • 关联标签 1 = DINT 数据类型 • 关联标签 2 = STRING 数据类型	1522 字节

- 一个不带关联标签的模拟报警通常需要 2.2 KB

模拟报警示例	近似大小
不带关联标签并采用以下组态的模拟报警： • HH 报警信息：液位报警 • H 报警信息：液位报警 • L 报警信息：液位报警 • LL 报警信息：液位报警 • 正变化率信息：填充过快 • 负变化率信息：排空过快 • 报警类别：油罐区 A	2228 字节
带两个关联标签并采用以下组态的模拟报警： • HH 报警信息：液位报警 • H 报警信息：液位报警 • L 报警信息：液位报警 • LL 报警信息：液位报警 • 正变化率信息：填充过快 • 负变化率信息：排空过快 • 报警类别：油罐区 A • 关联标签 1 = DINT 数据类型 • 关联标签 2 = DINT 数据类型	2604 字节
带两个关联标签并采用以下组态的模拟报警： • HH 报警信息：液位报警 • H 报警信息：液位报警 • L 报警信息：液位报警 • LL 报警信息：液位报警 • 正变化率信息：填充过快 • 负变化率信息：排空过快 • 报警类别：油罐区 A • 关联标签 1 = DINT 数据类型 • 关联标签 2 = STRING 数据类型	4536 字节

更长的信息字符串和多语言信息字符串将占用更多的控制器存储空间。

实际存储器使用情况取决于报警的组态、信息长度和随报警一起传送的任何关联标签。

扫描时间

以下执行时间说明了 ALMD 指令和 ALMA 指令对总的扫描时间的影响情况。

梯级状态		执行时间	
		数字报警 (ALMD)	模拟报警 (ALMA)
无报警状态变化	梯级为假	8 µs	17 µs
	梯级为真	8 µs	60 µs
有报警状态变化	梯级为假	35 µs	17 µs
	梯级为真	35 µs	126 µs

报警状态变化是指任何事件 (例如，确认或抑制报警) 使报警状态发生改变。通过对相关报警创建依存关系最大限度降低大量报警同时更改状态的可能性 (报警突增)。出现大量报警会严重影响应用项目代码扫描时间。

注：

位指令
(XIC、 XIO、 OTE、 OTL、 OTU、 ONS、 OSR、 OSF、 OSRI、 OSFI)

简介

使用位 (继电器型) 指令可监控位的状态。

如果要	使用以下指令	在以下语言中可用	页码
当相关位置位时使能输出	XIC	梯形图 结构化文本(1)	78
当相关位清零时使能输出	XIO	梯形图 结构化文本 (1)	81
置位某位	OTE	梯形图 结构化文本 (1)	84
置位某位 (可保持)	OTL	梯形图 结构化文本 (1)	86
清零某位 (可保持)	OTU	梯形图 结构化文本 (1)	88
每次梯级变为真时，使能输出一次扫描的时间	ONS	梯形图 结构化文本 (1)	90
每次梯级变为真时，置位某位一次扫描的时间	OSR	梯形图	93
每次梯级变为假时，置位某位一次扫描的时间	OSF	梯形图	96
每次在功能块中置位输入位时，置位某位一次扫描的时间	OSRI	结构化文本 功能块	98
每次在功能块中清零输入位时，置位某位一次扫描的时间	OSFI	结构化文本 功能块	101

(1) 没有对等的结构化文本指令。使用其它结构化文本编程可获得相同结果。请参见指令说明。

检查是否闭合指令 (XIC)

XIC 指令检查数据位是否置位。

操作数：

梯形图

操作数	类型	格式	说明
数据位	BOOL	标签	要测试的位

结构化文本

结构化文本没有 XIC 指令，但可通过 IF...THEN 结构获得相同结果。

```
IF data_bit THEN
    <statement>;
END_IF;
```

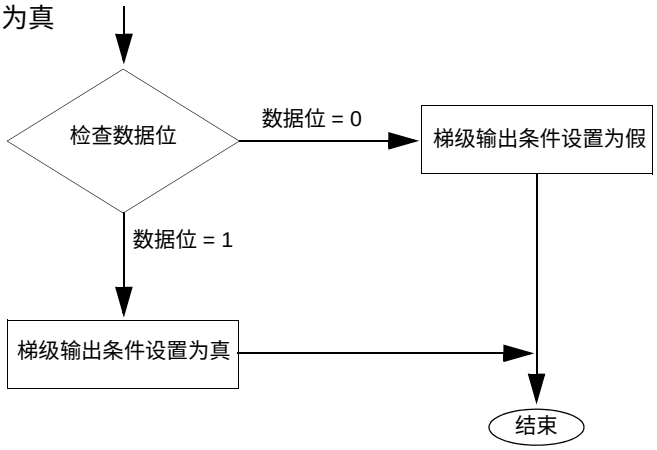
有关结构化文本中结构语法的信息，请参见功能块属性。

说明：　XIC 指令检查数据位是否置位。

算术状态标志：　不受影响

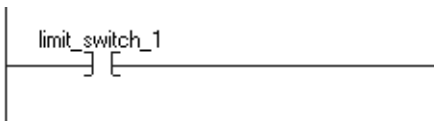
故障条件：　无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

示例 **1**： 如果 *limit_switch_1* 置位，则使能下一条指令 (梯级输出条件为真)。

梯形图



结构化文本

```
IF limit_switch THEN

    <statement>;

END_IF;
```

示例 **2**： 如果 S:V 置位 (表示溢出已发生)，则使能下一条指令 (梯级输出条件为真)。

梯形图

结构化文本

```
IF S:V THEN
    <statement>;
END_IF;
```

检查是否断开指令 (XIO)

XIO 指令检查数据位是否清零。

操作数：

梯形图

操作数	类型	格式	说明
数据位	BOOL	标签	要测试的位

结构化文本

结构化文本没有 XIO 指令，但可通过 IF...THEN 结构获得相同结果。

```
IF NOT data_bit THEN
    <statement>;
END_IF;
```

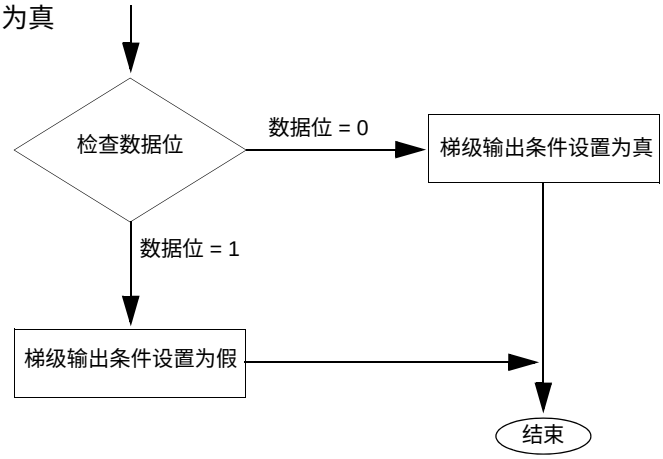
有关结构化文本中结构语法的信息，请参见功能块属性。

说明： XIO 指令检查数据位是否清零。

算术状态标志： 不受影响

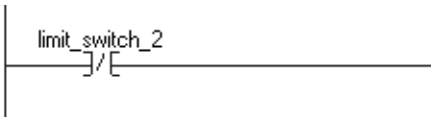
故障条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

示例 **1**： 如果 *limit_switch_2* 清零，则使能下一条指令 (梯级输出条件为真)。

梯形图

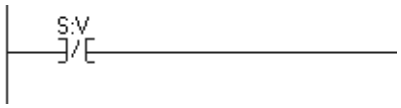


结构化文本

```
IF NOT limit_switch_2 THEN

    <statement>;

END_IF;
```

示例 **2**： 如果 S:V 清零 (表示没有溢出发生)，则使能下一条指令 (梯级输出条件为真)。

梯形图



结构化文本

```
IF NOT S:V THEN
    <statement>;
END_IF;
```

输出激活指令 (OTE)

OTE 指令置位或清零数据位。

操作数：

梯形图

操作数	类型	格式	说明
数据位	BOOL	标签	要置位或清零的位

结构化文本

结构化文本没有 OTE 指令，但可通过非保持型赋值方法获得相同结果。

```
data_bit [:=] BOOL_expression;
```

有关结构化文本中赋值和表达式语法的信息，请参见功能块属性。

说明：　使能 OTE 指令时，控制器置位数据位。禁止 OTE 指令时，控制器清零数据位。

算术状态标志：　不受影响

故障条件：　无

执行：

条件	梯形图操作
预扫描	清零数据位。 梯级输出条件设置为假。
梯级输入条件为假	清零数据位。 梯级输出条件设置为假。
梯级输入条件为真	置位数据位。 梯级输出条件设置为真。
后扫描	清零数据位。 梯级输出条件设置为假。

示例： switch 置位时，OTE 指令置位 (接通)light_1。*switch* 清零时，OTE 指令清零 (断开)light_1。

梯形图

结构化文本

```
light_1 [:=] switch;
```

输出锁存指令 (OTL)

OTL 指令置位 (锁存) 数据位。

操作数：

梯形图

操作数	类型	格式	说明
数据位	BOOL	标签	要置位的位

结构化文本

结构化文本没有 OTL 指令，但可通过 IF...THEN 结构和赋值获得相同结果。

```
IF BOOL_expression THEN

    data_bit := 1;

END_IF;
```

有关结构化文本中结构、表达式和赋值语法的信息，请参见功能块属性。

说明： 使能 OTL 指令时， OTL 指令置位数据位。数据位保持置位状态直到清零为止，通常由 OTU 指令清除。禁止 OTL 指令时，OTL 指令不会改变数据位的状态。

算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作
预扫描	不修改数据位。 梯级输出条件设置为假。
梯级输入条件为假	不修改数据位。 梯级输出条件设置为假。
梯级输入条件为真	置位数据位。 梯级输出条件设置为真。
后扫描	不修改数据位。 梯级输出条件设置为假。

示例： 使能 OTL 指令时， OTL 指令置位 *light_2*。该位保持置位状态直到清零为止，通常由 OTU 指令清除。

梯形图

结构化文本

```
IF BOOL_expression THEN

    light_2 := 1;

END_IF;
```

输出解锁 (OTU)

OUT 指令清零 (解锁) 数据位。

操作数：

梯形图

操作数	类型	格式	说明
数据位	BOOL	标签	要清零的位

结构化文本

结构化文本没有 OTU 指令，但可通过 IF...THEN 结构和赋值获得相同结果。

```
IF BOOL_expression THEN

    data_bit := 0;

END_IF;
```

有关结构化文本中结构、表达式和赋值语法的信息，请参见功能块属性。

说明：使能 OTU 指令时，OTU 指令清零数据位。禁止 OTU 指令时，OTU 指令不会改变数据位的状态。

算术状态标志：不受影响

故障条件：无

执行：

条件	梯形图操作
预扫描	不修改数据位。 梯级输出条件设置为假。
梯级输入条件为假	不修改数据位。 梯级输出条件设置为假。
梯级输入条件为真	清零数据位。 梯级输出条件设置为真。
后扫描	不修改数据位。 梯级输出条件设置为假。

示例： 使能 OTU 指令时， OTU 指令清零 *light_2*。

梯形图

结构化文本

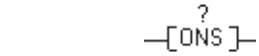
```
IF BOOL_expression THEN
    light_2 := 0;
END_IF;
```

单脉冲触发 (ONS)

ONS 指令根据存储位的状态使能或禁止梯级的其余部分。

操作数：

梯形图



操作数	类型	格式	说明
存储位	BOOL	标签	内部存储位。 存储上次执行指令时的梯级输入条件

结构化文本

结构化文本没有 ONS 指令，但可通过 IF...THEN 结构获得相同结果。

```
IF BOOL_expression AND NOT storage_bit THEN
    <statement>;
END_IF;
storage_bit := BOOL_expression;
```

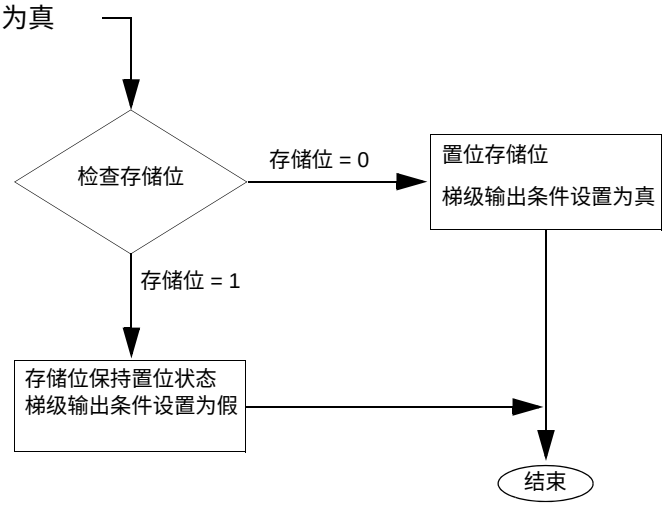
有关结构化文本中结构、表达式和赋值语法的信息，请参见功能块属性。

说明：当 ONS 指令使能并且存储位清零时， ONS 指令使能梯级的其余部分。当 ONS 指令禁止并且存储位置位时， ONS 指令禁止梯级的其余部分。

算术状态标志：不受影响

故障条件：无

执行：

条件	梯形图操作
预扫描	置位存储位，防止第一次扫描期间发生无效触发。 梯级输出条件设置为假。
梯级输入条件为假	清零存储位。 梯级输出条件设置为假。
梯级输入条件为真	
后扫描	清零存储位。 梯级输出条件设置为假。

示例： 典型的用法是在 ONS 指令前面有一条输入指令，它使能时和禁止时，扫描 ONS 指令才会有正确的操作。一旦 ONS 指令被使能，必须梯级输入条件清零或存储位清零， ONS 指令才可以再次使能。

如果扫描到 *limit_switch_1* 清零或者 *storage_1* 置位，则此梯级没有任何作用。如果扫描到 *limit_switch_1* 置位且 *storage_1* 清零，ONS 指令将置位 storage_1， ADD 指令将使 sum 加 1。只要 *limit_switch_1* 保持置位状态， sum 就维持该值不变。只有 *limit_switch_1* 再次从清零状态转变为置位状态，才能再次对 sum 执行一次加法操作。

梯形图

结构化文本

```
IF limit_switch_1 AND NOT storage_1 THEN

    sum := sum + 1;

END_IF;

storage_1 := limit_switch_1;
```

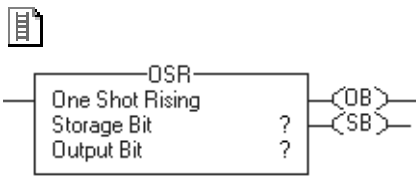
上升沿单脉冲触发 (OSR)

OSR 指令根据存储位的状态置位或清零输出位。

该指令在结构化文本和功能块中以 OSRI 形式提供，请参见第 98 页。.

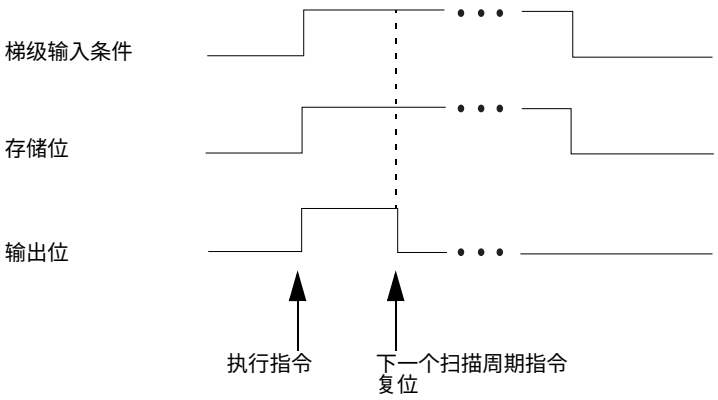
操作数：

梯形图



操作数	类型	格式	说明
存储位	BOOL	标签	内部存储位 存储上次执行指令时的梯级输入条件
输出位	BOOL	标签	要置位的位

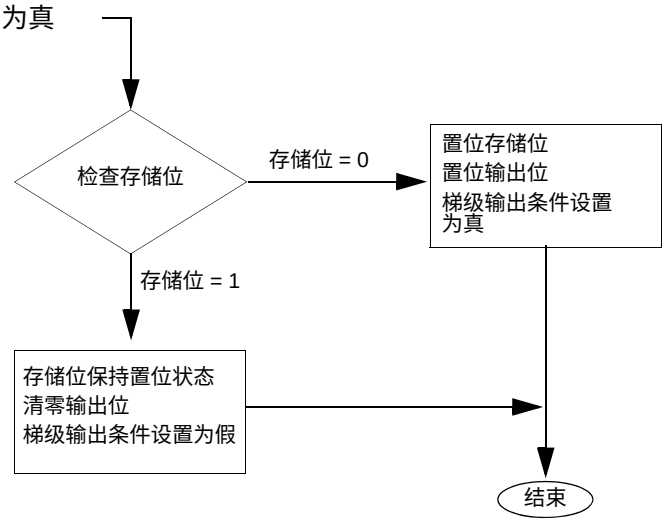
说明： OSR 指令使能并且存储位清零时，OSR 指令将置位输出位。OSR 指令使能并且存储位置位时，或者禁止时， OSR 指令将清零输出位



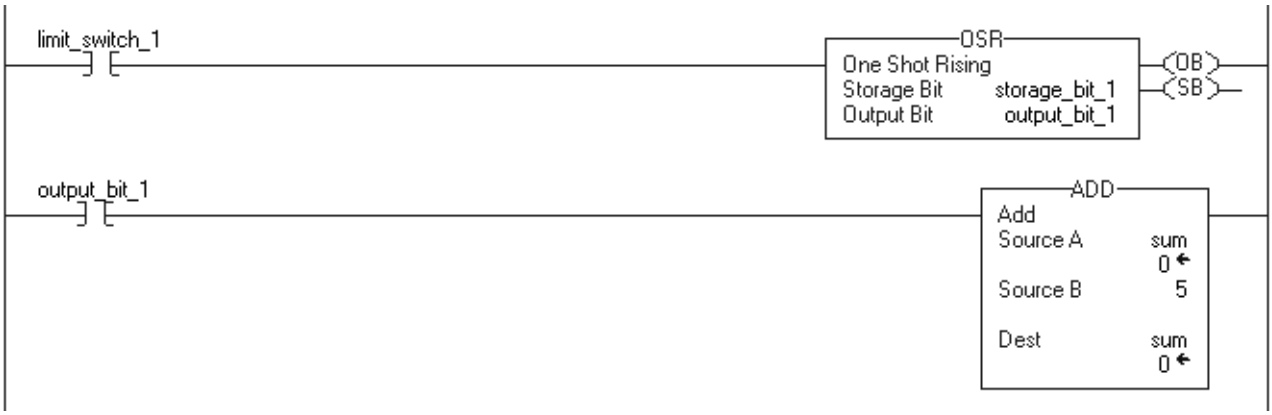
算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作
预扫描	置位存储位，防止第一次扫描期间发生无效触发。 清零输出位。 梯级输出条件设置为假。
梯级输入条件为假	清零存储位。 不修改输出位。 梯级输出条件设置为假。
梯级输入条件为真	
后扫描	清零存储位。 不修改输出位。 梯级输出条件设置为假。

示例： 每次 *limit_switch_1* 从清零状态转变为置位状态时，OSR 指令都将置位 *output_bit_1*，ADD 指令将使 sum 加 5。只要 *limit_switch_1* 保持置位状态， sum 就维持该值不变。只有 *limit_switch_1* 再次从清零状态转变为置位状态，才能再次对 sum 执行加法操作。可以在多个梯级中使用 *output_bit_1* 来触发其它操作

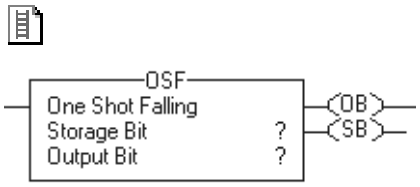


下降沿单脉冲触发 (OSF)

OSF 指令根据存储位状态置位或清零输出位。

该指令在结构化文本和功能块中以 OSFI 形式提供，请参见第 101 页。

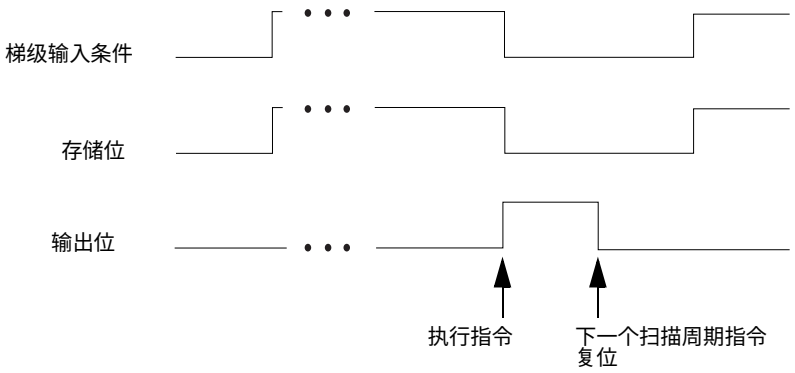
操作数：



梯形图操作数

操作数	类型	格式	说明
存储位	BOOL	标签	内部存储位 存储上次执行指令时的梯级输入条件
输出位	BOOL	标签	要置位的位

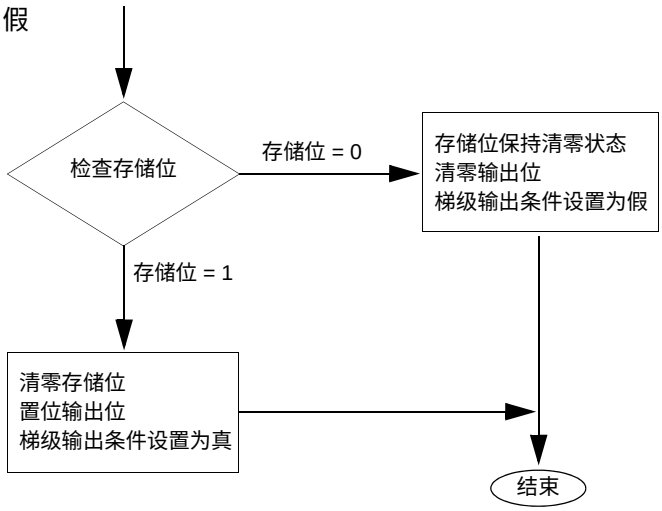
说明：OSF 指令禁止并且存储位置位时，OSF 指令将置位输出位。OSF 指令禁止并且存储位清零时，或者使能时， OSF 指令将清零输出位。



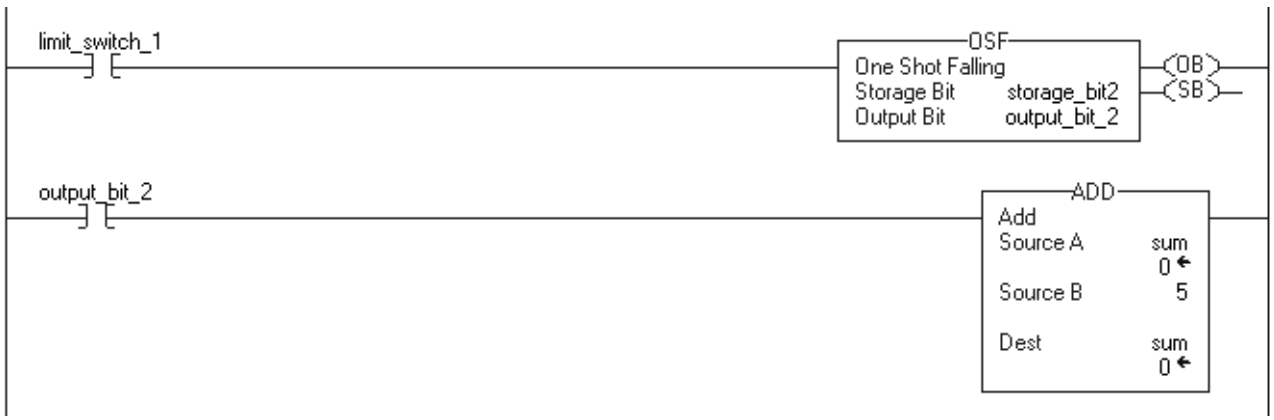
算术状态标志：不受影响

故障条件：无

执行：

条件	梯形图操作
预扫描	清零存储位，防止第一次扫描期间发生无效触发。 清零输出位。 梯级输出条件设置为假。
梯级输入条件为假	
梯级输入条件为真	置位存储位。 清零输出位。 梯级输出条件设置为真。
后扫描	请参见上面的梯级输入条件为假。

示例： 每次 *limit_switch_1* 从置位状态转变为清零状态时，OSF 指令将置位 *output_bit_2*， ADD 指令将使 sum 加 5。只要 *limit_switch_1* 保持清零状态， sum 就维持该值不变。只有 *limit_switch_1* 再次从置位状态转变为清零状态，才能再次对 sum 执行加法操作。可以在多个梯级中使用 *output_bit_2* 来触发其它操作



带输入的上升沿单脉冲触发 (OSRI)

当输入位从清零状态转变为置位状态时， OSRI 指令将输出位置位一个执行周期的时间。

该指令在梯形图中以 OSR 形式提供，请参见第 93 页。

操作数：

结构化文本

```
OSRI(OSRI_tag);
```

操作数	类型	格式	说明
OSRI 标签	FBD_ONESHOT	结构	OSRI 结构

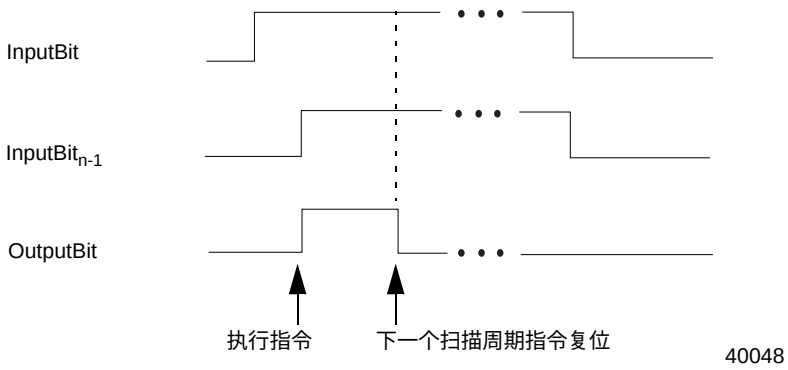
功能块

操作数	类型	格式	说明
OSRI 标签	FBD_ONESHOT	结构	OSRI 结构

FBD_ONESHOT 结构

输入参数	数据类型	说明
EnableIn	BOOL	功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。执行该指令。
InputBit	BOOL	输入位。这等效于梯形图 OSR 指令的梯级条件。 默认为清零状态。
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
OutputBit	BOOL	输出位

说明： 当 InputBit 置位并且 $InputBit_{n-1}$ 清零时， OSRI 指令将置位 OutputBit。当 $InputBit_{n-1}$ 置位或 InputBit 清零时，OSRI 指令将清零 OutputBit。 .



算术状态标志： 不受影响

故障条件： 无

执行：

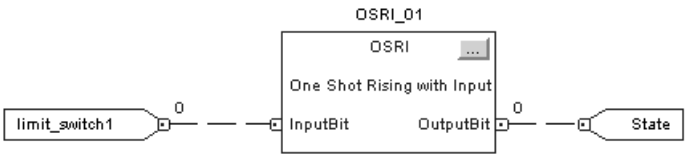
条件	功能块操作	结构化文本操作
预扫描	不执行任何操作。	不执行任何操作。
指令第一次扫描	置位 InputBit $_{n-1}$。	置位 InputBit $_{n-1}$。
指令第一次运行	置位 InputBit $_{n-1}$。	置位 InputBit $_{n-1}$。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	无
EnableIn 处于置位状态	InputBit 从清零转变为置位状态时，指令将置位 InputBit $_{n-1}$。 执行该指令。 置位 EnableOut。	InputBit 从清零转变为置位状态时，指令将置位 InputBit $_{n-1}$。 EnableIn 始终置位。 执行该指令。
后扫描	不执行任何操作。	不执行任何操作。

示例： 当 *limit_switch1* 从清零状态转变为置位状态时， OSRI 指令将 OutputBit 置位一次扫描的时间。

结构化文本

```
OSRI_01.InputBit := limit_switch1;


OSRI(OSRI_01);


State := OSRI_01.OutputBit;
```

功能块



带输入的下降沿单脉冲触发 (OSFI)

当 InputBit 从置位状态转变为清零状态时，OSFI 指令将 OutputBit 置位一个执行周期的时间。

该指令在梯形图中以 OSF 形式提供，请参见第 96 页。

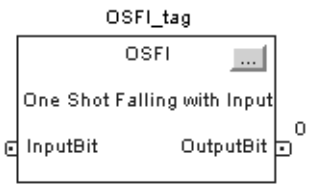
操作数：

结构化文本

```
OSFI(OSFI_tag);
```

操作数	类型	格式	说明
OSFI 标签	FBD_ONESHOT	结构	OSFI 结构

功能块

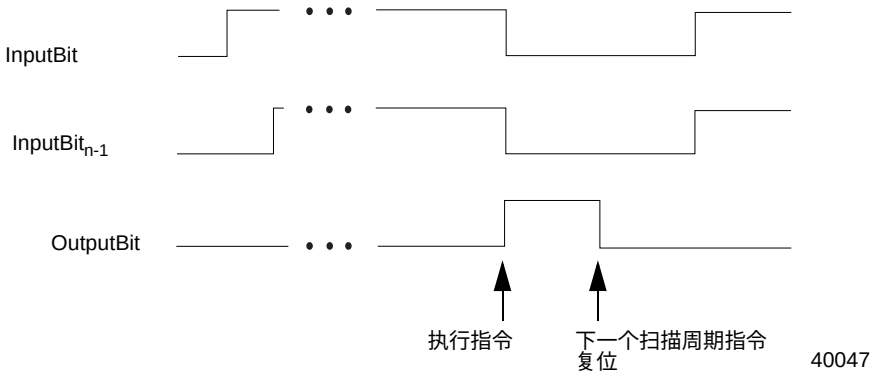


操作数	类型	格式	说明
OSFI 标签	FBD_ONESHOT	结构	OSFI 结构

FBD_ONESHOT 结构

输入参数	数据类型	说明
EnableIn	BOOL	功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。执行该指令。
InputBit	BOOL	输入位。这等效于梯形图 OSF 指令的梯级条件 默认为清零状态。
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
OutputBit	BOOL	输出位

说明： 当 InputBit 清零并且 InputBit $_{n-1}$ 置位时， OSFI 指令将置位 OutputBit。
当 InputBit $_{n-1}$ 清零或者 InputBit 置位时， OSFI 指令将清零 OutputBit。



算术状态标志： 不受影响

故障条件： 无

执行：

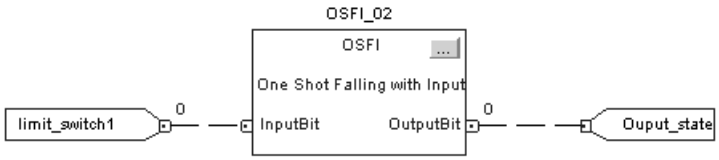
条件	功能块操作	结构化文本操作
预扫描	不执行任何操作。	不执行任何操作。
指令第一次扫描	清零 InputBit $_{n-1}$。	清零 InputBit $_{n-1}$。
指令第一次运行	清零 InputBit $_{n-1}$。	清零 InputBit $_{n-1}$。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	无
EnableIn 处于置位状态	InputBit 从清零转变为置位状态时，指令将清零 InputBit $_{n-1}$。 执行该指令。 置位 EnableOut。	InputBit 从清零转变为置位状态时，指令将清零 InputBit $_{n-1}$。 EnableIn 始终置位。 执行该指令。
后扫描	不执行任何操作。	不执行任何操作。

示例： 当 *limit_switch1* 从置位状态转变为清零状态时， OSFI 指令将 OutputBit 置位一次扫描的时间。

结构化文本

```
OSFI_01.InputBit := limit_switch1;


OSFI(OSFI_01);


Output_state := OSFI_01.OutputBit;
```

功能块



注：

计时器和计数器指令 (TON、 TOF、 RTO、 TONR、 TOFR、 RTOR、 CTU、 CTD、 CTUD、 RES)

简介

计时器和计数器根据时间或事件数控制操作。

如果要	使用以下指令	在以下语言中可用	页码
对计时器的使能时间进行计时	TON	梯形图	106
对计时器的禁止时间进行计时	TOF	梯形图	110
累计时间	RTO	梯形图	114
对功能块中具有内置复位功能的计时器的使能时间进行计时	TONR	结构化文本 功能块	118
对功能块中具有内置复位功能的计时器的禁止时间进行计时	TOFR	结构化文本 功能块	122
功能块中具有内置复位的情况下累计时间	RTOR	结构化文本 功能块	126
增计数	CTU	梯形图	130
减计数	CTD	梯形图	134
在功能块中增计数和减计数	CTUD	结构化文本 功能块	138
复位计时器或计数器	RES	梯形图	143

所有计时器的时基均为 1 ms。

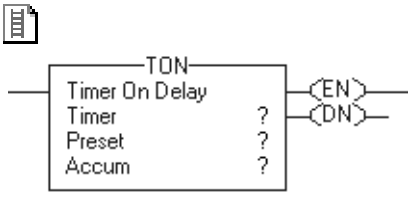
接通延时计时器 (TON)

TON 指令是非保持型计时器，当指令使能 (梯级输入条件为真) 时累计时间。

该指令在结构化文本和功能块中以 TONR 形式提供。

操作数：

梯形图



操作数	类型	格式	说明
计时器 (Timer)	TIMER	标签	计时器结构
预置值 (Preset)	DINT	立即数 标签	延时时间 (累计时间)
Accum	DINT	立即数 标签	计时器已计数的总毫秒数 初始值通常为 0

TIMER 结构

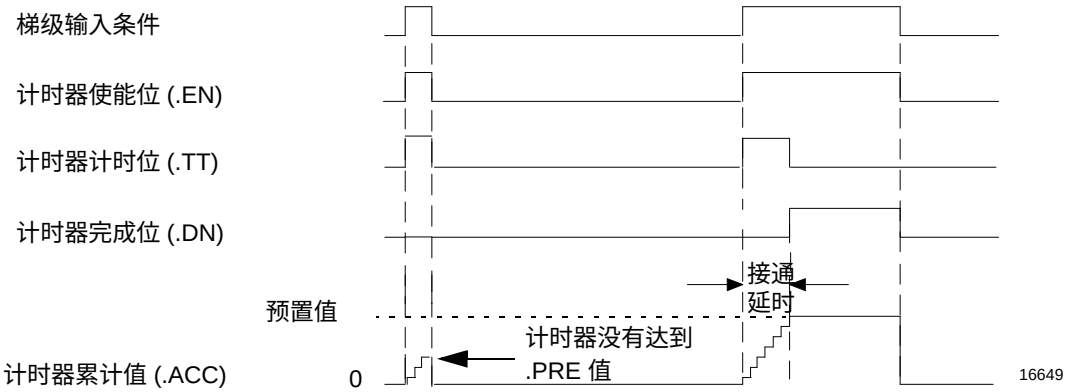
助记符	数据类型	说明
.EN	BOOL	使能位指示 TON 指令是否使能。
.TT	BOOL	计时位指示是否正在执行计时操作。
.DN	BOOL	当 .ACC ≥ .PRE 时完成位置位。
.PRE	DINT	预置值指定在指令置位 .DN 位前，累计值必须达到的值 (以 1 毫秒为单位)。
.ACC	DINT	累计值指明从 TON 指令使能以来所累计的毫秒数。

说明： TON 指令一直累计时间，直到：

- 禁止 TON 指令。
- .ACC ≥ .PRE。

时基始终是 1 ms。例如，对于两秒计时器，输入 2000 作为 .PRE 值。

当禁止 TON 指令时，清零 .ACC 值。



计时器的运行方式是从当前时间中减去其上次扫描的时间：

ACC = ACC + (*current_time - last_time_scanned*)

更新 ACC 后，计时器将 last_time_scanned 设置为 current_time。这样，计时器即为下次扫描做好准备。

重要事项

确保在计时器运行时至少每 69 分钟扫描一次计时器。否则， ACC 值将不正确。

last_time_scanned 值的最大范围是 69 分钟。如果没有在 69 分钟内扫描计时器，计时器的计算将溢出。 如果发生此情况， ACC 值是不正确的。

如果将计时器放置在以下位置，则当计时器运行时，请在 69 分钟内对其进行扫描：

- 子例程。
- JMP 和 LBL 指令间的代码段。
- 顺序功能图 (SFC)。
- 事件型任务或周期型任务。
- 阶段的状态例程。

算术状态标志： 不受影响

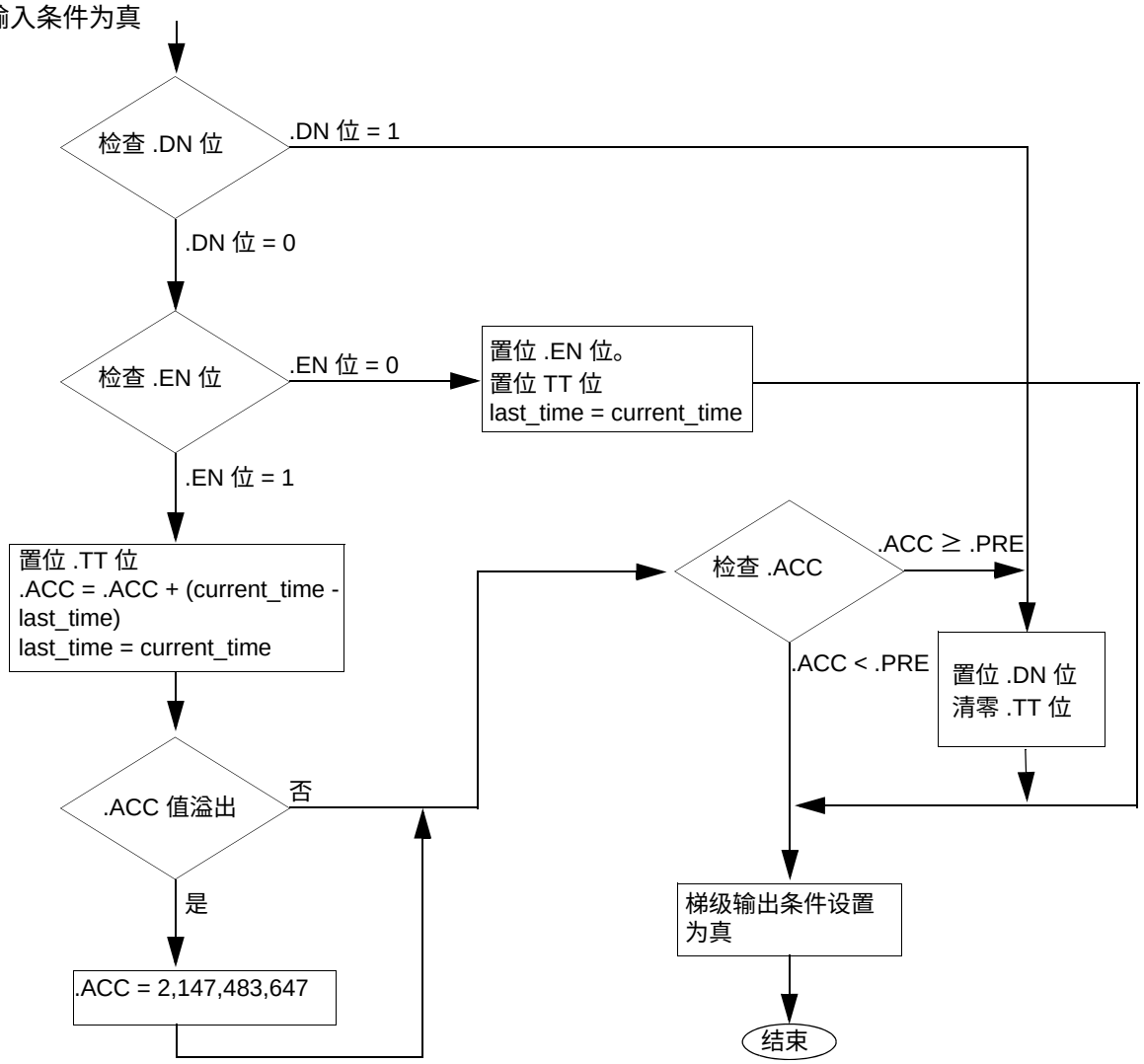
故障条件：

出现主要故障的条件	故障类型	故障代码
.PRE < 0	4	34
.ACC < 0	4	34

执行：

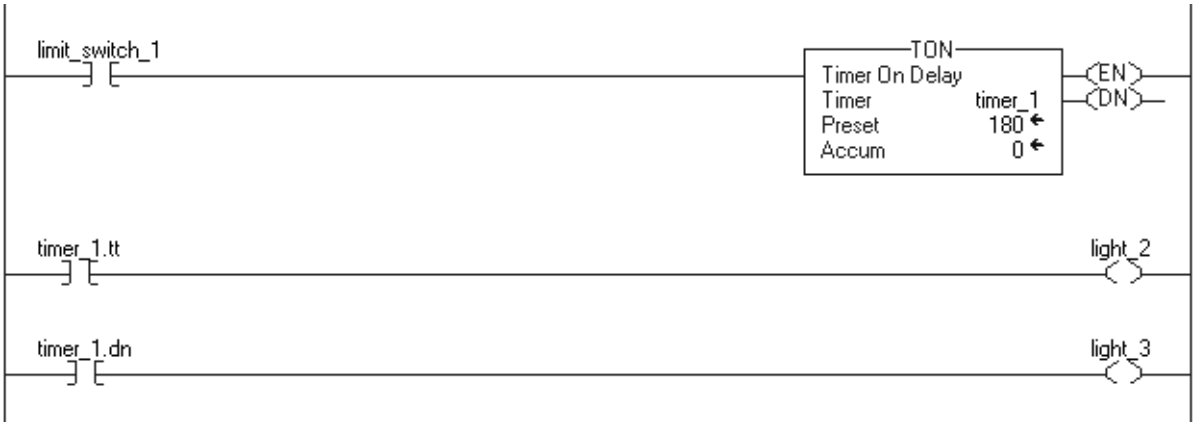
条件	梯形图操作
预扫描	清零 .EN、 .TT 和 .DN 位。 清零 .ACC 值。 梯级输出条件设置为假。
梯级输入条件为假	清零 .EN、 .TT 和 .DN 位。 清零 .ACC 值。 梯级输出条件设置为假。

梯级输入条件为真



后扫描	梯级输出条件设置为假。

示例： 当 *limit_switch_1* 置位时， *light_2* 接通 180 ms(timer_1 正在计时)。当 *timer_1.acc* 值达到 180 时，*light_2* 熄灭，*light_3* 点亮。*Light_3* 一直点亮，直到禁止 TON 指令为止。如果在 *timer_1* 计时期间 *limit_switch_1* 清零， *light_2* 将熄灭。



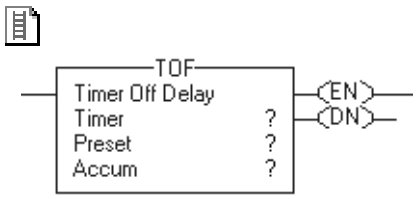
关断延时计时器 (TOF)

TOF 指令是非保持型计时器，当指令使能 (梯级输入条件为假) 时累计时间。

该指令在结构化文本和功能块中以 TOFR 形式提供。

操作数：

梯形图



操作数	类型	格式	说明
计时器 (Timer)	TIMER	标签	计时器结构
预置值 (Preset)	DINT	立即数	延时时间 (累计时间)
Accum	DINT	立即数	计时器已计数的总毫秒数 初始值通常为 0

TIMER 结构

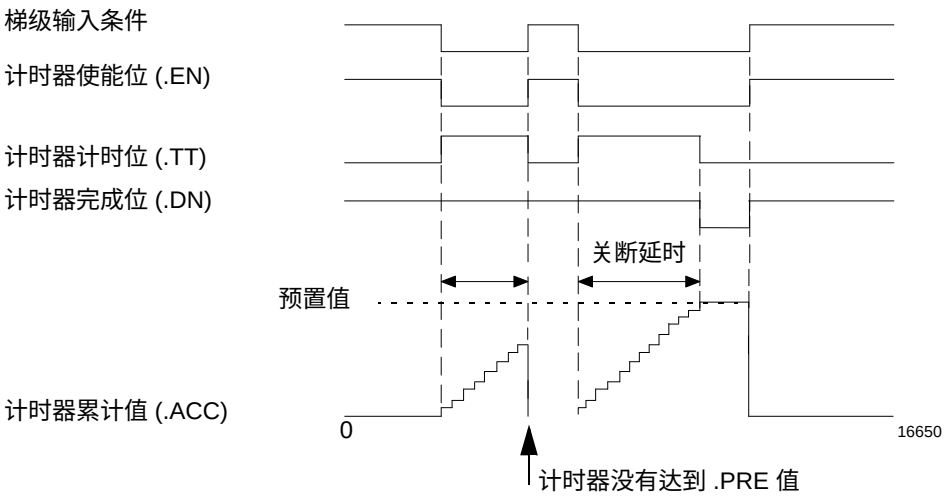
助记符	数据类型	说明
.EN	BOOL	使能位指示 TOF 指令是否使能。
.TT	BOOL	计时位指示是否正在执行计时操作
.DN	BOOL	当 .ACC ≥ .PRE 时清零完成位。
.PRE	DINT	预置值指定在指令对 .DN 位清零前，累计值必须达到的值 (以 1 毫秒为单位)。
.ACC	DINT	累计值指明从 TOF 指令使能以来所经过的毫秒数。

说明： TOF 指令一直累计时间，直到：

- 禁止 TOF 指令。
- .ACC ≥ .PRE。

时基始终是 1 ms。例如，对于两秒计时器，输入 2000 作为 .PRE 值。

当禁止 TOF 指令时，清零 .ACC 值。



计时器的运行方式是从当前时间中减去其上次扫描的时间：

ACC = ACC + (*current_time - last_time_scanned*)

更新 ACC 后，计时器将 last_time_scanned 设置为 current_time。这样，计时器即为下次扫描做好准备。

重要事项

确保在计时器运行时至少每 69 分钟扫描一次计时器。否则， ACC 值将不正确。

last_time_scanned 值的最大范围是 69 分钟。如果没有在 69 分钟内扫描计时器，计时器的计算将溢出。 如果发生此情况， ACC 值是不正确的。

如果将计时器放置在以下位置，则当计时器运行时，请在 69 分钟内对其进行扫描：

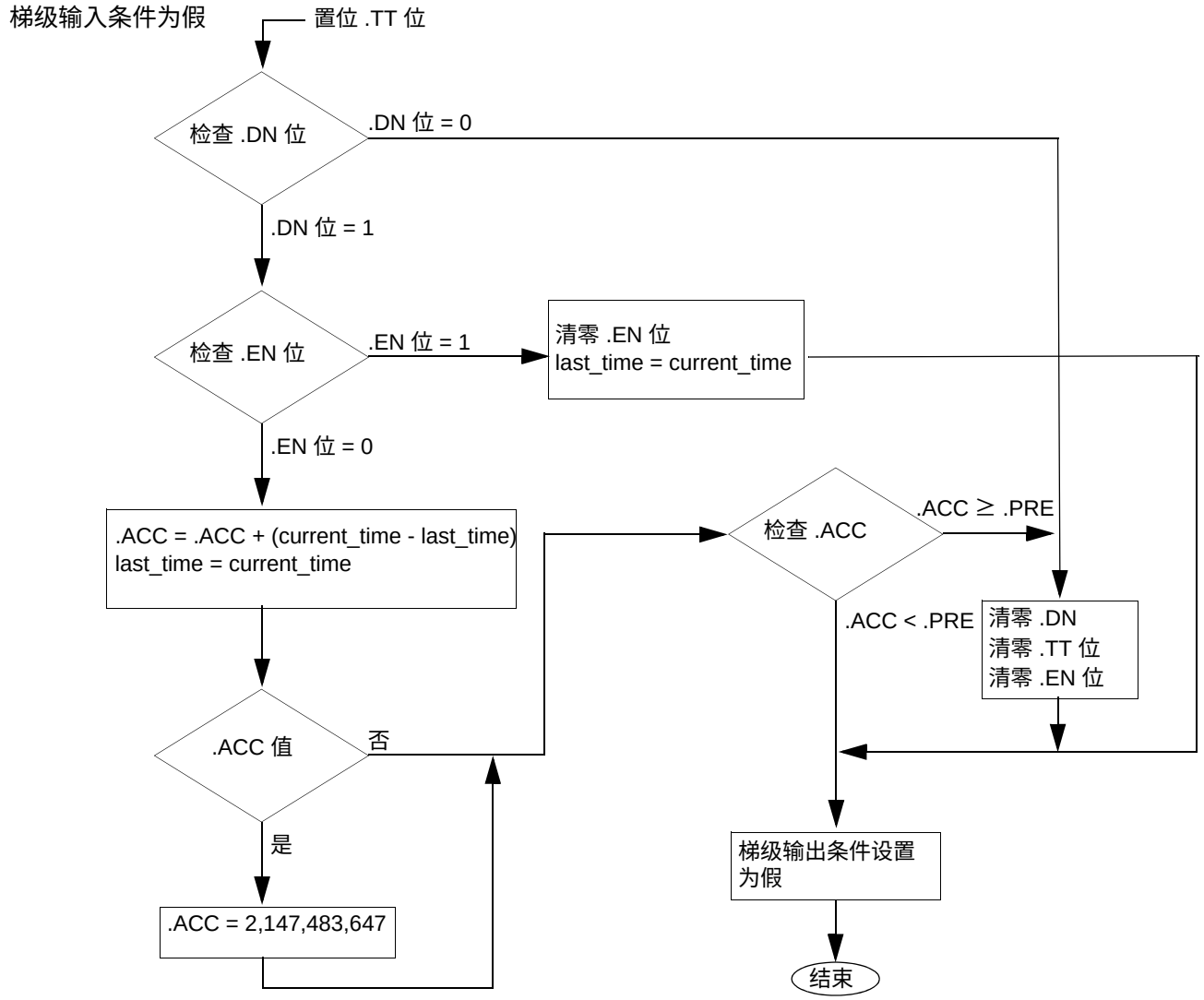
- 子例程。
- JMP 和 LBL 指令间的代码段。
- 顺序功能图 (SFC)。
- 事件型任务或周期型任务。
- 阶段的状态例程。

算术状态标志： 不受影响

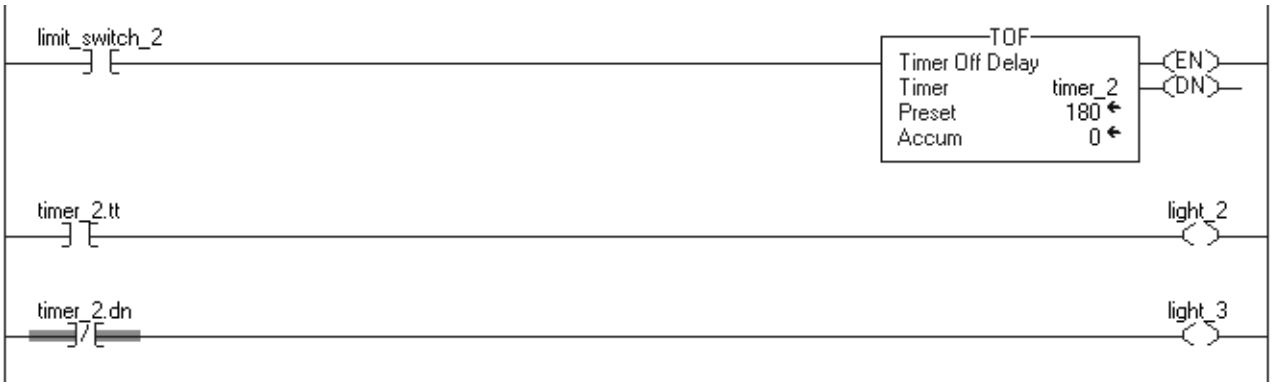
故障条件：

出现主要故障的条件	故障类型	故障代码
.PRE < 0	4	34
.ACC < 0	4	34

执行：

条件	梯形图操作
预扫描	清零 .EN、 .TT 和 .DN 位。 .ACC 值设为等于 .PRE 值。 梯级输出条件设置为假。
梯级输入条件为假	
梯级输入条件为真	置位 .EN 和 .DN 位 清零 .TT 位。 清零 .ACC 值。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

示例： 当清零 *limit_switch_2* 时， *light_2* 接通 180 ms(timer_2 正在计时)。当 *timer_2.acc* 值达到 180 时，*light_2* 熄灭，*light_3* 点亮。*Light_3* 一直点亮，直到使能 TOF 指令。如果在 *timer_2* 计时期间置位 *limit_switch_2*， *light_2* 将熄灭。



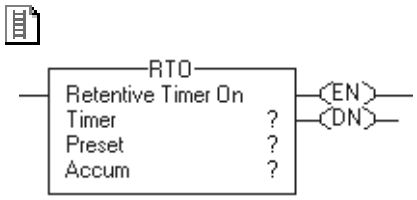
保持型接通计时器 (RTO)

RTO 指令是保持型计时器，当指令使能时累计时间。

该指令在结构化文本和功能块中以 RTOR 形式提供。

操作数：

梯形图



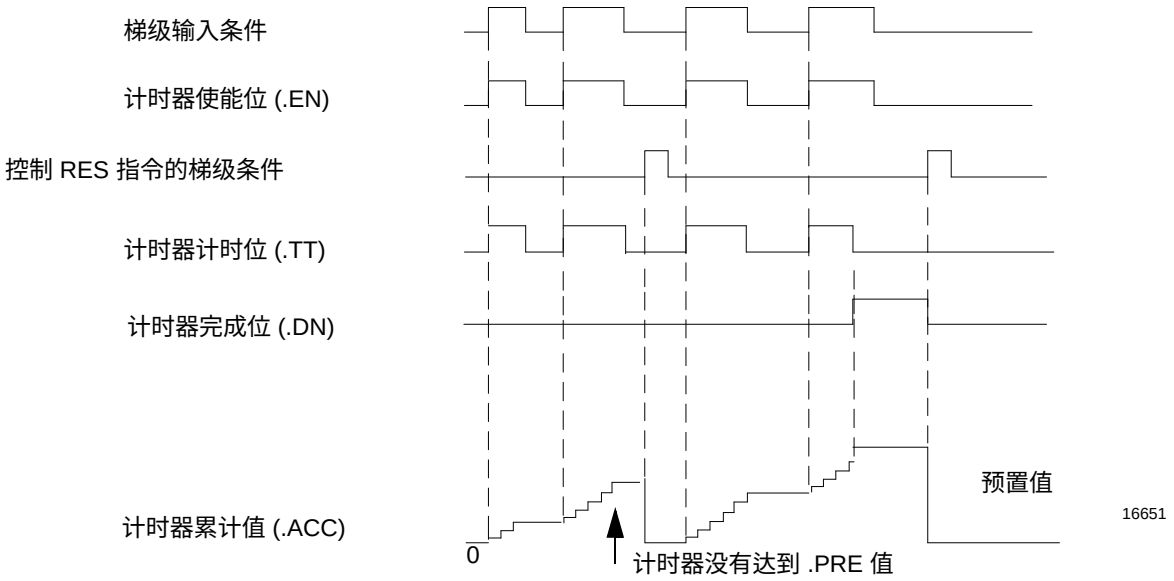
操作数	类型	格式	说明
计时器 (Timer)	TIMER	标签	计时器结构
预置值 (Preset)	DINT	立即数	延时时间 (累计时间)
Accum	DINT	立即数	计时器已计数的毫秒数 初始值通常为 0

TIMER 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 RTO 指令是否使能。
.TT	BOOL	计时位指示是否正在执行计时操作
.DN	BOOL	完成位指明 .ACC 是否大于等于 .PRE。
.PRE	DINT	预置值指定在指令置位 .DN 位前，累计值必须达到的值 (以 1 毫秒为单位)。
.ACC	DINT	累计值指明从 RTO 指令使能以来所经过的毫秒数。

说明： RTO 指令一直累计时间直到被禁止。当禁止 RTO 指令时，会保持其 .ACC 值。当必须清零 .ACC 值时，通常使用清除 TIMER 结构的 RES 指令。

时基始终是 1 ms。例如，对于两秒计时器，输入 2000 作为 .PRE 值。



计时器的运行方式是从当前时间中减去其上次扫描的时间：

```
ACC = ACC + (current_time - last_time_scanned)
```

更新 ACC 后，计时器将 last_time_scanned 设置为 current_time。这样，计时器即为下次扫描做好准备。

重要事项

确保在计时器运行时至少每 69 分钟扫描一次计时器。否则， ACC 值将不正确。

last_time_scanned 值的最大范围是 69 分钟。如果没有在 69 分钟内扫描计时器，计时器的计算将溢出。 如果发生此情况， ACC 值是不正确的。

如果将计时器放置在以下位置，则当计时器运行时，请在 69 分钟内对其进行扫描：

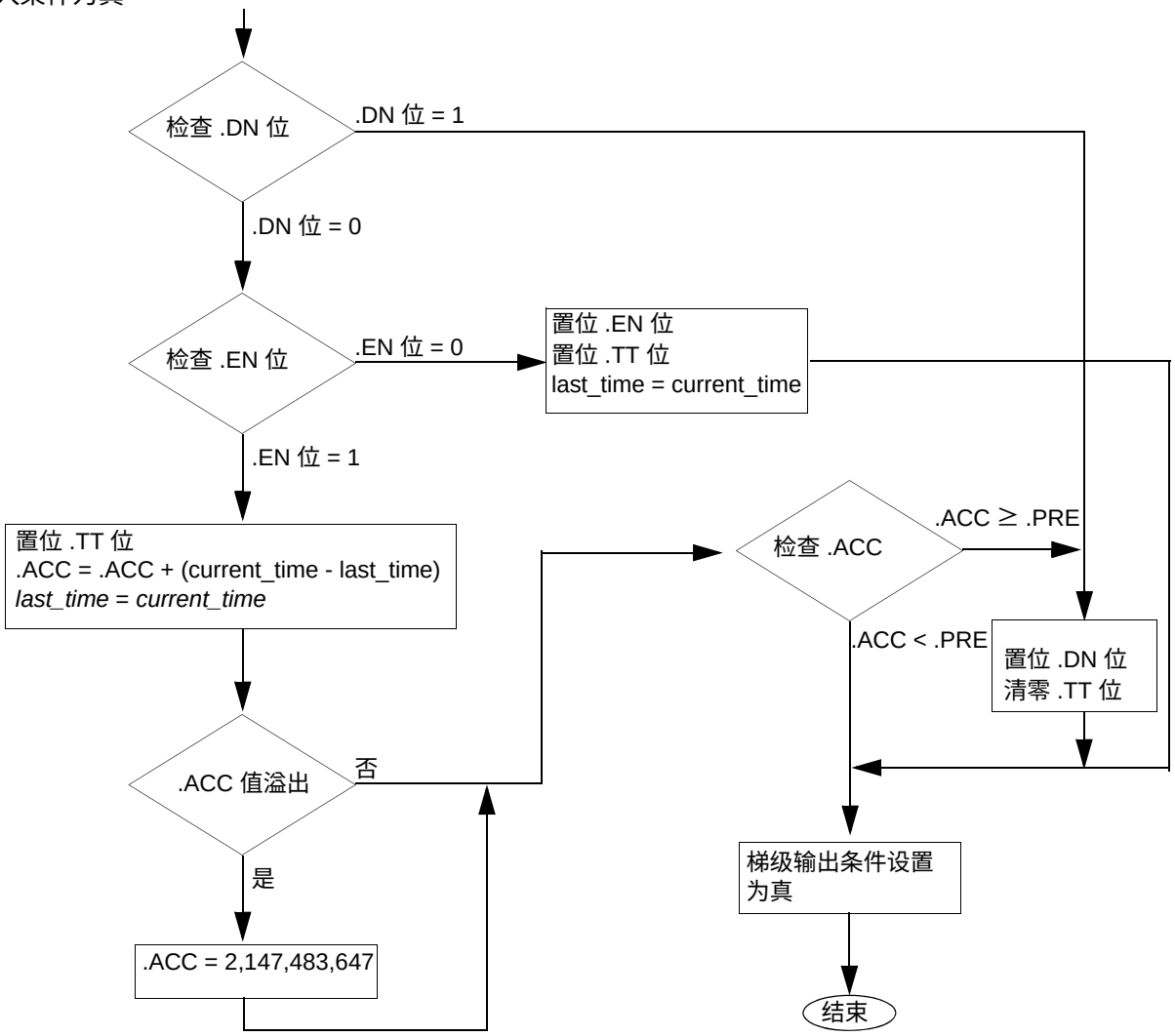
- 子例程。
- JMP 和 LBL 指令间的代码段。
- 顺序功能图 (SFC)。
- 事件型任务或周期型任务。
- 阶段的状态例程。

算术状态标志： 不受影响

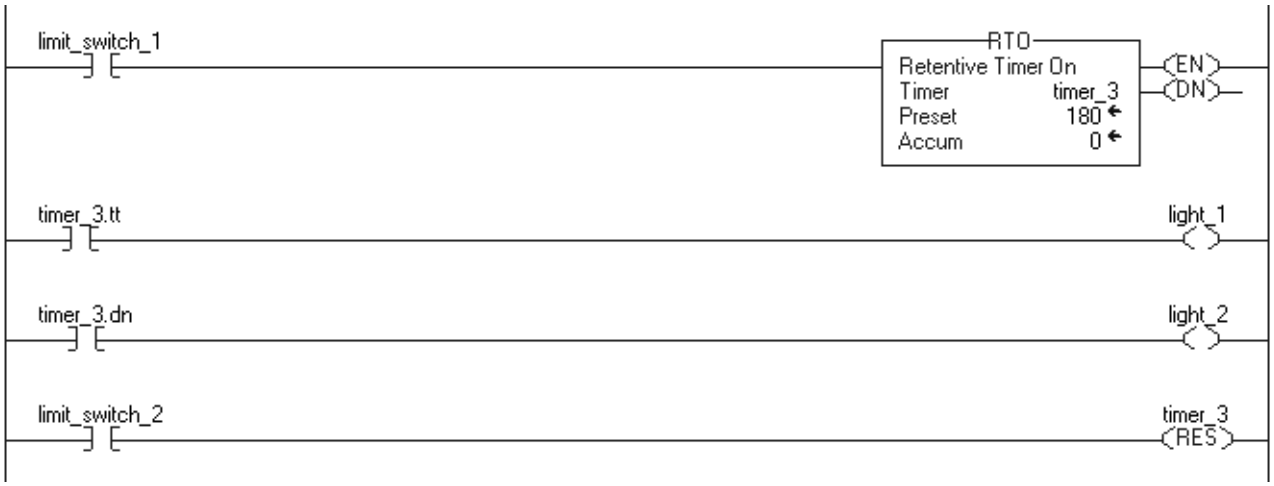
故障条件：

出现主要故障的条件	故障类型	故障代码
.PRE < 0	4	34
.ACC < 0	4	34

执行：

条件	梯形图操作
预扫描	清零 .EN、 .TT 和 .DN 位。 不修改 .ACC 值。 梯级输出条件设置为假。
梯级输入条件为假	清零 .EN 和 .TT 位。 不修改 .DN 位。 不修改 .ACC 值。 梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

示例： 当 *limit_switch_1* 置位时， *light_1* 接通 180 ms(*timer_2* 正在计时)。当 *timer_3.acc* 值达到 180 时，*light_1* 熄灭，*light_2* 点亮。*Light_2* 一直点亮，直到 *timer_3* 复位。如果在 *timer_3* 计时期间 *limit_switch_2* 清零，*light_1* 将保持接通。当 *limit_switch_2* 置位时，RES 指令复位 *timer_3*(清零状态位和 .ACC 值)。



带复位的接通延时计时器 (TONR)

TONR 指令是非保持型计时器，当 TimerEnable 置位时累计时间。

该指令功能在梯形图中以两个独立指令提供：

- TON(请参见第 106 页)。
- RES(请参见第 143 页)。

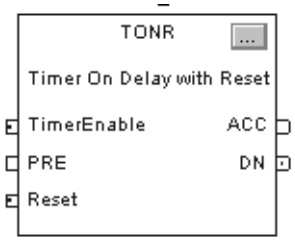
操作数：

结构化文本

```
TONR(TONR_tag);
```

变量	类型	格式	说明
TONR 标签	FBD_TIMER	结构	TONR 结构

功能块



操作数	类型	格式	说明
TONR 标签	FBD_TIMER	结构	TONR 结构

FBD_TIMER 结构

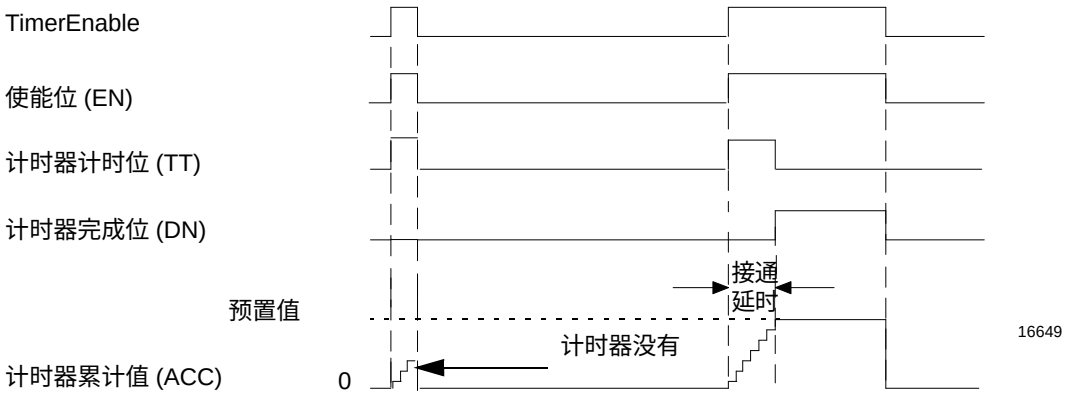
输入参数	数据类型	说明
EnableIn	BOOL	功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。执行该指令。
TimerEnable	BOOL	如果为置位状态，则会使能计时器运行并累计时间。 默认为清零状态。
PRE	DINT	计时器预置值。这是在计时完成之前 ACC 必须达到的值，以 1 ms 为单位。 如果无效，该指令将置位 Status 中的相应位，并且计时器不执行。 有效值 = 0 到最大正整数
复位 (Reset)	BOOL	请求复位计时器。置位时，计时器复位。 默认为清零状态。
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
ACC	BOOL	以毫秒为单位的累计时间。
EN	BOOL	计时器使能输出。指示计时器指令是否使能。

输入参数	数据类型	说明
TT	BOOL	计时器计时输出。置位时，表示正在执行计时操作。
DN	BOOL	计时完成输出。指示累计时间大于等于预置值的时间。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	该指令检测到下列执行错误之一。这不是次要或主要控制器错误。检查其余状态位，确定发生的情况。
PresetInv (Status.1)	BOOL	预置值无效。

说明： TONR 指令一直累计时间，直到：

- 禁止 TONR 指令。
- ACC ≥ PRE。

时基始终是 1 ms。例如，对于两秒计时器，输入 2000 作为 PRE 值。



置位 Reset 输入参数将复位该指令。如果当 Reset 置位的情况下 TimerEnable 置位，则当 Reset 清零时 TONR 指令将再次开始计时。

计时器的运行方式是从当前时间中减去其上次扫描的时间：

```
ACC = ACC + (current_time - last_time_scanned)
```

更新 ACC 后，计时器将 last_time_scanned 设置为 current_time。这样，计时器即为下次扫描做好准备。

重要事项

确保在计时器运行时至少每 69 分钟扫描一次计时器。否则， ACC 值将不正确。

last_time_scanned 值的最大范围是 69 分钟。如果没有在 69 分钟内扫描计时器，计时器的计算将溢出。 如果发生此情况， ACC 值是不正确的。

如果将计时器放置在以下位置，则当计时器运行时，请在 69 分钟内对其进行扫描：

- 子例程。
- JMP 和 LBL 指令间的代码段。
- 顺序功能图 (SFC)。
- 事件型任务或周期型任务。
- 阶段的状态例程。

算术状态标志： 不受影响

故障条件： 无

执行：

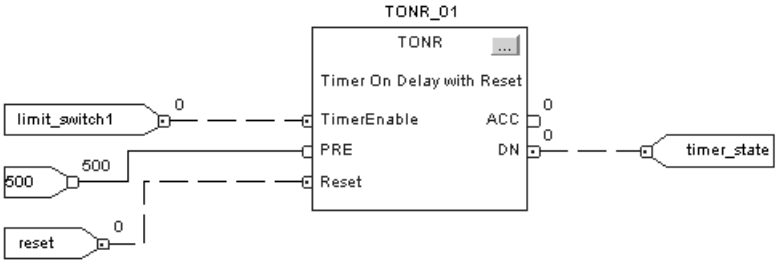
条件	功能块操作	结构化文本操作
预扫描	不执行任何操作。	不执行任何操作。
指令第一次扫描	清零 EN、 TT 和 DN。 ACC 值设置为 0。	清零 EN、 TT 和 DN。 ACC 值设置为 0。
指令第一次运行	清零 EN、 TT 和 DN。 ACC 值设置为 0。	清零 EN、 TT 和 DN。 ACC 值设置为 0。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	无
EnableIn 处于置位状态	当 EnableIn 从清零状态切换到置位状态时，该指令按照指令第一次扫描的方式进行初始化。 执行该指令。 置位 EnableOut。	EnableIn 始终置位。 执行该指令。
复位	当 Reset 输入参数置位时，指令将清零 EN、 TT 和 DN 并设置 ACC = 0。	当 Reset 输入参数置位时，指令将清零 EN、 TT 和 DN 并设置 ACC = 0。
后扫描	不执行任何操作。	不执行任何操作。

示例： 每次扫描到 limit_switch1 置位， TONR 指令就会给 ACC 值增加经历的时间，直到 ACC 值达到 PRE 值。当 ACC ? PRE 时，置位 DN 参数，并置位 timer_state。

结构化文本

```
TONR_01.Preset := 500;

TONR_01.Reset := reset;

TONR_O1.TimerEnable := limit_switch1;


TONR(TONR_01);


timer_state := TONR_01.DN;
```

功能块示例



带复位的关断延时计时器 (TOFR)

TOFR 指令是非保持型计时器，当 TimerEnable 清零时累计时间。

该指令功能在梯形图中以两个独立指令提供：

- TON(请参见第 106 页)。
- RES(请参见第 143 页)。

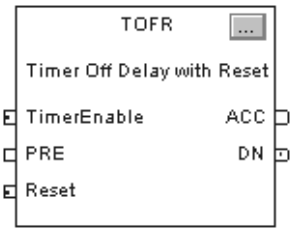
操作数：

结构化文本

```
TOFR(TOFR_tag);
```

变量	类型	格式	说明
TOFR 标签	FBD_TIMER	结构	TOFR 结构

功能块操作数



操作数	类型	格式	说明
TOFR 标签	FBD_TIMER	结构	TOFR 结构

FBD_TIMER 结构

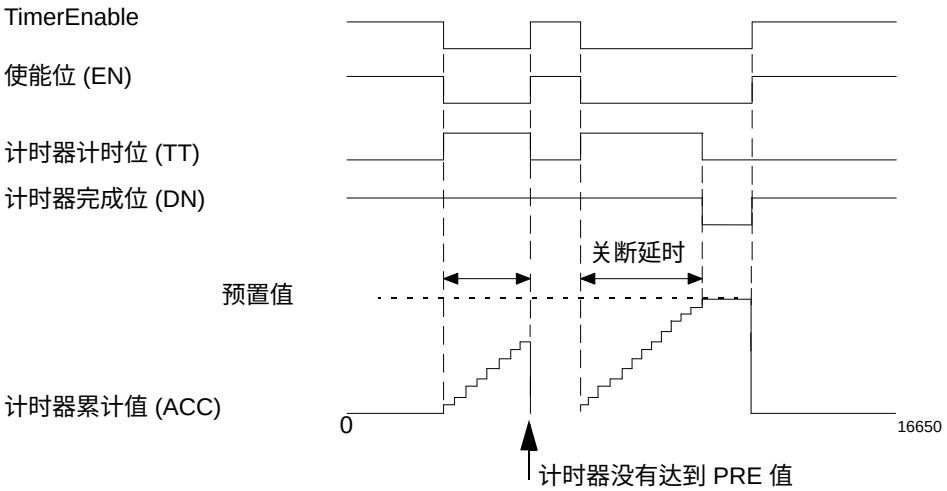
输入参数	数据类型	说明
EnableIn	BOOL	功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。执行该指令。
TimerEnable	BOOL	如果为清零状态，则会使能计时器运行并累计时间。 默认为清零状态。
PRE	DINT	计时器预置值。这是在计时完成之前 ACC 必须达到的值，以 1 ms 为单位。 如果无效，该指令将置位 Status 中的相应位，并且计时器不执行。 有效值 = 0 到最大正整数
复位 (Reset)	BOOL	请求复位计时器。置位时，计时器复位。 默认为清零状态。
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
ACC	BOOL	以毫秒为单位的累计时间。

输入参数	数据类型	说明
EN	BOOL	计时器使能输出。指示计时器指令是否使能。
TT	BOOL	计时器计时输出。置位时，表示正在执行计时操作。
DN	BOOL	计时完成输出。指示累计时间大于等于预置值的时间。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	该指令检测到下列执行错误之一。这不是次要或主要控制器错误。检查其余状态位，确定发生的情况。
PresetInv (Status.1)	BOOL	预置值无效。

说明： TOFR 指令一直累计时间，直到：

- 禁止 TOFR 指令。
- ACC ≥ PRE。

时基始终是 1 ms。例如，对于两秒计时器，输入 2000 作为 PRE 值。



置位 Reset 输入参数将复位该指令。如果当 Reset 置位时清零 TimerEnable，则当 Reset 清零时 TOFR 指令并不会再次开始计时。

计时器的运行方式是从当前时间中减去其上次扫描的时间：

```
ACC = ACC + (current_time - last_time_scanned)
```

更新 ACC 后，计时器将 *last_time_scanned* 设置为 *current_time*。这样，计时器即为下次扫描做好准备。

重要事项

确保在计时器运行时至少每 69 分钟扫描一次计时器。否则， ACC 值将不正确。

last_time_scanned 值的最大范围是 69 分钟。如果没有在 69 分钟内扫描计时器，计时器的计算将溢出。 如果发生此情况， ACC 值是不正确的。

如果将计时器放置在以下位置，则当计时器运行时，请在 69 分钟内对其进行扫描：

- 子例程
- JMP 和 LBL 指令间的代码段。
- 顺序功能图 (SFC)
- 事件型任务或周期型任务
- 阶段的状态例程

算术状态标志： 不受影响

故障条件： 无

执行：

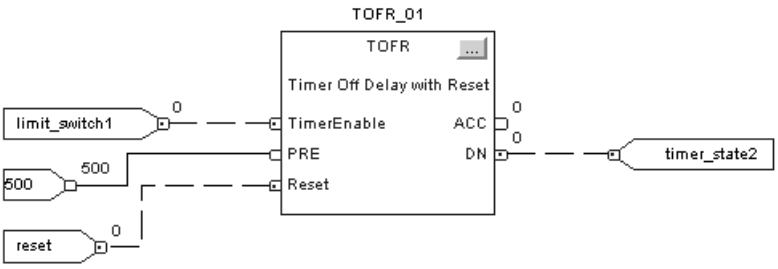
条件	功能块操作	结构化文本操作
预扫描	不执行任何操作。	不执行任何操作。
指令第一次扫描	清零 EN、 TT 和 DN。 ACC 值设置为 PRE。	清零 EN、 TT 和 DN。 ACC 值设置为 PRE。
指令第一次运行	清零 EN、 TT 和 DN。 ACC 值设置为 PRE。	清零 EN、 TT 和 DN。 ACC 值设置为 PRE。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	无
EnableIn 处于置位状态	当 EnableIn 从清零状态切换到置位状态时，该指令按照指令第一次扫描的方式进行初始化。 执行该指令。 置位 EnableOut。	EnableIn 始终置位。 执行该指令。
复位	当 Reset 输入参数置位时，指令将清零 EN、 TT 和 DN 并设置 ACC = PRE。请注意，这与对 TOF 指令使用 RES 指令不同。	当 Reset 输入参数置位时，指令将清零 EN、 TT 和 DN 并设置 ACC = PRE。请注意，这与对 TOF 指令使用 RES 指令不同。
后扫描	不执行任何操作。	不执行任何操作。

示例： 每次扫描到 *limit_switch1* 清零， TOFR 指令就会给 ACC 值增加经历的时间，直到 ACC 值达到 PRE 值。当 ACC ≥ PRE 时，清零 DN 参数，并置位 *timer_state2*。

结构化文本

```
TOFR_01.Preset := 500

TOFR_01.Reset := reset;

TOFR_O1.TimerEnable := limit_switch1;


TOFR(TOFR_01);


timer_state2 := TOFR_01.DN;
```

功能块



带复位的保持型接通计时器

RTOR 指令是保持型计时器，当 TimerEnable 置位时累计时间。

该指令功能在梯形图中以两个独立指令提供：

- RTO(请参见第 114 页)。
- RES(请参见第 143 页)。

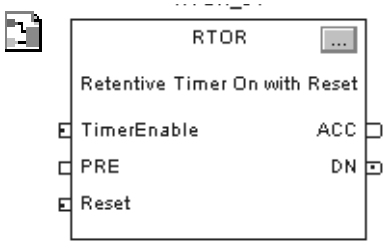
操作数：

结构化文本

```
RTOR(RTOR_tag);
```

变量	类型	格式	说明
RTOR 标签	FBD_TIMER	结构	RTOR 结构

功能块操作数



操作数	类型	格式	说明
RTOR 标签	FBD_TIMER	结构	RTOR 结构

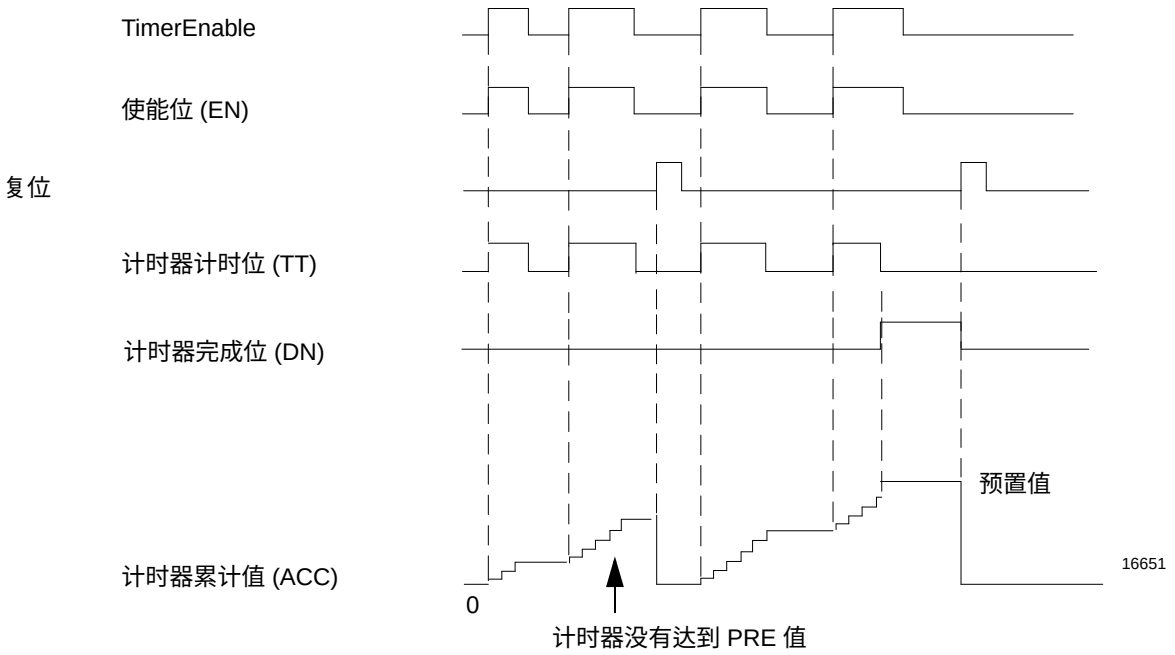
FBD_TIMER 结构

输入参数	数据类型	说明
EnableIn	BOOL	功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。执行该指令。
TimerEnable	BOOL	如果为置位状态，则会使能计时器运行并累计时间。 默认为清零状态。
PRE	DINT	计时器预置值。这是在计时完成之前 ACC 必须达到的值，以 1 ms 为单位。如果无效，该指令将置位 Status 中的相应位，并且计时器不执行。 有效值 = 0 到最大正整数
复位 (Reset)	BOOL	请求复位计时器。置位时，计时器复位。
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
ACC	DINT	以毫秒为单位的累计时间。即使 TimerEnable 输入清零，该值仍会被保留。这使得该块的行为不同于 TONR 块。

输入参数	数据类型	说明
EN	BOOL	计时器使能输出。指示计时器指令是否使能。
TT	BOOL	计时器计时输出。置位时，表示正在执行计时操作。
DN	BOOL	计时完成输出。指示累计时间大于等于预置值的时间。
Status	DINT	功能块的状态。
InstructFault (Status.0)	BOOL	该指令检测到下列执行错误之一。这不是次要或主要控制器错误。检查其余状态位，确定发生的情况。
PresetInv (Status.1)	BOOL	预置值无效。

说明： RTOR 指令一直累计时间直到被禁止。当禁止 RTOR 指令时，会保持其 ACC 值。必须使用 Reset 输入清零 .ACC 值。

时基始终是 1 ms。例如，对于两秒计时器，输入 2000 作为 PRE 值。



置位 Reset 输入参数将复位该指令。如果当 Reset 置位的情况下 TimerEnable 置位，则当 Reset 清零时 RTOR 指令将再次开始计时。

计时器的运行方式是从当前时间中减去其上次扫描的时间：

```
ACC = ACC + (current_time - last_time_scanned)
```

更新 ACC 后，计时器将 last_time_scanned 设置为 current_time。这样，计时器即为下次扫描做好准备。

重要事项

确保在计时器运行时至少每 69 分钟扫描一次计时器。否则， ACC 值将不正确。

last_time_scanned 值的最大范围是 69 分钟。如果没有在 69 分钟内扫描计时器，计时器的计算将溢出。 如果发生此情况， ACC 值是不正确的。

如果将计时器放置在以下位置，则当计时器运行时，请在 69 分钟内对其进行扫描：

- 子例程。
- JMP 和 LBL 指令间的代码段。
- 顺序功能图 (SFC)。
- 事件型任务或周期型任务。
- 阶段的状态例程。

算术状态标志： 不受影响

故障条件： 无

执行：

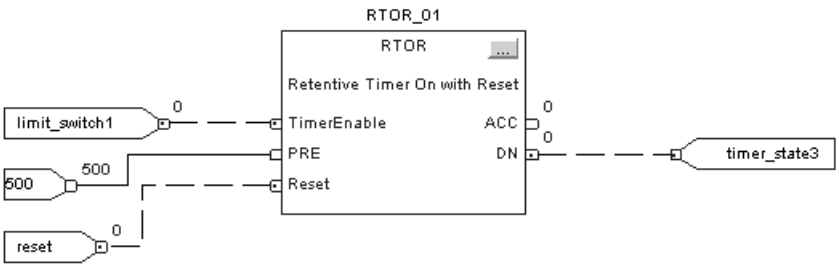
条件	功能块操作	结构化文本操作
预扫描	不执行任何操作。	不执行任何操作。
指令第一次扫描	清零 EN、 TT 和 DN 不修改 ACC 值	清零 EN、 TT 和 DN 不修改 ACC 值
指令第一次运行	清零 EN、 TT 和 DN 不修改 ACC 值	清零 EN、 TT 和 DN 不修改 ACC 值
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	无
EnableIn 处于置位状态	功能块 当 EnableIn 从清零状态切换到置位状态时，该指令按照指令第一次扫描的方式进行初始化。 执行该指令。 置位 EnableOut。	EnableIn 始终置位。 执行该指令。
复位	当 Reset 输入参数置位时，指令将清零 EN、 TT 和 DN 并设置 ACC = 0。	当 Reset 输入参数置位时，指令将清零 EN、 TT 和 DN 并设置 ACC = 0。
后扫描	不执行任何操作。	不执行任何操作。

示例： 每次扫描到 *limit_switch1* 置位， RTOR 指令就会给 ACC 值增加经历的时间，直到 ACC 值达到 PRE 值。当 ACC ≥ PRE 时，置位 DN 参数，并置位 *timer_state3*。

结构化文本

```
RTOR_01.Preset := 500

RTOR_01.Reset := reset;

RTOR_O1.TimerEnable := limit_switch1;


RTOR(RTOR_01);


timer_state3 := RTOR_01.DN;
```

功能块



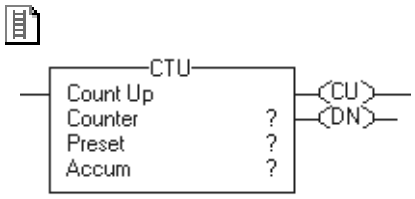
增计数 (CTU)

CTU 指令执行增计数操作。

该指令在结构化文本和功能块中以 CTUD 形式提供。

操作数：

梯形图

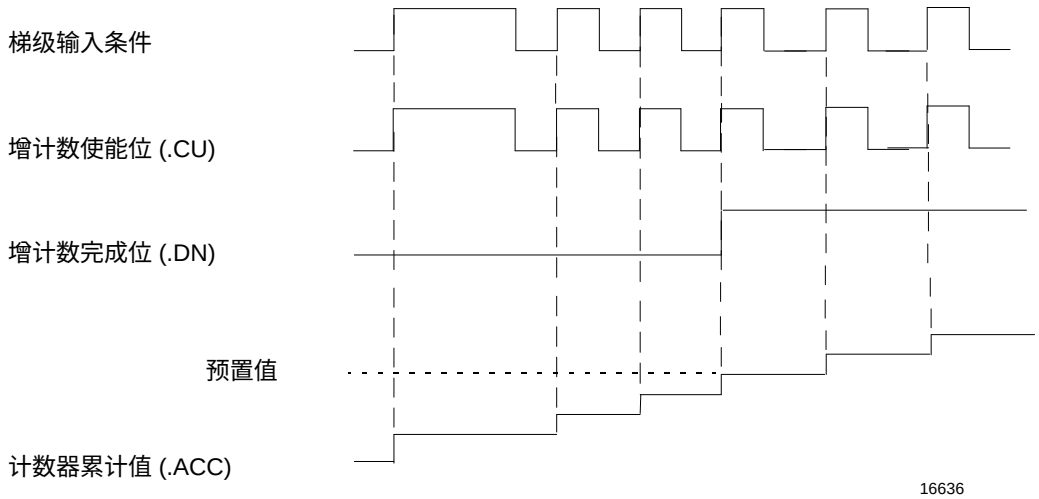


操作数	类型	格式	说明
计数器 (Counter)	COUNTER	标签	计数器结构
预置值 (Preset)	DINT	立即数	计数上限
Accum	DINT	立即数	计数器已计数的次数 初始值通常为 0

COUNTER 结构

助记符	数据类型	说明
.CU	BOOL	增计数使能位指示 CTU 指令是否使能。
.DN	BOOL	完成位指示 .ACC 是否大于等于 .PRE。
.OV	BOOL	上溢出位指示计数器超出 2,147,483,647 这一上限，计数器然后将翻转到 -2,147,483,648 并重新开始增计数。
.UN	BOOL	下溢出位指示计数器超出 -2,147,483,648 这一下限，计数器然后将翻转到 2,147,483,647 并重新开始减计数。
.PRE	DINT	预置值指定指令置位 .DN 位前，累计值必须达到的值。
.ACC	DINT	累计值指明指令已计数的次数。

说明： 当 CTU 指令使能且 .CU 位清零时，CTU 指令使计数器加 1。当 CTU 指令使能且 .CU 位置位时，或者 CTU 指令禁止时，CTU 指令保持其 .ACC 值。



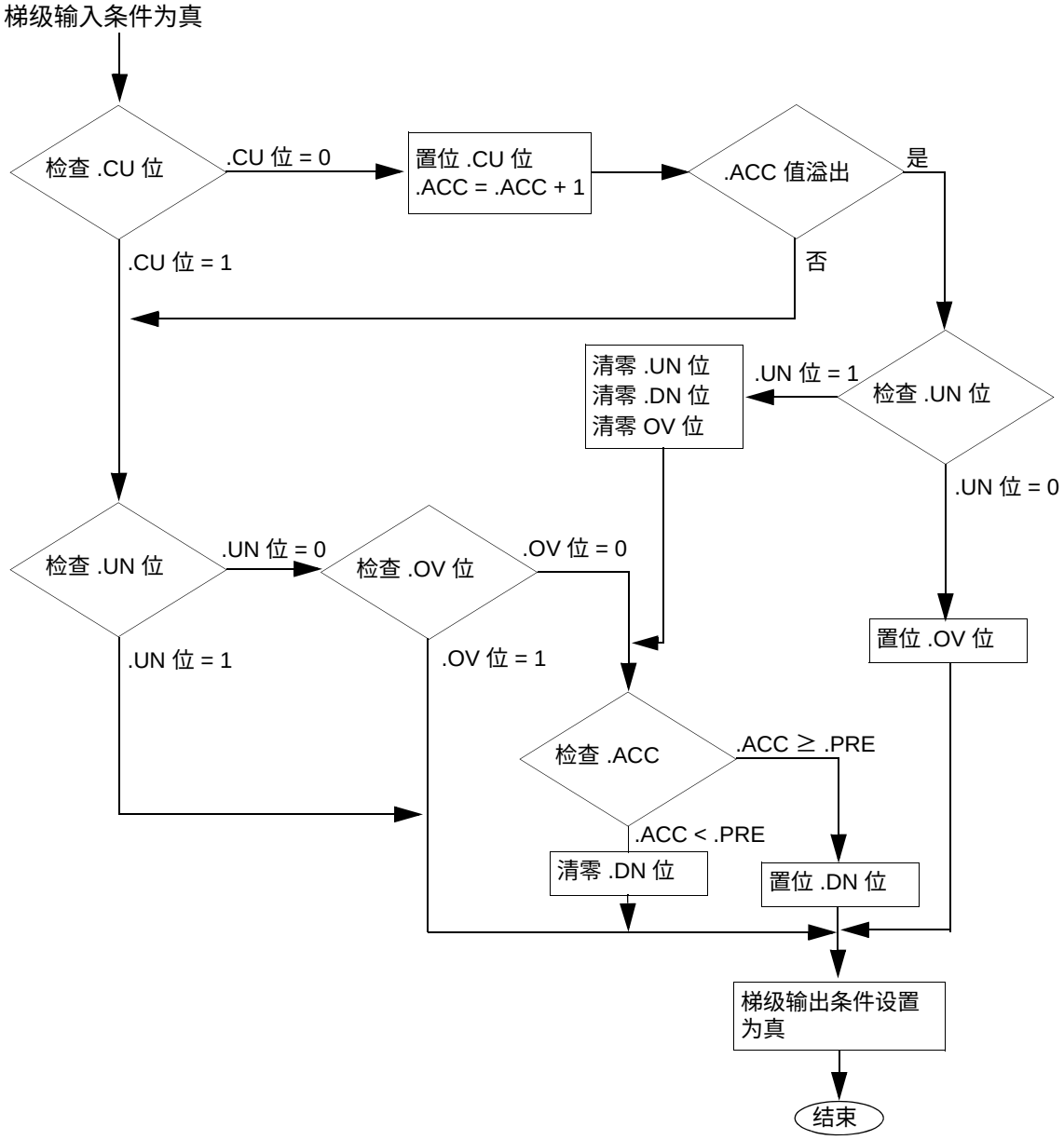
即使在 .DN 位置位后，累计值仍会继续递增。要清零累计值，请使用清除计数器结构的 RES 指令，或将 0 写入累计值。

算术状态标志： 不受影响

故障条件： 无

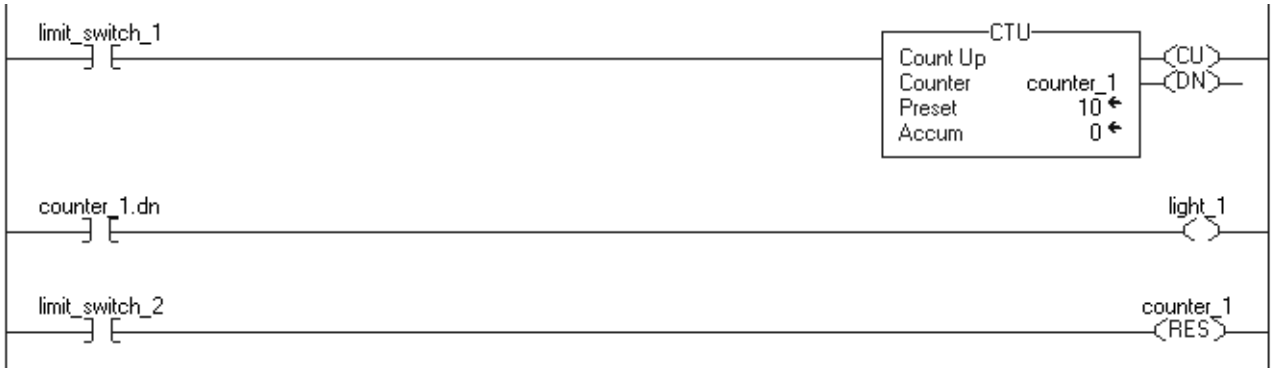
执行：

条件	梯形图操作
预扫描	置位 .CU 位，防止第一次程序扫描期间发生无效递增。 梯级输出条件设置为假。
梯级输入条件为假	清零 .CU 位。 梯级输出条件设置为假。



后扫描	梯级输出条件设置为假。

示例： 当 *limit_switch_1* 从禁止切换到使能达 10 次后，.DN 位置位，*light_1* 接通。如果 *limit_switch_1* 继续从禁止切换到使能， *counter_1* 继续增计数，并且 .DN 位保持置位状态。当 *limit_switch_2* 使能时，RES 指令复位 *counter_1*(清零状态位和 .ACC 值)， *light_1* 关闭。



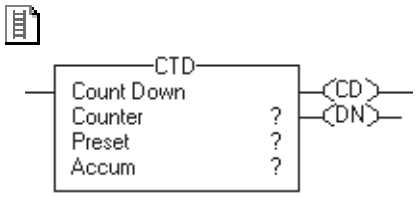
减计数 (CTD)

CTD 指令执行减计数操作。

该指令在结构化文本和功能块中以 CTUD 形式提供。

操作数：

梯形图



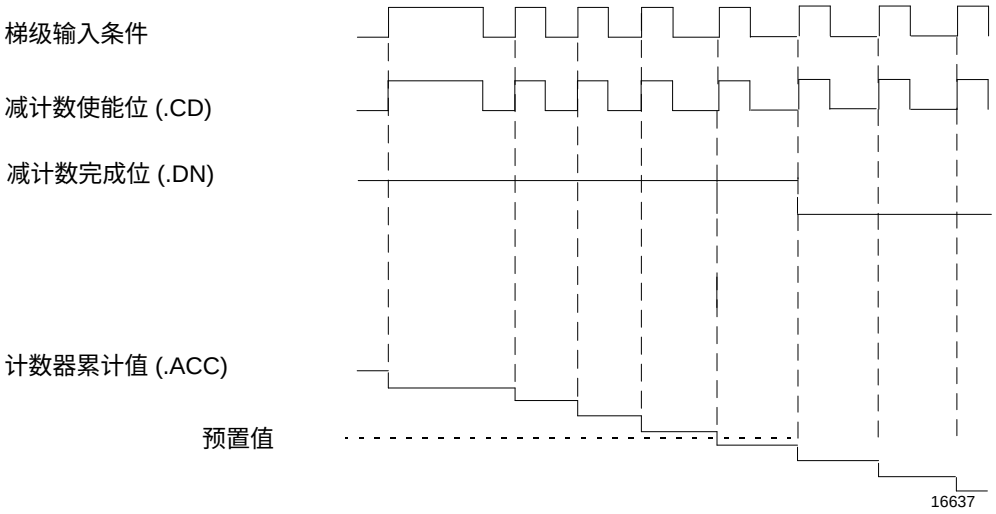
操作数	类型	格式	说明
计数器 (Counter)	COUNTER	标签	计数器结构
预置值 (Preset)	DINT	立即数	计数下限
Accum	DINT	立即数	计数器已计数的次数 初始值通常为 0

COUNTER 结构

助记符	数据类型	说明
.CD	BOOL	减计数使能位指示 CTD 指令是否使能。
.DN	BOOL	完成位指示 .ACC 是否大于等于 .PRE。
.OV	BOOL	上溢出位指示计数器超出 2,147,483,647 这一上限，计数器然后将翻转到 -2,147,483,648 并重新开始增计数。
.UN	BOOL	下溢出位指示计数器超出 -2,147,483,648 这一下限，计数器然后将翻转到 2,147,483,647 并重新开始减计数。
.PRE	DINT	预置值指定指令置位 .DN 位前，累计值必须达到的值。
.ACC	DINT	累计值指明指令已计数的次数。

说明： CTD 指令通常与引用相同计数器结构的 CTU 指令一起使用。

当 CTD 指令使能且 .CD 位清零时，CTD 指令使计数器减 1。当 CTD 指令使能且 .CD 位置位时，或者 CTD 指令禁止时，CTD 指令保持其 .ACC 值。



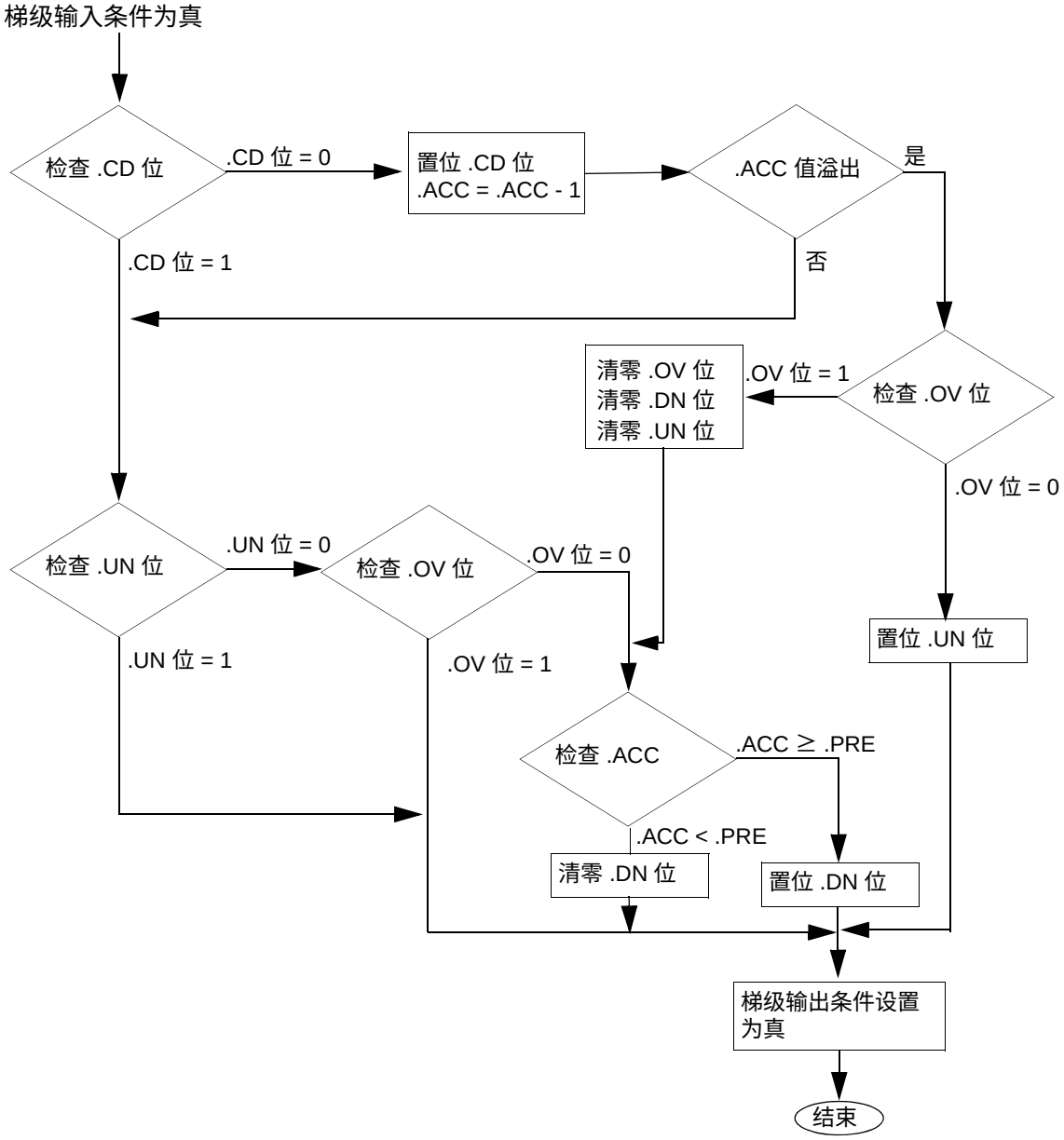
即使在 .DN 位置位后，累计值仍会继续递减。要清零累计值，请使用清除计数器结构的 RES 指令，或将 0 写入累计值。

算术状态标志： 不受影响

故障条件： 无

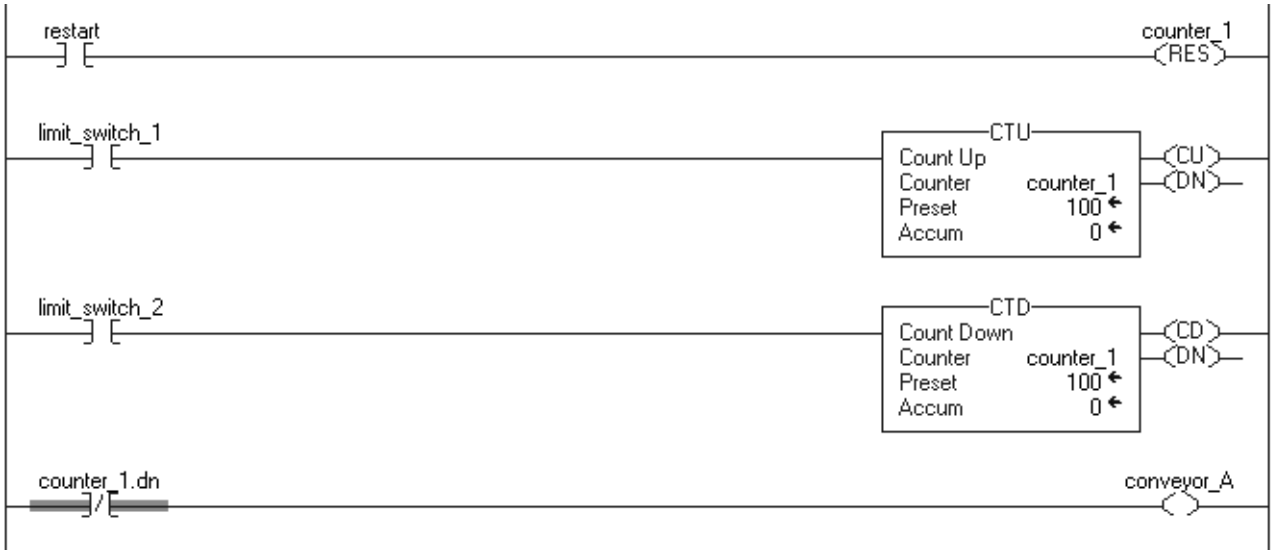
执行：

条件	梯形图操作
预扫描	置位 .CD 位，防止第一次程序扫描期间发生无效递减。 梯级输出条件设置为假。
梯级输入条件为假	清零 .CD 位。 梯级输出条件设置为假。



后扫描	梯级输出条件设置为假。

示例： 传送带将零件传送到缓冲区域中。每次有零件进入， *limit_switch_1* 就会使能， *counter_1* 加 1。每次有零件离开， *limit_switch_2* 就会使能， *counter_1* 减 1。如果缓冲区域中有 100 个零件 (*counter_1.dn* 置位)， *conveyor_A* 关闭并阻止传送带再传入任何其它零件，直到缓冲区有空间可以容纳其它零件。



增 / 减计数 (CTUD)

当 CUEnable 从清零状态转变为置位状态时，CTUD 指令以 1 为单位增计数。当 CDEnable 从清零状态转变为置位状态时，指令以 1 为单位减计数。

该指令功能在梯形图中以三个独立指令提供：

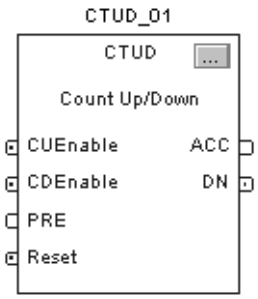
- CTU(请参见第 130 页)。
- CTD(请参见第 134 页)。
- RES(请参见第 143 页)。

操作数：

```
CTUD(CTUD_tag);
```

结构化文本

变量	类型	格式	说明
CTUD 标签	FBD_COUNTER	结构	CTUD 结构



功能块

操作数	类型	格式	说明
CTUD 标签	FBD_COUNTER	结构	CTUD 结构

FBD_COUNTER 结构

输入参数	数据类型	说明
EnableIn	BOOL	功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。执行该指令。
CUEnable	BOOL	使能增计数。当输入从清零状态转变为置位状态时，累加器以 1 为单位减计数。 默认为清零状态。

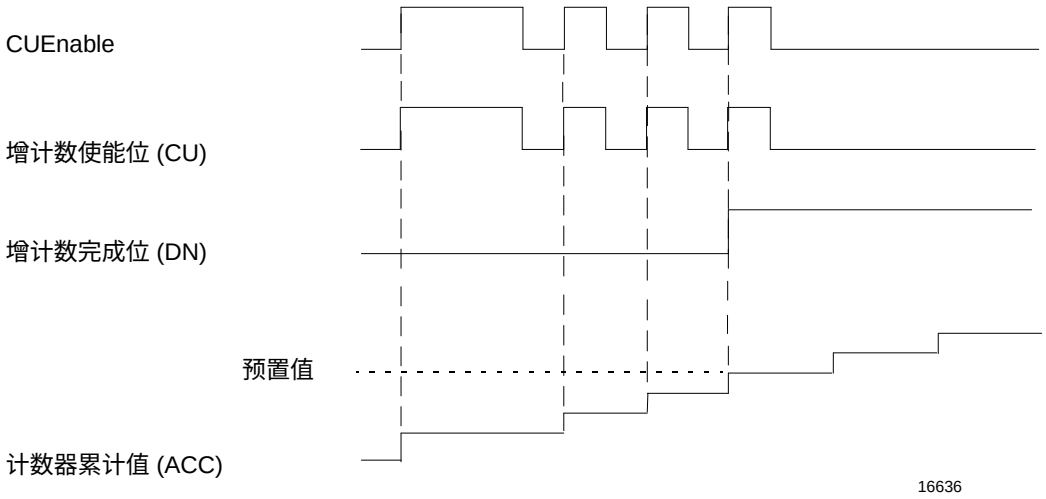
输入参数	数据类型	说明
CDEnable	BOOL	使能减计数。当输入从清零状态转变为置位状态时，累加器以 1 为单位减计数。 默认为清零状态。
PRE	DINT	计数器预置值。这是在 DN 置位之前累计值必须达到的值。 任何整数都有效 默认值为 0。
复位 (Reset)	BOOL	请求复位计数器。置位时，计数器复位。 默认为清零状态。

输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
ACC	DINT	累计值。
CU	BOOL	增计数使能。
CD	BOOL	减计数使能。
DN	BOOL	计数完成。当累计值大于等于预置值时置位。
OV	BOOL	计数器上溢出。指示计数器超出 2,147,483,647 这一上限。 计数器然后翻转到 –2,147,483,648 并重新开始增计数。
UN	BOOL	计数器下溢出。指示计数器超出 –2,147,483,648 这一下限。 计数器然后翻转到 2,147,483,647 并重新开始减计数。

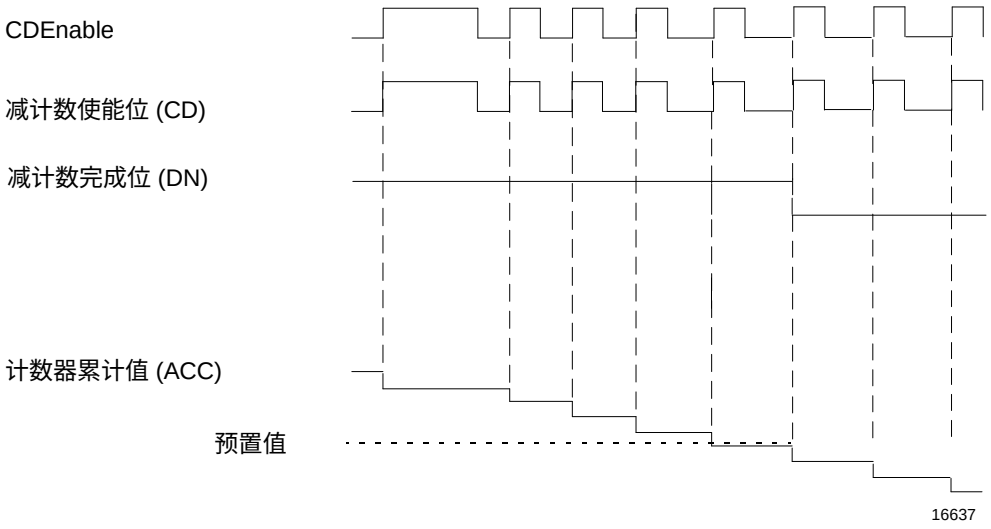
说明 当 CTUD 指令使能且 CUEnable 置位时，CTUD 指令使计数器加一。
当 CTUD 指令使能且 CDEnable 置位时， CTUD 指令使计数器减 1。

在同一扫描期间， CUEnable 和 CDEnable 输入参数都可以切换。该指令将依次执行增计数和减计数。

增计数



减计数



当禁止 CTUD 指令时， CTUD 指令保持其累计值。置位 FBD_COUNTER 结构的 Reset 输入参数将复位该指令。

算术状态标志： 不受影响

故障条件： 无

执行：

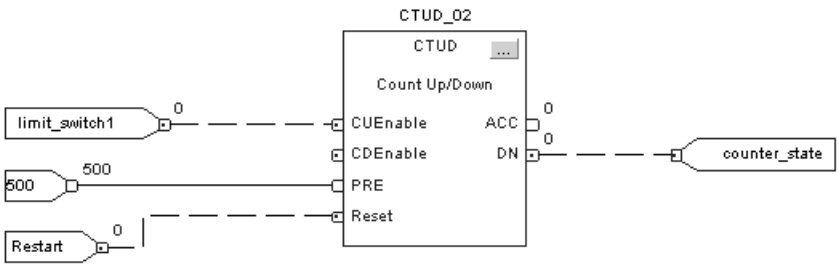
条件	功能块操作	结构化文本操作
预扫描	无需初始化。	无需初始化。
指令第一次扫描	置位 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。	置位 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。
指令第一次运行	置位 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。	置位 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	无
EnableIn 处于置位状态	该指令置位 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。 在 EnableIn 从清零状态转变为置位状态时： • 执行该指令。 • 置位 EnableOut。	该指令置位 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。 EnableIn 始终置位。 执行该指令。
复位	置位时，该指令清零 $CUEnable_{n-1}$、 $CDEnable_{n-1}$、 CU、 CD、 DN、 OV 和 UN，并设置 ACC = 0。	置位时，该指令清零 $CUEnable_{n-1}$、 $CDEnable_{n-1}$、 CU、 CD、 DN、 OV 和 UN，并设置 ACC = 0。
后扫描	不执行任何操作。	不执行任何操作。

示例： 当 *limit_switch1* 从清零状态转变为置位状态时， CUEnable 置位一个扫描周期， CTUD 指令使 ACC 值加 1。当 ACC ≥ PRE 时，置位 DN 参数，这样将在执行 CTUD 指令后使能功能块指令。

结构化文本

```
CTUD_01.Preset := 500;

CTUD_01.Reset := Restart;

CTUD_O1.CUEnable := limit_switch1;


CTUD(CTUD_01);


counter_state := CTUD_01.DN;
```

功能块

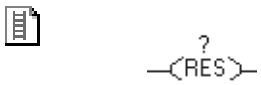

CTUD_02
CTUD
...
Count Up/Down
limit_switch1
0
CUEnable
ACC
0
CDEnable
DN
0
counter_state
500
500
PRE
Reset
Restart
0

复位 (RES)

RES 指令可复位 TIMER、 COUNTER 或 CONTROL 结构。

操作数：

梯形图



操作数	类型	格式	说明
结构	TIMER CONTROL COUNTER	标签	要复位的结构

说明： 使能时， RES 指令清零下列元素。

当 RES 指令用于	该指令清零
TIMER	.ACC 值 控制状态位
COUNTER	.ACC 值 控制状态位
CONTROL	.POS 值 控制状态位

注 意

由于 RES 指令清零 .ACC 值、 .DN 位和 .TT 位，所以不要使用 RES 指令复位 TOF 计时器。

算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	RES 指令复位指定结构。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

输入 / 输出指令
(MSG、 GSV、 SSV、 IOT)

简介

输入 / 输出指令用于向控制器写入数据或从控制器读取数据。输入 / 输出指令还用于向另一个网络中的另一个模块写入数据块中的数据或从中读取数据。

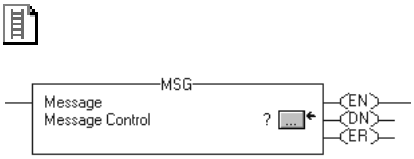
如果要	使用以下指令	可用编程语言	页码
对其它模块读取数据或写入数据	MSG	梯形图 结构化文本	146
获取控制器状态信息	GSV	梯形图 结构化文本	182
设置控制器状态信息	SSV	梯形图 结构化文本	182
在逻辑中的特定点向 I/O 模块或消费者标签控制器发送输出值。 触发另一控制器中的事件任务。	IOT	梯形图 结构化文本	209

信息 (MSG)

MSG 指令可从网络中的另一个模块异步读取数据块或向其异步写入数据块。

操作数：

梯形图



操作数	类型	格式	说明
信息控制 (Message control)	MESSAGE	标签	MESSAGE 结构

结构化文本

```
MSG(MessageControl);
```

操作数与梯形图 MSG 指令的操作数相同。

MESSAGE 结构

注 意

如果多次校验状态位。

控制器将改变与逻辑扫描不同步的 DN、 ER、 EW 和 ST 位。如果在逻辑中的多个位置校验这些位，请使用它们的副本。否则，这些位可能在扫描期间发生变化，从而使逻辑不按照预期方式工作。

生成副本的一种方法是使用 FLAGS 字。将 FLAGS 字复制到另一个标签中并校验副本中的这些位。

重要事项

不要更改 MSG 指令的以下状态位：

- DN
- EN
- ER
- EW
- ST

不要单独更改这些位，也不要将它们作为 FLAGS 字的一部分来更改。如果进行更改，控制器可能发生不可恢复的故障。控制器发生不可恢复的故障时，会将项目从其内存中清除。

助记符	数据类型	说明	
.FLAGS	INT	通过 FLAGS 子元素，可以访问一个 16 位字中的状态子元素 (位)。	
		位	对应子元素
		2	.EW
		4	.ER
		5	.DN
		6	.ST
		7	.EN
		8	.TO
		9	.EN_CC
		重要说明：不要更改 FLAGS 成员的 EW、 ER、 DN 或 ST 位。例如，不要清除整个 FLAGS 字。控制器会忽略更改并使用内部存储的位值。	
.ERR	INT	如果 .ER 位置位，则错误代码字将显示 MSG 指令的错误代码。	
.EXERR	INT	扩展错误代码字为某些错误代码指定附加错误代码信息。	
.REQ_LEN	INT	请求的长度指定信息指令尝试传送的字数。	
.DN_LEN	INT	完成的长度标识实际传送的字数。	
.EW	BOOL	当控制器检测到信息请求已进入队列时，将置位使能等待位。 .ST 位置位时，控制器将复位 .EW 位。 重要说明：不要更改 EW 位。控制器会忽略更改并使用内部存储的位值。	
.ER	BOOL	当控制器检测到传送失败时，将置位错误位。下次梯级输入条件从假变为真时，.ER 位将复位。 重要说明：不要更改 ER 位。	
.DN	BOOL	成功传送完最后一个信息包时，完成位将置位。下次梯级输入条件从假变为真时，.DN 位将复位。 重要说明：不要更改 DN 位。	
.ST	BOOL	当控制器开始执行 MSG 指令时，启动位将置位。 .DN 位或 .ER 位置位时， .ST 位将复位。 重要说明：不要更改 ST 位。控制器会忽略更改并使用内部存储的位值。	
.EN	BOOL	当梯级输入条件变为真时，使能位将置位并一直保持置位状态，直到 .DN 位或 .ER 位被置位并且梯级输入条件为假。如果梯级输入条件变为假，但 .DN 位和 .ER 位已清零，则 .EN 位将保持置位状态。 重要说明：不要更改 EN 位。	
.TO	BOOL	如果手动置位 .TO 位，则控制器将停止处理信息指令，并置位 .ER 位。	
.EN_CC	BOOL	使能缓存连接位确定如何管理 MSG 连接。请参见 选择缓存连接选项 第 179 页。即使 .EN_CC 位已置位，发送到串口的 MSG 指令连接也不会被缓存连接。	
.ERR_SRC	SINT	RSLogix 5000 软件用它来在 “ 信息组态 ” (Message Configuration) 对话框中显示错误路径	

助记符	数据类型	说明	
.DestinationLink	INT	要更改 DH+ 或“ 带源 ID 的 CIP ” (CIP with Source ID) 信息的“ 目标链接 ” (Destination Link)，请将此设置为所需值。	
.DestinationNode	INT	要更改 DH+ 或“ 带源 ID 的 CIP ” (CIP with Source ID) 信息的“ 目标节点 ” (Destination Node)，请将此设置为所需值。	
.SourceLink	INT	要更改 DH+ 或“ 带源 ID 的 CIP ” (CIP with Source ID) 信息的“ 源链接 ” (Source Link)，请将此设置为所需值。	
.Class	INT	要更改“ CIP 通用 ” (CIP Generic) 信息的“ 级别 ” (Class) 参数，请将此设置为所需值。	
.Attribute	INT	要更改“ CIP 通用 ” (CIP Generic) 信息的“ 属性 ” (Attribute) 参数，请将此设置为所需值。	
.Instance	DINT	要更改“ CIP 通用 ” (CIP Generic) 信息的“ 实例 ” (Instance) 参数，请将此设置为所需值。	
.LocalIndex	DINT	如果使用星号 [*] 指定本地数组的元素编号， LocalIndex 将提供此元素编号。要更改元素编号，请将此设置为所需值。	
		如果信息	则本地数组为
		读取数据	目标元素
		写入数据	源元素
.Channel	SINT	要将信息从 1756-DHRIO 模块的其它通道发送出去，请将此设置为所需值。请使用 ASCII 字符 A 或 B。	
.Rack	SINT	要更改块传送信息的机架编号，请将此成员设置为所需的机架编号 (八进制)。	
.Group	SINT	要更改块传送信息的组编号，请将此成员设置为所需的组编号 (八进制)。	
.Slot	SINT	要更改块传送信息的插槽编号，请将此成员设置为所需的插槽编号。	
		如果信息通过此网络传输	则将节点编号指定为
		通用远程 I/O	八进制
		ControlNet	十进制 (0…15)
.Path	STRING	要将通信发送到其它控制器，请将此成员设置为新路径。 • 键入十六进制的路径值。 • 省略逗号 [,] 例如，对于路径 1, 0, 2, 42, 1, 3，请输入 $01$00$02$2A$01$03。 要浏览到某设备并自动创建部分或所有新字符串，请右键单击字符串标签，然后选择“ 转到信息路径编辑器 ” (Go to Message Path Editor)。	
.RemoteIndex	DINT	如果使用星号 [*] 指定远程数组的元素编号， RemoteIndex 将提供此元素编号。要更改元素编号，请将此设置为所需值。	
		如果信息	则远程数组为
		读取数据	源元素
		写入数据	目标元素

助记符	数据类型	说明	
.RemoteElement	STRING	要指定信息发送到的控制器中的其它标签或地址，请将此设置为所需值。输入 ASCII 字符形式的标签或地址。	
		如果信息	则远程数组为
		读取数据	源元素
		写入数据	目标元素
.UnconnnectedTimeout	DINT	未连接信息或建立连接时的超时。默认值为 30 秒。	
		如果信息	则
		未连接	如果控制器在 UnconnectedTimeout 时间内未得到响应，ER 位将开启。
		已连接	如果控制器在 UnconnectedTimeout 时间内未得到关于建立连接的响应， ER 位将开启。
.ConnectionRate	DINT	连接后的已连接信息的超时。此超时针对来自另一设备的关于数据发送的响应。	
.TimeoutMultiplier	SINT	• 只有在建立连接后才会应用此超时。 • 超时 = ConnectionRate x TimeoutMultiplier。 • 默认 ConnectionRate 为 7.5 秒。 • 默认 TimeoutMultiplier 为 0(表示倍乘系数为 4)。 • 已连接信息的默认超时为 30 秒 (7.5 秒 x 4 = 30 秒)。 • 要更改超时，请更改 ConnectionRate 并使 TimeoutMultiplier 保留为默认值。	

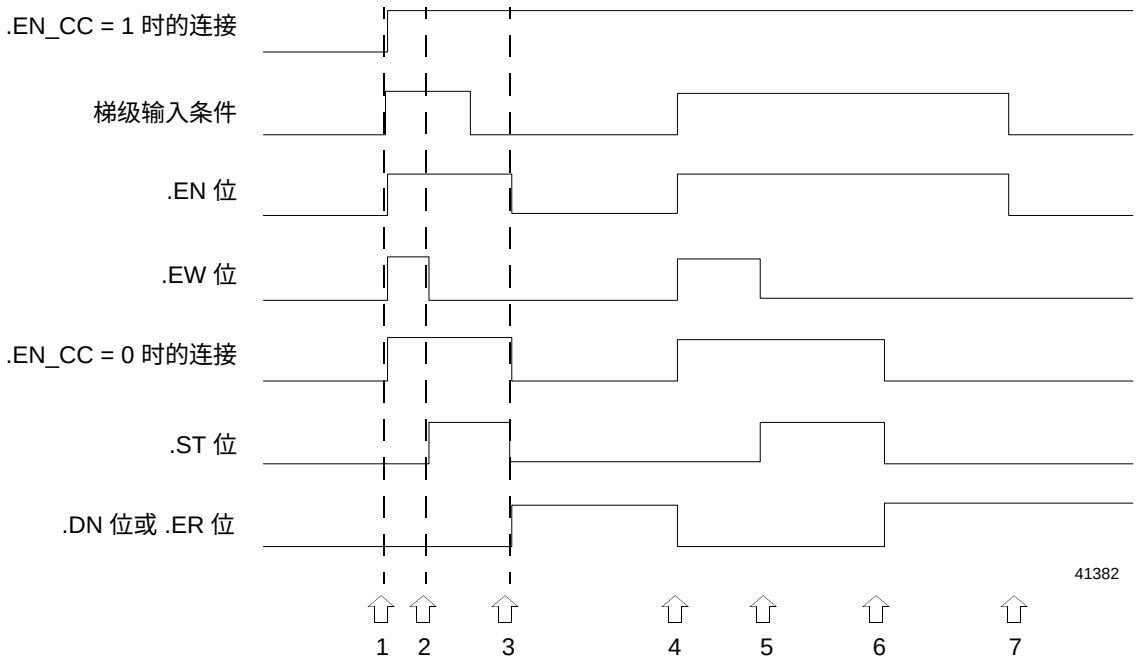
说明　MSG 指令传送数据的元素。

这是一条触发执行指令。

- 在梯形图中，每次梯级输入条件从清零跳变为置位时，都应执行此指令。
- 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。

请参见附录 B。

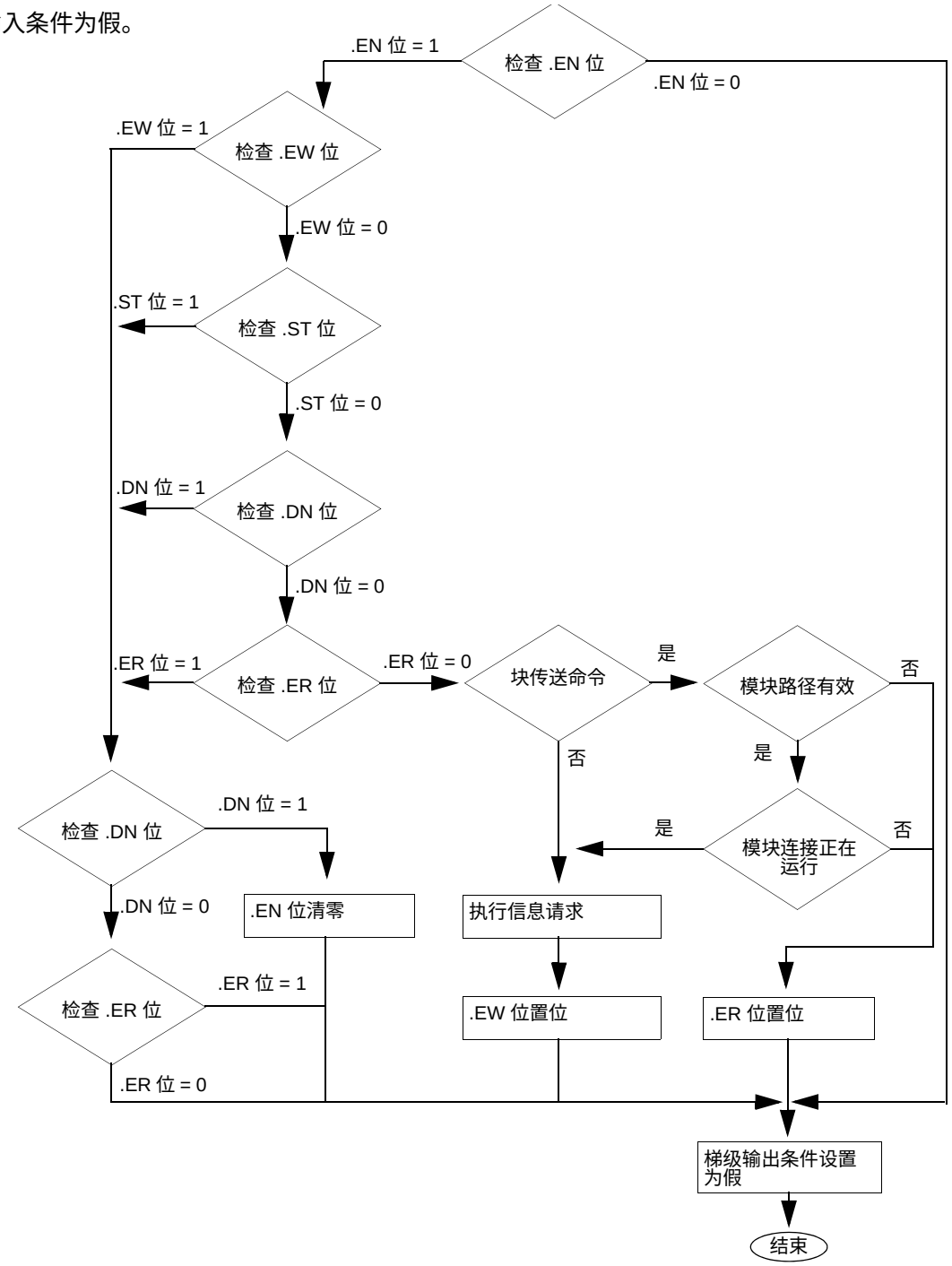
每个元素尺寸取决于所指定的数据类型和所使用的信息命令类型。



其中	说明	其中	说明
1	梯级输入条件为真。 .EN 置位。 .EW 置位。 连接断开 *。	5	发送信息。 .ST 置位。 .EW 清零。
2	发送信息。 .ST 置位。 .EW 清零。	6	信息已完成或出错。 梯级输入条件仍然为真。 .DN 或 .ER 置位。 .ST 清零。 连接关闭 (如果 .EN_CC = 0)。
3	信息已完成或出错。 梯级输入条件为假。 .DN 或 .ER 置位。 .ST 清零。 连接关闭 (如果 .EN_CC = 0)。 .EN 清零 (梯级输入条件为假)。	7	梯级输入条件变为假并且 .DN 或 .ER 置位。 .EN 清零。
4	梯级输入条件为真。 .DN 或 .ER 之前已置位。 .EN 置位。 .EW 置位。连接断开 *。 .DN 或 .ER 清零。	N/A	N/A

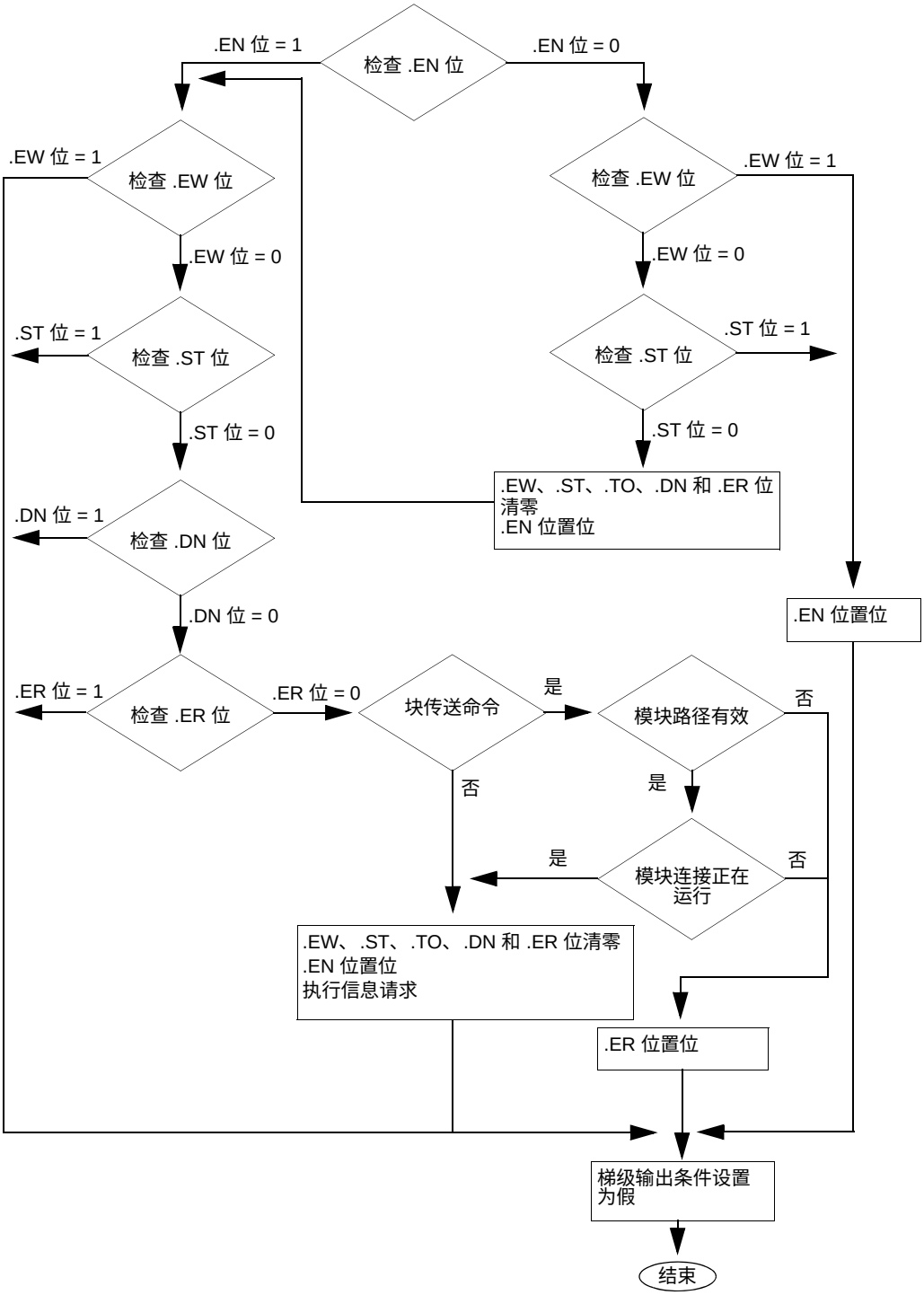
执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不采取任何操作。

条件	梯形图操作	结构化文本操作
梯级输入条件为假。		
梯级输入条件为真。	执行该指令。 梯级输出条件设置为真。	N/A

条件	梯形图操作	结构化文本操作
EnableIn 置位。	N/A	EnableIn 始终置位。 执行该指令。

指令执行



后扫描	梯级输出条件设置为假。	不采取任何操作。

算术状态标志： 不受影响

故障条件： 无

MSG 错误代码

错误代码取决于 MSG 指令的类型。

错误代码

RSLogix 5000 编程软件不会显示完整说明。

错误代码 (十六进制)	说明	软件中的显示内容
0001	连接故障 (参见扩展错误代码)	与说明相同
0002	资源不足	
0003	无效值	
0004	IOI 语法错误 (参见扩展错误代码)	
0005	目标未知、级别不支持、实例未定义或结构元素未定义 (参见扩展错误代码)	
0006	数据包空间不足	
0007	连接断开	
0008	服务不支持	
0009	数据段中有错误或属性值无效	
000A	属性列表错误	
000B	状态已存在	
000C	对象模型冲突	
000D	对象已存在	
000E	属性无法设置	
000F	权限被拒绝	
0010	设备状态冲突	
0011	应答不合适	
0012	片段原型	
0013	命令数据不足	
0014	属性不支持	
0015	数据过多	
001A	网桥请求太大	
001B	网桥响应太大	
001C	缺少属性列表	

错误代码 (十六进制)	说明	软件中的显示内容
001D	属性列表无效	与说明相同
001E	嵌入式服务错误	
001F	连接相关故障 (参见扩展错误代码)	
0022	收到无效应答	
0025	关键段错误	
0026	无效 IOI 错误	
0027	列表中存在非预期属性	
0028	DeviceNet 错误 - 子元素 ID 无效	
0029	DeviceNet 错误 - 子元素不可设置	
00D1	模块未处于运行状态	未知错误
00FB	信息端口不支持	
00FC	信息数据类型不支持	
00FD	信息未初始化	
00FE	信息超时	
00FF	常规错误 (参见扩展错误代码)	

扩展错误代码

RSLogix 5000 编程软件不会显示扩展错误代码任何文字。

下面是错误代码 **0001** 的扩展错误代码。

扩展错误代码 (十六进制)	说明
0100	连接正在使用中
0103	传送不受支持
0106	所有权冲突
0107	未找到连接
0108	连接类型无效
0109	连接尺寸无效
0110	模块未组态
0111	EPR 不支持
0114	模块错误
0115	设备类型错误
0116	版本错误
0118	组态格式无效
011A	应用连接超出

扩展错误代码 (十六进制)	说明
0203	连接超时
0204	未连接信息超时
0205	未连接发送参数错误
0206	信息过大
0301	无缓冲内存
0302	带宽不可用
0303	无可用筛选器
0305	签名匹配
0311	端口不可用
0312	链接地址不可用
0315	段类型无效
0317	连接未规划

下面是错误代码 **001F** 的扩展错误代码。

扩展错误代码 (十六进制)	说明
0203	连接超时

下面是错误代码 **0004** 和 **0005** 的扩展错误代码。

扩展错误代码 (十六进制)	说明
0000	内存超出扩展状态
0001	实例超出扩展状态

下面是错误代码 **00FF** 的扩展错误代码。

扩展错误代码 (十六进制)	说明
2001	IOI 过多
2002	参数值错误
2018	拒绝接收信号
201B	尺寸过小
201C	尺寸无效
2100	权限无效
2101	钥匙开关位置无效
2102	密码无效
2103	未发出密码
2104	地址超出范围
2105	地址和数量超出范围
2106	数据正在使用
2107	类型无效或不支持

扩展错误代码 (十六进制)	说明
2108	控制器处于上载或下载模式
2109	试图更改数组维数
210A	符号名称无效
210B	符号不存在
210E	搜索失败
210F	任务无法开始
2110	无法写入
2111	无法读取
2112	共享例程不可编辑
2113	控制器处于故障模式
2114	运行模式被禁止

PLC 和 SLC 错误代码 (.ERR)

Logix 固件版本 10.x 及更高版本为与 PLC 和 SLC 信息类型 (PCCC 信息) 相关的错误提供了新的错误代码。

- 这个改变使 RSLogix 5000 软件能够为许多错误显示更有价值的说明。以前，软件不为与 00F0 错误代码相关的任何错误提供说明。
- 这个改变也使错误代码与其它控制器 (如 PLC-5 控制器) 返回的错误代码的一致性更高。

下表显示了 R9.x 及更低版本与 R10.x 及更高版本之间在错误代码方面的改变。通过这种改变， .ERR 子元素可为每个 PCCC 错误返回一个惟一值。这些错误不再需要 .EXERR。

PLC 和 SLC 错误代码 (十六进制)

R9.x 及更低版本		**R10.x** 及更高版本		说明
.ERR	**.EXERR**	**.ERR**	**.EXERR**	
0010		1000		来自本地处理器的命令或格式非法
0020		2000		通信模块未工作
0030		3000		远程节点丢失、已断开或被关闭
0040		4000		处理器已连接但存在故障 (硬件)
0050		5000		站编号错误
0060		6000		请求的功能不可用
0070		7000		处理器处于编程模式
0080		8000		处理器的兼容性文件不存在
0090		9000		远程节点无法缓冲命令
00B0		B000		处理器正在下载，因此无法访问
00F0	0001	F001		处理器未正确转换地址
00F0	0002	F002		地址不完整
00F0	0003	F003		地址不正确
00F0	0004	F004		地址格式非法 - 符号未找到
00F0	0005	F005		地址格式非法 - 符号的字符数为 0 或超过设备支持的最大数目
00F0	0006	F006		地址文件在目标处理器中不存在
00F0	0007	F007		目标文件对于请求的字数而言过小
00F0	0008	F008		无法完成请求 在多包操作过程中，情况发生变化
00F0	0009	F009		数据或文件过大 内存不可用

PLC 和 **SLC** 错误代码 **(** 十六进制 **)** （续）

R9.x 及更低版本		**R10.x** 及更高版本		说明
.ERR	**.EXERR**	**.ERR**	**.EXERR**	
00F0	000A	F00A		目标处理器无法将请求的信息放入包中
00F0	000B	F00B		权限错误；访问被拒绝
00F0	000C	F00C		请求的功能不可用
00F0	000D	F00D		请求是多余的
00F0	000E	F00E		命令无法执行
00F0	000F	F00F		溢出；柱状图溢出
00F0	0010	F010		无访问权限
00F0	0011	F011		请求的数据类型与可用的数据不匹配
00F0	0012	F012		命令参数不正确
00F0	0013	F013		地址引用位于已删除区域
00F0	0014	F014		命令执行失败，原因不明 PLC-3 柱状图溢出
00F0	0015	F015		数据转换错误
00F0	0016	F016		扫描器无法与 1771 机架适配器通信
00F0	0017	F017		适配器无法与模块通信
00F0	0018	F018		1771 模块响应无效
00F0	0019	F019		标签重复
00F0	001A	F01A		文件所有者处于激活状态 - 文件正在使用中
00F0	001B	F01B		程序所有者处于激活状态 - 有人正在下载或在线编辑程序
00F0	001C	F01C		磁盘文件被写保护或无法访问 (仅限离线状态)
00F0	001D	F01D		磁盘文件正被另一应用程序使用 更新不能执行 (仅限离线状态)

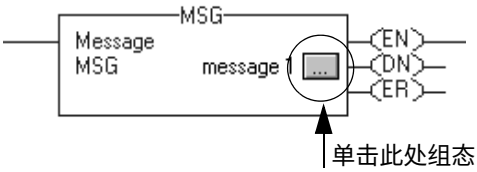
块传送错误代码

下面是 Logix5000 块传送专用的错误代码。

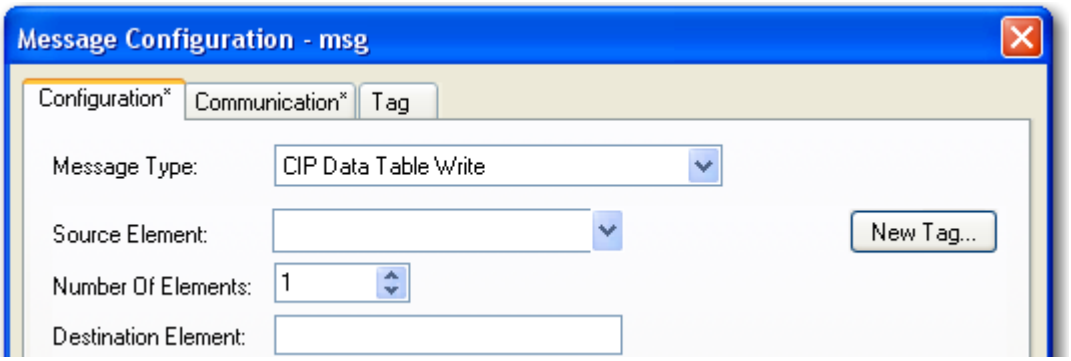
错误代码 (十六进制)	说明	软件中的显示内容
00D0	发出请求后的 3.5 秒内，扫描器没有收到来自块传送模块的块传送响应	未知错误
00D1	来自读取响应的校验和与数据流的校验和不匹配	
00D2	扫描器请求了读取或写入操作，但块传送模块以相反操作进行了响应	
00D3	扫描器请求了某个长度，但块传送模块以其它长度进行了响应	
00D6	扫描器收到来自块传送模块的响应，该响应指示写入请求失败	
00EA	扫描器未组态与包含此块传送模块的机架进行通信	
00EB	指定的逻辑插槽不适用于给定的机架尺寸	
00EC	当前有一个块传送请求正在进行，需要进行响应后，另一个请求才能开始	
00ED	块传送尺寸请求与有效块传送尺寸请求不一致	
00EE	块传送类型请求与预期的 BT_READ 或 BT_WRITE 不一致	
00EF	扫描器无法在块传送表中找到可用插槽来满足块传送请求	
00F0	块传送执行未完成时，扫描器收到复位远程 I/O 通道的请求	
00F3	远程块传送的队列已满	
00F5	没有为请求的机架或插槽组态通信通道	
00F6	没有为远程 I/O 组态通信通道	
00F7	块传送在完成之前超过指令中设置的块传送超时时限	
00F8	块传送协议出错 - 块传送未经请求	
00F9	由于通信通道错误，块传送数据丢失	
00FA	块传送模块请求的长度与关联的块传送指令的长度不同	
00FB	块传送读取数据的校验和错误	
00FC	适配器与块传送模块之间进行的块传送写入数据的传送无效	
00FD	块传送尺寸与块传送数据表中的索引尺寸之和大于块传送数据表文件的尺寸	

指定组态详细信息

输入 MSG 指令并指定 MESSAGE 结构后，可使用 “ 信息组态 ” (Message Configuration) 对话框指定信息的详细信息。



用户组态的详细内容取决于所选择的信息类型。



42976

如果目标设备为	选择以下一种信息类型	页码
Logix5000 控制器	CIP 数据表读取 (CIP Data Table Read)	162
	CIP 数据表写入 (CIP Data Table Write)	
使用 RSLogix 5000 软件组态的 I/O 模块	模块重新组态 (Module Reconfigure)	163
	CIP 通用 (CIP Generic)	164
PLC-5 控制器	PLC5 典型读取 (PLC5 Typed Read)	165
	PLC5 典型写入 (PLC5 Typed Write)	
	PLC5 字范围读取 (PLC5 Word Range Read)	
	PLC5 字范围写入 (PLC5 Word Range Write)	
SLC 控制器 MicroLogix 控制器	SLC 典型读取 (SLC Typed Read)	167
	SLC 典型写入 (SLC Typed Write)	
块传送模块	块传送读取 (Block-Transfer Read)	167
	块传送写入 (Block-Transfer Write)	
PLC-3 处理器	PLC3 典型读取 (PLC3 typed read)	168
	PLC3 典型写入 (PLC3 typed write)	
	PLC3 字范围读取 (PLC3 word range read)	
	PLC3 字范围写入 (PLC3 word range write)	
PLC-2 处理器	PLC2 非保护的读取 (PLC2 unprotected read)	169
	PLC2 非保护的写入 (PLC2 unprotected write)	

Logix5000 控制器作为目标设备指定数据

如果 Logix5000 控制器为目标设备，则使用此组态信息。

对于此属性	指定
源元素 (Source Element)	• 如果选择读信息类型，则 “ 源元素 ” 是在目标设备中读取的数据的地址。请使用目标设备的寻址语法。 • 如果选择写信息类型，则 “ 源标签 ” 是发送到目标设备的标签的第一个元素。 • 惟一不支持的源数据类型是布尔型。其它所有基本数据类型 (SINT、 INT、 DINT、 LINT、 REAL) 均可使用。预定义的、模块定义的或者用户自定义的结构类型也可用于发送通信。
元素数目 (Number of Elements)	读取 / 写入的元素数目取决于用户正在使用的数据类型。一个元素是指一个相关数据的组块。例如，标签 timer1 是由一个计时器结构组成的元素。
目标元素 (Destination Element)	• 如果选择读信息类型，则 “ 目标元素 ” 是 Logix5000 控制器中标签的第一个元素，用户在此处存放从目标设备中读取的数据。 • 如果选择写信息类型，则 “ 目标元素 ” 是目标设备中要写入数据的地址。

指定 CIP 数据表读取和写入信息

CIP 数据表读取和写入信息类型在 Logix5000 控制器之间传送数据。

选择此命令	如果要
CIP 数据表读取 (CIP Data Table Read)	从另一个控制器中读取数据。 “ 源 ” 和 “ 目标 ” 类型必须匹配。
CIP 数据表写入 (CIP Data Table Write)	将数据写入另一个控制器。 “ 源 ” 和 “ 目标 ” 类型必须匹配。

重新组态 I/O 模块

使用 “ 模块重新组态 ” 信息可将新的组态信息发送到 I/O 模块。

重新组态期间：

- 输入模块继续将输入数据发送到控制器。
- 输出模块继续控制其输出设备。

“ 模块重新组态 ” 信息需要以下组态属性。

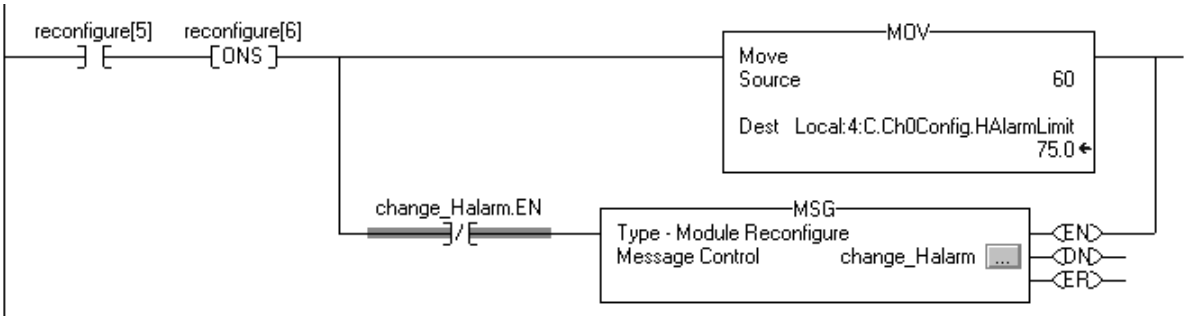
在此属性中	选择
信息类型 (Message Type)	模块重新组态 (Module Reconfigure)

示例： 请按以下步骤重新组态 I/O 模块。

1. 将模块组态标签的所需子元素设置为新值。
2. 向模块发送 “ 模块重新组态 ” 信息。

reconfigure[5] 置位时，对于插槽 4 中的本地模块，将上限报警设置为 60。“ 模块重新组态 ” 信息此时会将新报警值发送到模块。单脉冲触发指令可防止在 *reconfigure[5]* 使能时梯级向模块发送多条信息。

梯形图



结构化文本

```
IF reconfigure[5] AND NOT reconfigure[6]THEN
     Local:4:C.Ch0Config.HAlarmLimit := 60;
     IF NOT change_Halarm.EN THEN
          MSG(change_Halarm);
     END_IF;
END_IF;
     reconfigure[6] := reconfigure[5];
```

指定 “ **CIP** 通用 ” 信息

“ CIP 通用 ” 信息在 I/O 模块中执行特定操作。

如果要	在此属性中	键入或选择	
在数字量输出模块中执行脉冲测试	信息类型 (Message Type)	CIP 通用 (CIP Generic)	
	服务类型 (Service Type)	脉冲测试 (Pulse Test)	
	源 (Source)	类型为 INT [5] 的 *tag_name*	
		此数组包含	说明
		tag_name[0]	要测试的点的位屏蔽码 (一次只测试一个点)
		tag_name[1]	保留，保留为 0
		tag_name[2]	脉冲宽度 (数百微秒，通常为 20)
		tag_name[3]	ControlLogix I/O 的零交叉延迟 (数百微秒，通常为 40)
		tag_name[4]	检验延迟
	目标 (Destination)	保持空白	
复位数字输出模块中的电子保险回路	信息类型 (Message Type)	CIP 通用 (CIP Generic)	
	服务类型 (Service Type)	复位电子保险回路 (Reset Electronic Fuse)	
	源 (Source)	类型为 DINT 的 *tag_name* 此标签表示要复位保险回路的点的位屏蔽码。	
	目标 (Destination)	保持空白	
复位数字量输入模块中的锁存诊断	信息类型 (Message Type)	CIP 通用 (CIP Generic)	
	服务类型 (Service Type)	复位锁存诊断 (I) (Reset Latched Diagnostics (I))	
	源 (Source)	类型为 DINT 的 *tag_name* 此标签表示要复位诊断的点的位屏蔽码。	
复位数字量输出模块中的锁存诊断	信息类型 (Message Type)	CIP 通用 (CIP Generic)	
	服务类型 (Service Type)	复位锁存诊断 (O) (Reset Latched Diagnostics (O))	
	源 (Source)	类型为 DINT 的 *tag_name* 此标签表示要复位诊断的点的位屏蔽码。	

如果要	在此属性中	键入或选择
解锁模拟量输入模块的报警	信息类型 (Message Type)	CIP 通用 (CIP Generic)
	服务类型 (Service Type)	选择要解锁的报警。 • 解锁所有报警 (I) (Unlatch All Alarms (I)) • 解锁模拟量上限报警 (I) (Unlatch Analog High Alarm (I)) • 解锁模拟量上上限报警 (I) (Unlatch Analog High High Alarm (I)) • 解锁模拟量下限报警 (I) (Unlatch Analog Low Alarm (I)) • 解锁模拟量下下限报警 (I) (Unlatch Analog Low Low Alarm (I)) • 解锁速率报警 (I) (Unlatch Rate Alarm (I))
	实例 (Instance)	要解锁的报警的通道
解锁模拟量输出模块的警报	信息类型 (Message Type)	CIP 通用 (CIP Generic)
	服务类型 (Service Type)	选择要解锁的报警。 • 解锁所有报警 (O) (Unlatch All Alarms (O)) • 解锁上限报警 (O) (Unlatch High Alarm (O)) • 解锁下限报警 (O) (Unlatch Low Alarm (O)) • 解锁斜率报警 (O) (Unlatch Ramp Alarm (O))
	实例 (Instance)	要解锁的报警的通道

指定 PLC-5 信息

使用 PLC-5 信息类型可与 PLC-5 控制器通信。

选择此命令	如果要
PLC5 典型读取 (PLC5 Typed Read)	读取 16 位整型、浮点型或字符串型数据，并保持数据完整性。请参见 “ PLC5 典型读取 ” 和 “ PLC5 典型写入 ” 信息的数据类型 第 166 页。
PLC5 典型写入 (PLC5 Typed Write)	写入 16 位整型、浮点型或字符串型数据，并保持数据完整性。请参见 “ PLC5 典型读取 ” 和 “ PLC5 典型写入 ” 信息的数据类型 第 166 页
PLC5 字范围读取 (PLC5 Word Range Read)	读取 PLC-5 内存中某一连续范围的 16 位字，不考虑数据类型。 此命令从指定为 “ 源元素 ” (Source Element) 的地址处开始，按顺序读取所请求数目的 16 位字。 从指定为 “ 目标标签 ” (Destination Tag) 的地址处开始，存放来自 “ 源元素 ” (Source Element) 的数据。
PLC5 字范围写入 (PLC5 Word Range Write)	将 Logix5000 内存中某一连续范围的 16 位字写入 PLC-5 内存，不考虑数据类型。 此命令从指定为 “ 源标签 ” (Source Tag) 的地址处开始，按顺序读取所请求数目的 16 位字。 从 PLC-5 处理器中指定为 “ 目标元素 ” (Destination Element) 的地址处开始，存放来自 “ 源标签 ” (Source Tag) 的数据。

下表显示了可与“ PLC5 典型读取 ”(PLC5 Typed Read) 和“ PLC5 典型写入 ”(PLC5 Typed Write) 通信一起使用的数据类型。

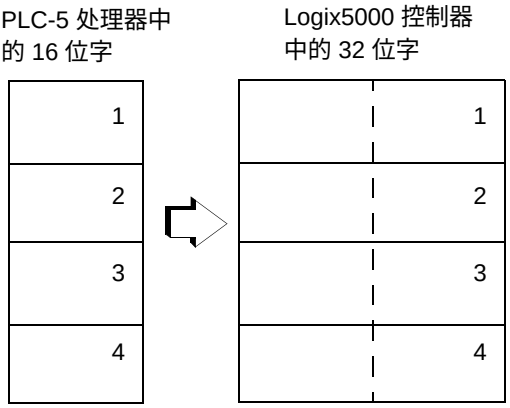
“ **PLC5** 典型读取 ” 和“ **PLC5** 典型写入 ” 信息的数据类型

对于此 PLC-5 数据类型	使用此 Logix5000 数据类型
B	INT
F	REAL
N	INT
	DINT(只有值≥ –32,768 且 ≤ 32,767 时，才能将 DINT 值写入 PLC-5 控制器。)
S	INT
ST	STRING

“ 典型读取 ”(Typed Read) 和“ 典型写入 ”(Typed Write) 命令还可用于 SLC 5/03 处理器 (OS303 及更高型号)、 SLC 5/04 处理器 (OS402 及更高型号) 和
SLC 5/05 处理器。

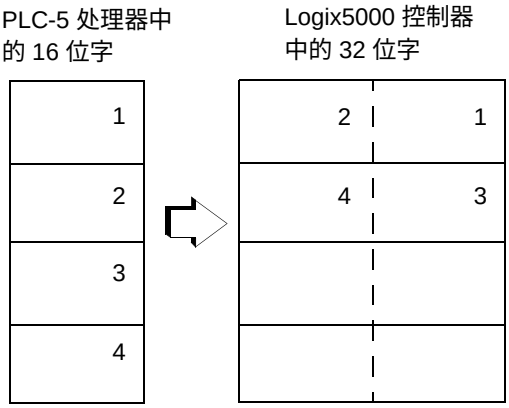
下图显示了典型命令与字范围命令之间的区别。该示例使用读取命令将 PLC-5 处理器中的命令读入 Logix5000 控制器。

典型读取命令



典型命令保持数据结构和值不变。

字范围读取命令



字范围命令连续填充目标标签。数据结构和值随目标数据类型而变化。

指定 SLC 信息

使用 SLC 通信类型与 SLC 和 MicroLogix 控制器通信。下表显示了指令允许访问的数据类型。同时还显示了对应的 Logix5000 数据类型。

对于此 SLC 或 MicroLogix 数据类型	使用此 Logix5000 数据类型
F	REAL
L(MicroLogix 1200 和 1500 控制器)	DINT
N	INT

指定块传送信息

块传送信息类型用于通过通用远程 I/O 网络与块传送模块通信。

如果要	选择此命令
从块传送模块读取数据 此信息类型替代 BTR 指令	块传送读取 (Block-Transfer Read)
将数据写入块传送模块 此信息类型替代 BTW 指令	块传送写入 (Block-Transfer Write)

要组态块传送信息，请遵循以下原则：

- 除 MESSAGE、 AXIS 和 MODULE 结构外，源标签 (用于 BTW) 和目标标签 (用于 BTR) 必须足够大，以便可以接受请求的数据。
- 指定要发送或接收的 16 位整数 (INT) 的数目。可以指定 0…64 之间的整数。

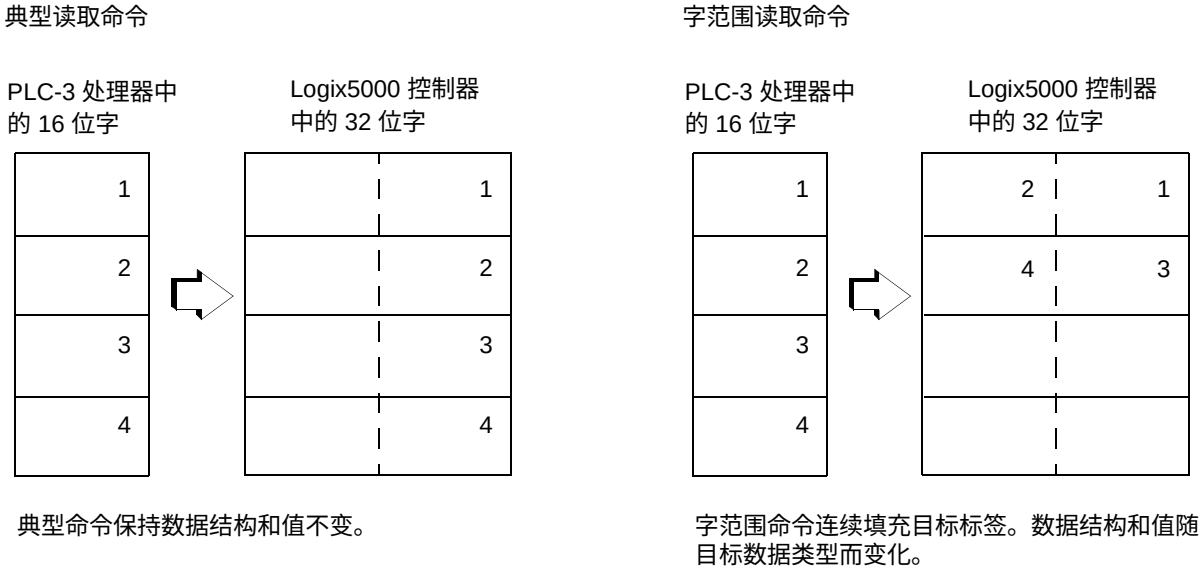
如果希望	则指定
块传送模块确定要发送的 16 位整数的数目 (BTR)	0 作为元素数目
控制器发送 64 个整数 (BTW)	

指定 **PLC-3** 信息

PLC-3 信息类型专为 PLC-3 处理器而设计。

选择此命令	如果要
PLC3 典型读取 (PLC3 Typed Read)	读取整型或 REAL 型数据。 对于整数，此命令可从 PLC-3 处理器中读取 16 位整数，然后将它们存储在 Logix5000 控制器中的 SINT、 INT 或 DINT 数据数组中，并保持数据完整性。 此命令还可从 PLC-3 中读取浮点型数据，并将其存放在 Logix5000 控制器中的 REAL 数据类型标签中。
PLC3 典型写入 (PLC3 Typed Write)	写入整型或 REAL 型数据。 此命令可将 SINT 或 INT 数据写入 PLC-3 整型文件，并保持数据完整性。可以写入 DINT 数据，只要该数据在 INT 数据类型范围内 (–32,768 ≥ 数据 ≤ 32,767)。 此命令还可将 Logix5000 控制器中的 REAL 型数据写入 PLC-3 浮点型文件。
PLC3 字范围读取 (PLC3 Word Range Read)	读取 PLC-3 内存中某一连续范围的 16 位字，不考虑数据类型。 此命令从指定为“ 源元素 ” (Source Element) 的地址处开始，按顺序读取所请求数目的 16 位字。 从指定为“ 目标标签 ” (Destination Tag) 的地址处开始，存放来自“ 源元素 ” (Source Element) 的数据。
PLC3 字范围写入 (PLC3 Word Range Write)	将 Logix5000 内存中某一连续范围的 16 位字写入 PLC-3 内存，不考虑数据类型。 此命令从指定为“ 源标签 ” (Source Tag) 的地址处开始，按顺序读取所请求数目的 16 位字。 从 PLC-3 处理器中指定为“ 目标元素 ” (Destination Element) 的地址处开始，存放来自“ 源标签 ” (Source Tag) 的数据。

下图显示了典型命令与字范围命令之间的区别。该示例使用读取命令将 PLC-3 处理器中的命令读入 Logix5000 控制器。



典型命令保持数据结构和值不变。

字范围命令连续填充目标标签。数据结构和值随目标数据类型而变化。

指定 **PLC-2** 信息

PLC-2 信息类型专为 PLC-2 处理器而设计。

选择此命令	如果要
PLC2 非保护的读取 (PLC2 Unprotected Read)	从 PLC-2 数据表的任意区域或另一个处理器的 PLC-2 兼容性文件中读取 16 位字。
PLC2 非保护的写入 (PLC2 Unprotected Write)	将 16 位字写入 PLC-2 数据表的任意区域或另一个处理器的 PLC-2 兼容性文件。

该信息传送使用 16 位字，因此请确保 Logix5000 标签正确存放所传送的数据 (通常以 INT 数组的形式存储)。

MSG 组态示例

下面的示例显示了不同控制器组合的源标签和目标标签以及源元素和目标元素。

下表说明了来源于 Logix5000 控制器并写入到另一个控制器的 MSG 指令的路径。

信息路径	源和目标示例	
Logix5000 → Logix5000	源标签	*array_1[0]*
	目标标签	*array_2[0]*
	可以为源标签使用别名标签 (在最初的 Logix5000 控制器中)。 不能为目标标签使用别名。目标标签必须是基本数据标签。	
Logix5000 → PLC-5 Logix5000 → SLC	源标签	*array_1[0]*
	目标元素	*N7:10*
	可以为源标签使用别名标签 (在最初的 Logix5000 控制器中)。	
Logix5000 → PLC-2	源标签	*array_1[0]*
	目标元素	*010*

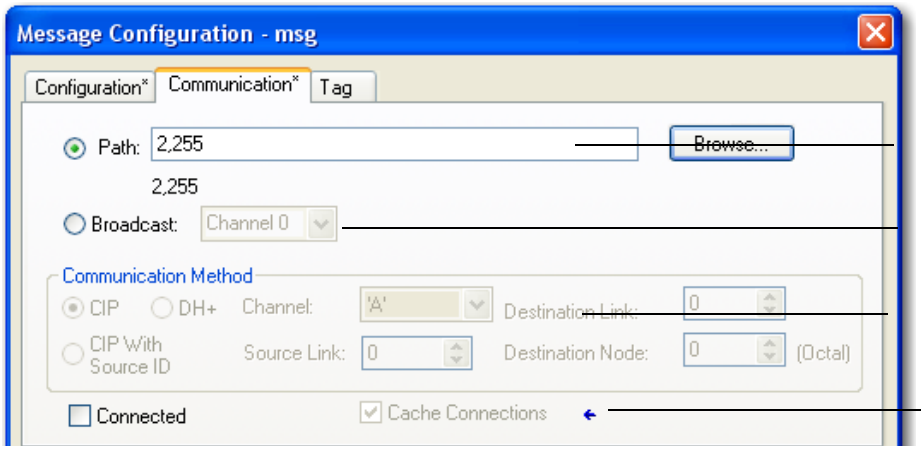
下表说明了来源于 Logix5000 控制器并从另一个控制器读取的 MSG 指令的路径。

信息路径	源和目标示例	
Logix5000 → Logix5000	源标签	*array_1[0]*
	目标标签	*array_2[0]*
	不能为源标签使用别名标签。源标签必须是基本数据标签。 可以为目标标签使用别名标签 (在最初的 Logix5000 控制器中)。	
Logix5000 → PLC-5 Logix5000 → SLC	源元素	*N7:10*
	目标标签	*array_1[0]*
	可以为目标标签使用别名标签 (在最初的 Logix5000 控制器中)。	
Logix5000 → PLC-2	源元素	*010*
	目标标签	*array_1[0]*

指定通信详细信息

您通常会在梯形逻辑程序或结构化文本程序中设置广播。在梯形逻辑中，可以添加梯级，然后单击 MSG 属性来访问 “ 信息组态 ” (Message Configuration) 对话框并设置新信息。在结构化文本中，可键入 MSG(aMsg)，然后右键单击 aMsg 获取 “ 信息组态 ” (Message Configuration) 对话框并对信息组态。

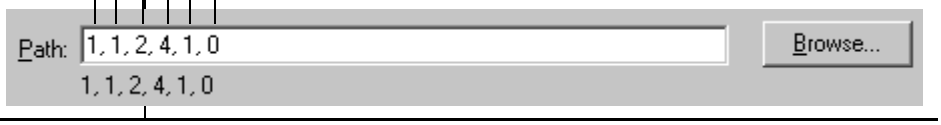
要组态 MSG 指令，可在 “ 通信 ” (Communication) 页面中指定这些详细信息。



指定路径

路径显示了信息到达目标所经过的路线。它使用控制器 I/O 组态中的名称、键入的数字或者两者都使用。您可通过使用广播按钮设置默认路径，该按钮必须使用系统协议和信息类型使能。

如果	则
控制器的 I/O 组态中包含获取信息的模块。	使用 “ 浏览 ” (Browse) 选择该模块。
控制器的 I/O 组态中只有本地通信模块。	1. 使用 “ 浏览 ” (Browse) 选择本地通信模块。 2. 键入路径的其余部分。
控制器的 I/O 组态中不包含需要用于信息的任何模块。	键入路径。

示例	
控制器的 I/O 组态中包含获取信息的模块。	单击“ 浏览 ”(Browse) 并选择模块。 
控制器的 I/O 组态中只有本地通信模块。	转到本地通信模块。 从 EtherNet/IP 端口出去… ... 到达地址 10.10.10.10。 通过背板… ... 到达插槽 0 中的模块。 
控制器的 I/O 组态中不包含需要用于信息的任何模块。	通过背板… ... 到达插槽 1 中的本地通信模块 从 ControlNet 端口出去… ... 到达节点 4 通过背板… ... 到达插槽 0 中的模块。 

要键入路径，请使用如下格式：

端口 , 下一个地址 , 端口 , 下一个地址 , …

位于	是	
	这种网络	键入
端口	背板	1
	DF1(串行，串行通道 0)	2
	ControlNet	
	EtherNet/IP	
	DH+ 通道 A	
	DH+ 通道 B	3
	DF1 通道 1(串行 通道 1)	
下一个地址	背板	模块的槽号
	DF1(串行)	站地址 (0-254)
	ControlNet	节点编号 (1-99 十进制)
	DH+	8# 后跟节点编号 (1-77 八进制) 例如，要指定八进制节点地址 37，请键入 8#37。
	EtherNet/IP	可以使用以下任意一种格式指定 EtherNet/IP 网络中的模块： IP 地址 (例如 10.10.10.10) IP 地址 : 端口 (例如 10.10.10.10:24) DNS 名称 (例如 tanks) DNS 名称 : 端口 (例如 tanks:24)

广播按钮

这一功能自 RSLogix 5000 软件版本 18 起开始提供，该功能可以定义信息发送到目的地所需的路径和信息类型。您仍可键入路径，如 “ 2,255 ”。但是，由于以下两个原因，手动输入路径会存在问题：

- 输入到路径中的数字可能比较混乱。
- 容易出错，因为用户可能会忘记设置其它条件，如系统协议和信息类型。

“ 广播 ” (Broadcast) 按钮在使能后，可以使您通过在组合框中选择可用通道来设置默认路径。组合框中列出的通道数取决于当前控制器。

默认情况下，“ 路径 ” (Path) 单选按钮在 “ 通信 ” (Communication) 页面中处于激活状态 (单选按钮中存在圆点)。执行以下步骤可使能 “ 广播 ” (Broadcast) 按钮并选择通道来设置信息的默认路径。

1. 在控制器项目管理器中，右键单击 “ 控制器 ” (Controller) 并选择 “ 属性 ” (Properties)。

   将显示 “ 控制器属性 ” (Controller Properties) 对话框。

2. 单击 “ 系统协议 ” (System Protocol) 页面。

3. 在 “ 协议 ” (Protocol) 框中选择 “ DF1 主站 ” (DF1 Master)。

   轮询模式默认为 “ 基于信息 ” (Message Based)(从站可以发起信息)。

4. 单击 “ 确定 ” (OK)。

5. 在梯形逻辑中，单击 MSG 标签内的框。

   将显示 “ 信息组态 ” (Message Configuration) 对话框，同时打开 “ 组态 ” (Configuration) 页面。

6. 在 “ 信息类型 ” (Message Type) 框中，选择 “ CIP 数据表写入 ” (CIP Data Table Write)。(1)

请参见第 161 页，查看示例。

(1) 要使能 “ 广播 ” (Broadcast) 按钮，请参见第 175 页的系统协议指令。

7. 单击 “ 确定 ” (OK)。

   “ 通信 ” (Communication) 页面上即可使能 “ 广播 ” (Broadcast) 按钮。

8. 单击 “ 通信 ” (Communication) 页面。

9. 下拉 “ 广播 ” (Broadcast) 按钮，在组合框中选择通道。组合框中的通道数取决于控制器。

   选择通道 0 或 1 时，“ 信息组态 ” (Message Configuration) 对话框上的相应信息路径会默认设置为 2,255(通道 0) 或 3,255(通道 1)。 “ 路径 ” (Path) 灰显，从而不允许手动输入路径值。

10. 单击 “ 确定 ” (OK)。

“ 系统协议 ” 页面组态

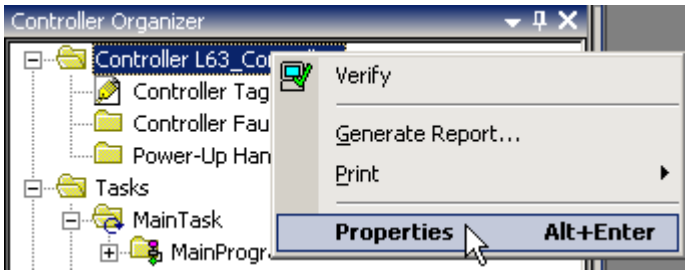
要通过 RSLogix 5000 编程软件在 ControlLogix 中运行广播，必须在 “ 控制器属性 ” (Controller Properties) 对话框中组态 “ 系统协议 ” (System Protocol) 页面。协议必须与 “ 信息组态 ” (Message Configuration) 对话框中的 “ 写 ” (write) 信息类型兼容。

有关 “ 信息类型 ” (Message Type) 框中的示例，请参见第 161 页。

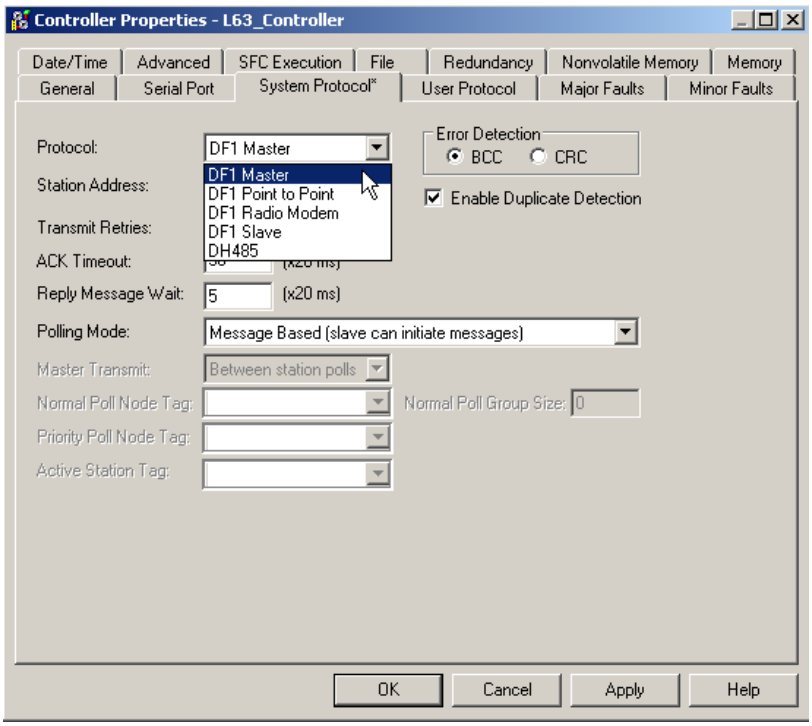
按以下步骤操作可设置与广播功能兼容的系统协议。

1. 在 RSLogix 5000 编程软件中创建或打开现有控制器。

2. 在控制器项目管理器中，右键单击控制器名称并选择 “ 属性 ” (Properties)。



将显示“ 控制器属性 ”(Controller Properties) 对话框。



3. 单击“ 系统协议 ”(System Protocol) 页面。
4. 从“ 协议 ”(Protocol) 框中选择协议。

重要事项

“ 信息组态 ”(Message Configuration) 页面对话框中的“ 信息类型 ”(Message Type) 框必须为写类型才能与系统协议兼容。否则，“ 广播 ”(Broadcast) 按钮将被禁止。

5. 为下述每个协议在“ 系统协议 ”(System Protocol) 页面上输入信息。

DF-1 主站

主题	说明
协议 (Protocol)	DF-1 主站
站地址 (Station Address)	键入控制器站地址编号
重试传送 (Transmit Retries)	3
ACK 超时 (ACK Timeout)	50
应答信息等待 (Reply Message Wait)	5

DF-1 主站

主题	说明
轮询模式 (Polling Mode)	选择以下模式： • “ 基于信息 ” (Message Based)，使用信息指令轮询从站 • “ 从站可以发起信息 ” (Slave can initiate message)，进行从站到从站的广播 • “ 标准 ” (Standard)，定期轮询从站
错误检测 (Error Detection)	BCC
重复检测 (Duplicate Detection)	使能 (勾选)

DF-1 从站

主题	说明
协议 (Protocol)	DF-1 从站
站地址 (Station Address)	键入控制器站地址编号
重试传送 (Transmit Retries)	3
从站轮询超时 (Slave Poll Timeout)	3000
EOT 抑制 (EOT Suppression)	禁止 (取消勾选)
错误检测 (Error Detection)	BCC
重复检测 (Duplicate Detection)	使能 (勾选)

DF-1 无线调制解调器

主题	说明
协议 (Protocol)	DF-1 从站
站地址 (Station Address)	键入控制器站地址编号
使能存储和转发 (Enable Store and Forward)	使能该框 (复选框) 来使用存储和转发标签
错误检测	BCC

6. 单击 “ 确定 ” (OK)。

对于块传送

对于块传送信息，请将以下模块添加到控制器的 I/O 组态中。

通过以下网络进行的块传送	将以下模块添加到 I/O 组态中
ControlNet	• 本地通信模块 (例如， 1756-CNB 模块) • 远程适配器模块 (例如， 1771-ACN 模块)
通用远程 I/O	• 本地通信模块 (例如， 1756-DHRIO 模块) • 远程适配器模块 (例如， 1771-ASB 模块)，每个机架或机架的一部分对应一个 • 块传送模块 (可选)

指定通信方法或模块地址

通过下表可为信息选择通信方法或模块地址。

如果目标设备为	则选择	并指定	
Logix5000 控制器	**CIP**	不需要指定其它内容。	
通过 EtherNet/IP 网络通信的 PLC-5 控制器			
通过 ControlNet 网络通信的 PLC-5 控制器			
SLC 5/05 控制器			
通过 DH+ 网络通信的 PLC-5 控制器	**DH+**	通道 (Channel)	连接到 DH+ 网络的 1756-DHRIO 模块的通道 A 或 B。
通过 DH+ 网络通信的 SLC 控制器		源链接 (Source Link)	1756-DHRIO 模块的路由表中分配给控制器背板的链接 ID。 (路由表中的源节点自动给出控制器的槽号。)
PLC-3 处理器		目标链接 (Destination Link)	目标设备所在的远程 DH+ 链接的链接 ID。
PLC-2 处理器		目标节点 (Destination Node)	目标设备的站地址，采用八进制形式。
		如果只有一个 DH+ 链接，且未使用 RSLinx 软件来为远程链接组态 DH/RIO 模块，则将 “ 源链接 ” (Source Link) 和 “ 目标链接 ” (Destination Link) 均指定为 0。	

如果目标设备为	则选择	并指定	
工作站上的应用程序，它借助 RSLinx 软件接收通过 EtherNet/IP 或 ControlNet 网络路由的未经请求的信息	带源 **ID 的 CIP (CIP with Source ID)** (这使应用程序可以接收来自控制器的数据。)	源链接 (Source Link)	RSLinx 软件中主题的远程 ID。
		目标链接 (Destination Link)	RSLinx 软件中设置的虚拟链接 ID (0…65535)。
		目标节点 (Destination Node)	应用程序向 RSLinx 提供的目标 ID(0…77 八进制)。对于 RSLinx 软件中的 DDE 主题，请使用 77。
		ControlLogix 控制器的槽号会用作 “ 源节点 ” (Source Node)。	
通用远程 I/O 网络通信的块传送模块	RIO	通道 (Channel)	连接到 RIO 网络的 1756-DHRIO 模块的通道 A 或 B。
		机架 (Rack)	模块的机架编号 (八进制)。
		组 (Group)	模块的组编号。
		插槽 (Slot)	模块所在插槽的编号。
通过 ControlNet 网络通信的块传送模块	ControlNet	插槽 (Slot)	模块所在插槽的编号。

选择缓存连接选项

根据用户组态 MSG 指令的方式，它可能使用连接来发送或接收数据。

信息类型	通信方法	是否使用连接
CIP 数据表读取或写入	→	由用户选择(1)
PLC-2、 PLC-3、 PLC-5 或 SLC(所有类型)	CIP 带源 ID 的 CIP (CIP with Source ID)	
	DH+	X
CIP 通用	→	由用户选择(2)
块传送读取或写入	→	X

(1) CIP 数据表读取或写入信息可以选连接或非连接。但是对于大多数应用，我们建议将 CIP 数据表读取或写入信息保留为连接。

(2) CIP 通用信息可以选连接或非连接。但是对于大多数应用，我们建议将 CIP 通用信息保留为非连接。

如果 MSG 指令使用连接，用户可以选择信息传送完毕保留连接打开 (缓存) 或关闭连接。

如果	则
缓存连接	执行完 MSG 指令后，连接保留在打开状态。这可以优化执行时间。每次信息执行时都要打开连接会增加执行时间。
不缓存连接	执行完 MSG 指令后，连接关闭。这可以将连接释放以作它用。

控制器对缓存连接数具有以下限制。

如果软件和固件版本为	则可以缓存
11.*x* 或更低版本	• 最多 16 个连接的块传送信息。 • 最多 16 个连接的其它类型的信息。
12.*x* 或更高版本	最多 32 个连接。

如果有多个信息发送到同一设备，则这些信息将能够共享一个连接。

如果 MSG 指令发送到	并且这些指令	则
不同设备	——►	每个 MSG 指令使用一个连接。
相同设备	同时使能	每个 MSG 指令使用一个连接。
	不同时使能	MSG 指令使用一个连接和一个缓存的缓冲区。它们共享连接和缓冲区。

举 例

共享一个连接

如果控制器向同一模块交替发送块传送读取信息和块传送写入信息，则这两个信息视为一个连接。对这两条信息的缓存在缓存列表中视为一次缓存。

指导原则

对 MSG 指令进行计划和编程时，请遵循以下指导原则 .

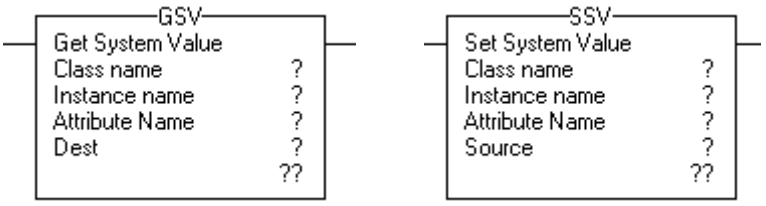
指导原则	详细信息
1. 为每个 MSG 指令创建一个控制标签。	每个 MSG 指令都需要具有自己的控制标签。
	• 数据类型 = MESSAGE
	• 范围 = 控制器
	• 标签不能是数组的一部分或用户定义的数据类型。
2. 使源和 / 或目标数据保持在控制器范围内。	MSG 指令只能访问 “ 控制器标签 ” (Controller Tags) 文件夹中的标签 (控制器范围)。
3. 如果 MSG 发送到使用 16 位整数的设备，则在 MSG 中使用 INT 数据缓冲区，并在整个项目中使用 DINT 数据缓冲区。	如果信息发送到使用 16 位整数的设备 (如 PLC-5® 或 SLC 500® 控制器)，并且它传输的是整数 (而非 REAL 型数据)，则在信息中使用 INT 数据缓冲区，并在整个项目中使用 DINT 数据缓冲区。
	这样可提高项目效率，因为在使用 32 位整数 (DINT) 时， Logix 控制器的执行效率更高，使用的内存也更少。
	要在 INT 与 DINT 之间进行转换，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。
4. 执行最频繁的缓存连接 MSG 指令。	缓存连接用于那些执行最频繁的 MSG 指令，指令数目最多可达到控制器版本允许的最大值。
	这样可缩短执行时间，因为控制器不必在每次执行信息时都要打开连接。
5. 如果要同时使能 16 条以上 MSG 指令，请使用某种管理策略。	如果同时使能 16 条以上 MSG 指令，则一些 MSG 指令在进入队列时可能产生延迟。要保证每条信息都得以执行，请使用以下一个选项：
	• 按顺序使能每条信息。
	• 按组使能信息。
	• 对信息进行编程，使其与多个设备通信。有关详细信息，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。 .
	• 对逻辑进行编程，以协调信息的执行。有关详细信息，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。
6. 使非连接和非缓存的 MSG 的数目小于非连接缓冲区的数目。	控制器可以有 10...40 个非连接的缓冲区。默认数目为 10。
	• 如果在指令离开信息队列时，所有非连接的缓冲区都在使用中，则指令会出错并且不会传送数据。
	• 可以增加非连接的缓冲区的数目 (最多 40 个)，但要继续按原则 5 操作。
	• 要增加非连接的缓冲区的数目，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。

获取系统值 (GSV) 和设置系统值 (SSV)

GSV/SSV
指令可获取和设置对象中存储的控制器系统数据。

操作数：

梯形图



操作数	类型	格式	说明
类名称 (Class name)		名称	对象的名称。
实例名称 (Instance name)		名称	特定对象的名称 (如果对象需要名称)。
属性名称 (Attribute Name)		名称	对象的属性。 数据类型取决于选择的属性。
目标 (GSV) (Destination (GSV))	SINT INT DINT REAL 结构	标签	属性数据的目标。
源 (SSV) (Source (SSV))	SINT INT DINT REAL 结构	标签	包含要复制到属性的数据的标签。

结构化文本

```
GSV(ClassName,InstanceName,AttributeName,Dest);
SSV(ClassName,InstanceName,AttributeName,Source);
```

这些操作数与梯形图 GSV
和 SSV 指令的操作数相同。

说明： GSV/SSV 指令用于获取和设置对象中存储的控制器系统数据。控制器将系统数据存储在对象中。没有像 PLC-5 处理器中那样的状态文件。

使能后， GSV 指令会检索指定信息，并将其放在目标位置。使能后，SSV 指令会使用来自源位置的数据设置指定属性。

键入 GSV/SSV 指令后，编程软件会为每条指令显示有效的对象类、对象名称和属性名称。对于 GSV 指令，可以获取所有可用属性的值。对于 SSV 指令，只显示允许设置的属性 (SSV)。

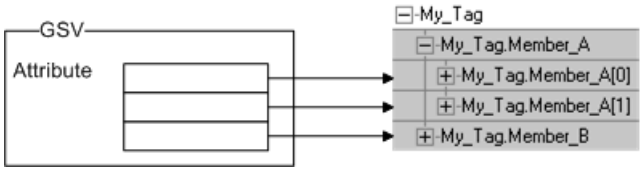
注 意

使用 GSV 和 SSV 指令时要小心谨慎。修改对象可能会导致控制器出现意外操作或造成人员损伤。

必须测试并确认那些您想要修改但指令不能修改的数据。

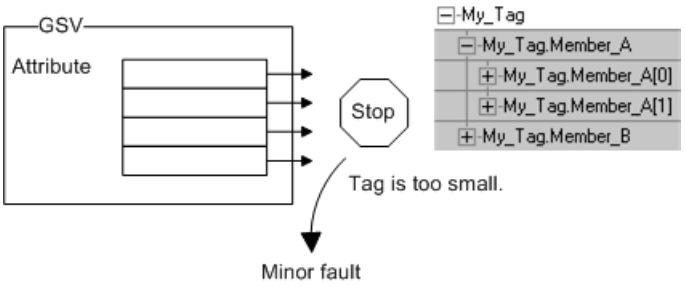
GSV 和 SSV 指令可越过标签的某个子元素而进入其它子元素进行读取或写入。如果标签太小，则指令不写入或读取数据。它们会记录次要故障。

示例 **1**



Member_A 对于该属性而言太小。因此 GSV 指令将最后一个值写入 Member_B。

示例 **2**



My_Tag 对于该属性而言太小。因此 GSV 指令会停止，并记录一个次要故障。

“ GSV/SSV 对象 ”部分显示了每个对象的属性及相关的数据类型。例如， Program 对象的 MajorFaultRecord 属性需要 DINT[11] 数据类型。

算术状态标志： 不受影响

故障条件：

出现次要故障的条件	故障类型	故障代码
对象地址无效	4	5
指定了不支持 GSV/SSV 的对象	4	6
属性无效	4	6
没有为 SSV 指令提供足够的信息	4	6
GSV 的目标地址不够大，无法保存请求的数据	4	7

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
执行指令	获取或设置指定值。	获取或设置指定值。
后扫描	梯级输出条件设置为假。	不执行任何操作。

GSV/SSV 对象

键入 GSV/SSV 指令时，需要指定要访问的对象及其属性。某些情况下，同一类型的对象有多个实例，因此可能还必须指定对象名称。例如，应用项目中可以有多个任务。各个任务都有自己的 TASK 对象，可以通过任务名称来访问该对象。

注 意

对于 GSV 指令，只会将指定大小的数据复制到目标位置。例如，如果属性规定为 SINT，而目标地址为 DINT，则只更新 DINT 目标的低 8 位，其余 24 位保持不变。

可以访问以下对象。

GSV/SSV 对象

有关此对象的信息	请参见此页面或出版物
AddOnInstructionDefinition	第 4 章 -186
Axis	请参见 《SERCOS 运动控制组态和启动用户手册》 (出版号 MOTION-UM001)。
Axis_Consumed	
Axis_Generic	
Axis_Generic_Drive	
Servo	
Servo_Drive	
Virtual	
Axis_CIP_Drive	请参见 《CIP 运动控制组态和启动用户手册》 (出版号 MOTION-UM003)。
Controller	第 4 章 -187
ControllerDevice	第 4 章 -188
CoordinateSystem	请参见 《运动控制坐标系用户手册》 (出版号 MOTION-UM002)。
CST	第 4 章 -190
DF1	第 4 章 -191
FaultLog	第 4 章 -194
Message	第 4 章 -195
Module	第 4 章 -196
MotionGroup	请参见 《运动控制参考手册》 (出版号 MOTION-RM001)。
Program	第 4 章 -198
Redundancy	请参见 《ControlLogix 增强型冗余系统用户手册》 (出版号 1756-UM535)。

GSV/SSV 对象

Routine	第 4 章 -200
SafetyAddonInstructionDefinition、SafetyController、 SafetyProgram、SafetyRoutine、 SafetyTask。	请参见 《GuardLogix 控制器用户手册》 (出版号 1756-UM020)。
SerialPort	第 4 章 -202
Task	第 4 章 -203
TimeSynchronization	请参见 《集成架构和 CIP 同步组态应用技术》 (出版号 IA-AT003)。
WallClockTime	第 4 章 -205

AddOnInstructionDefintion 属性

AddOnInstructionDefinition 对象允许用户为常用逻辑集自定义指令、为该逻辑提供通用接口以及为指令提供相关文档。

有关详细信息，请参见 《Logix5000 控制器用户自定义指令编程手册》 (出版号 1756-PM010)。

属性	数据类型	标准任务内的指令	安全任务内的指令	说明
LastEditDate	LINT	GSV	无	上次编辑用户自定义指令定义的日期和时间戳。
MajorRevision	DINT	GSV	无	用户自定义指令的主版本号。
MinorRevision	DINT	GSV	无	用户自定义指令的次版本号。
Name	字符串	GSV	GSV	用户自定义指令的名称。
RevisionExtendedText	字符串	GSV	无	说明用户自定义指令版本的文本。
SafetySignatureID	DINT	GSV	无	安全项目中，用户自定义指令定义的 ID 号、日期和时间戳。
SignatureID	DINT	GSV	无	用户自定义指令定义的 32 位标识号。
Vendor	字符串	GSV	无	创建用户自定义指令的供应商。

Controller 属性

Controller 对象提供有关控制器执行的状态信息。

属性	数据类型	指令	说明
CanUseRPIFrom Producer	DINT	GSV	识别是否使用了生产者指定的 RPI。 值 含义 0 没有使用生产者指定的 RPI 1 使用生产者指定的 RPI
ControllerLog Execution Modification Count	DINT	GSV SSV	由于程序 / 任务属性更改、在线编辑或控制器时间片更改而产生的控制器日志条目数。还可以将其组态为包含由于强制而产生的日志条目。如果 RAM 进入故障状态，该数目将复位。该数目不封顶在最大 DINT 值，可以发生翻转。
ControllerLog TotalEntryCount	DINT	GSV SSV	自上次固件升级后的控制器日志条目数。如果 RAM 进入故障状态，该数目将复位。该数目封顶在最大 DINT 值。
DataTablePad Percentage	INT	GSV	留出的空闲数据表内存的百分比 (0...100)。
IgnoreArrayFaults DuringPostScan	SINT	GSV SSV	用于组态 SFC Action 后扫描时激发的选择故障抑制。仅在 SFC 组态为自动复位时才有效。 值 含义 0 执行后扫描过程中不抑制故障 (默认行为 - 推荐)。 1 在后扫描 SFC Action 时自动抑制主要故障 4/20(数组下标过大) 和 4/83(值超出范围)。 抑制故障时，控制器会使用内部故障处理程序自动清除故障。这会导致出现故障的指令被跳过，同时继续执行随后的指令。 由于故障处理程序是内部的，因此不必组态故障处理程序来获得此行为。实际上，即使组态故障处理程序，被抑制的故障也不会触发该处理程序。
InhibitAutomatic FirmwareUpdate	BOOL	GSV SSV	识别是否使能固件监控。 值 含义 0 不执行固件监控 1 执行固件监控
KeepTestEditsOn Switchover	SINT	GSV	识别在控制器切换时是否保持测试编辑。 值 含义 0 切换时自动取消测试编辑 1 切换时继续测试编辑
Name	字符串	GSV	控制器的名称。
Redundancy Enabled	SINT	GSV	识别是否将控制器组态为冗余。 值 含义 0 没有组态为冗余 1 组态为冗余

属性	数据类型	指令	说明
ShareUnused TimeSlice	INT	GSV SSV	识别连续任务和后台任务如何共享未用时间片。 值　含义 0　操作系统不对连续任务进行控制，即使后台任务已完成 (默认值) 1　连续任务在后台任务完成时仍然运行。 2　操作系统不对任何任务进行控制，即使后台任务已完成
TimeSlice	INT	GSV SSV	分配给通信可用 CPU 的百分比 (10...90)。钥匙开关处于运行位置时，不能更改该值。

ControllerDevice 属性

ControllerDevice 对象标识控制器的物理硬件。

属性	数据类型	指令	说明
DeviceName	SINT[33]	GSV	用于识别控制器和内存卡目录号的 ASCII 字符串。第一个字节包含数组字符串中返回的 ASCII 字符的个数。
ProductCode	INT	GSV	识别控制器的类型： 值　含义 15SoftLogix5800 49 带 DriveLogix5725 的 PowerFlex 52 带 DriveLogix5730 的 PowerFlex 53Emulator 541756-L61 ControlLogix 551756-L62 ControlLogix 561756-L63 ControlLogix 571756-L64 ControlLogix 641769-L31 CompactLogix 651769-L35E CompactLogix 671756-L61S GuardLogix 681756-L62S GuardLogix 691756-LSP GuardLogix 721768-L43 CompactLogix 741768-L45 CompactLogix 761769-L32C CompactLogix 771769-L32E CompactLogix 801769-L35CR CompactLogix 851756-L65 ControlLogix 861756-L63S GuardLogix 871769-L23E-QB1 CompactLogix 881769-L23-QBFC1 CompactLogix 891769-L23E-QBFC1 CompactLogix
ProductRev	INT	GSV	识别当前产品版本。显示内容采用十六进制形式。低字节包含主版本；高字节包含次版本。
SerialNumber	DINT	GSV	设备的序列号。序列号在制造设备时分配。

属性	数据类型	指令	说明
Status	INT	GSV	设备状态位控制器状态位 **7...4**　含义　　**13...12**含义 0000　保留　　01　钥匙开关处于运行位置 0001　正在进行闪存更新 10　钥匙开关处于编程位置 0010　保留　　11　钥匙开关处于远程位置 0011　保留 0100　闪存错误　　**15...14**含义 0101　故障模式　　01　控制器正在改变模式 0110　运行　　10　调试模式 (如果控制器处于运行模式) 0111　编程 故障状态位 **11...8**　含义 0001　可恢复的次要故障 0010　不可恢复的次要故障 0100　可恢复的主要故障 1000　不可恢复的主要故障
类型	INT	GSV	识别如同控制器的设备。控制器 = 14。
供应商	INT	GSV	识别设备供应商。 Allen-Bradley = 0001。

CST 属性

协调系统时间 (CST) 对象为一个机架中的设备提供协调系统时间。

属性	数据类型	指令	说明
CurrentStatus	INT	GSV	协调系统时间的当前状态。 位　含义 0　计时器硬件发生故障：设备的内部计时器硬件处于故障状态 1　使能斜升：计时器低 16+ 位的当前值斜升至请求值，而不是突降到低位值。 2　系统时间主站：CST 对象是 ControlLogix 系统中的主时间源 3　已同步：CST 对象的 64 位 CurrentValue 由主 CST 对象通过系统时间更新进行了同步 4　本地网络主站：CST 对象是本地网络主时间源 5　中继模式：CST 对象正在时间中继模式下操作 6　检测到重复主站：检测到重复的本地网络时间主站。对于时间相关节点，该位始终为 0。 7　未使用 8-9　00 = 时间相关节点 01 = 时间主站节点 10 = 时间中继节点 11 = 未使用 10-15　未使用
CurrentValue	DINT[2]	GSV	计时器的当前值。 DINT[0] 包含低 32 位； DINT[1] 包含高 32 位。计时器源会进行调整，以与更新服务中提供的值和来自本地通信网络同步的值匹配。根据 CurrentStatus 属性中的报告，该调整可以是斜升到请求的值，也可以是立即设置为请求值。

DF1 属性

DF1 对象提供可以为串口组态的 DF1 通信驱动程序的接口。

属性	数据类型	指令	说明
ACKTimeout	DINT	GSV	等待信息传输应答的时间 (仅限点对点通信和主站通信)。有效值为 0-32,767。延迟以 20 毫秒为周期计数。默认值为 50(1 秒)。
Diagnostic Counters	INT[19]	GSV	DF1 通信驱动程序的诊断计数器数组。

字偏移量	**DF1** 点对点	**DF1** 从站	主站
0	签名 (0x0043)	签名 (0x0042)	签名 (0x0044)
1	调制解调器位数	调制解调器位数	调制解调器位数
2	发送的数据包数	发送的数据包数	发送的数据包数
3	接收的数据包数	接收的数据包数	接收的数据包数
4	未送达的数据包数	未送达的数据包数	未送达的数据包数
5	未使用	重试的信息数	重试的信息数
6	接收的 NAK 数	接收的 NAK 数	未使用
7	接收的 ENQ 数	接收的轮询包数	未使用
8	NAK 的坏数据包数	未 ACK 的坏数据包数	未 ACK 的坏数据包数
9	无内存已发送 NAK	无内存未 ACK	未使用
10	接收的重复数据包数	接收的重复数据包数	接收的重复数据包数
11	接收的错误字符数	未使用	未使用
12	DCD 恢复计数	DCD 恢复计数	DCD 恢复计数
13	丢失的调制解调器计数	丢失的调制解调器计数	丢失的调制解调器计数
14	未使用	未使用	优先级扫描时间最大值
15	未使用	未使用	优先级扫描时间最后值
16	未使用	未使用	正常扫描时间最大值
17	未使用	未使用	正常扫描时间最后值
18	发送的 ENQ 数	未使用	未使用

Duplicate Detection	SINT	GSV	使能重复信息检测。 值 含义 0 禁止重复信息检测 非零 使能重复信息检测
Embedded ResponseEnable	SINT	GSV	使能嵌入响应功能 (仅限点对点通信)。 值 含义 0 仅在收到一个响应后启动 (默认值) 1 无条件使能
EnableStoreFwd	SINT	GSV	使能接收信息时的存储转发行为。 值 含义 0 不转发信息 非零 请参见接收信息时的存储转发表 (默认值)
ENQTransmit Limit	SINT	GSV	ACK 超时后要发送的查询 (ENQ) 的数目 (仅限点对点通信)。有效值为 0-127。默认值为 3。

属性	数据类型	指令	说明
EOTSuppression	SINT	GSV	使能作为对轮询包的响应的 EOT 传输抑制 (仅限从站通信)。 值　含义 0　禁止 EOT 抑制 (禁止) 非零　使能 EOT 抑制
ErrorDetection	SINT	GSV	指定错误检测方案。 值　含义 0　BCC(默认值) 1　CRC
MasterMessageTransmit	SINT	GSV	主站信息传输的当前值 (仅限主站通信)。 值　含义 0　在站轮询之间 (默认值) 1　按轮询顺序 (替代主站的站编号)
MaxStation Address	SINT	GSV	DH-485 网络上最大节点地址的当前值 (0...31)。默认值为 31。
NAKReceiveLimit	SINT	GSV	停止传输前作为对信息的响应而接收到的 NAK 数 (仅限点对点通信)。有效值为 0...127。默认值为 3。
NormalPollGroupSize	INT	GSV	在轮询优先级轮询节点数组中的所有站点后，要在正常轮询节点数组中轮询的站点数 (仅限主站通信)。 有效值为 0...255。默认值为 0。
PollingMode	SINT	GSV	当前轮询模式 (仅限主站通信)。默认值为 1。 值　含义 0　基于信息，但不允许从站发起信息 1　基于信息，但允许从站发起信息 (默认值) 2　标准，每次节点扫描传送一条信息 3　标准，每次节点扫描传送多条信息
ReplyMessage Wait	DINT	GSV	接收到 ACK 之后，轮询从站以进行响应之前的等待时间 (充当主站)(仅限主站通信)。有效值为 0...65,535。延迟以 20 毫秒为周期计数。默认值为 5 个周期 (100 毫秒)。
SlavePollTimeout	DINT	GSV	在从站声明由于主站未激活而无法传送之前，从站等待主站轮询的时间 (以毫秒为单位)(仅限从站通信)。有效值为 0...32,767。延迟以 20 毫秒为周期计数。默认值为 3000 个周期 (1 分钟)。
StationAddress	INT	GSV	串口的当前站地址。有效值为 0...254。默认值为 0。
TokenHoldFactor	SINT	GSV	在 DH-485 网络上传送令牌之前此节点所发送信息的最大数目的当前值 (1...4)。默认值为 1。
TransmitRetries	SINT	GSV	在不获取应答的情况下重新发送信息的次数 (仅限主站和从站通信)。 有效值为 0...127。默认值为 3。
PendingACK Timeout	DINT	SSV	ACKTimeout 属性的待定值。
Pending Duplicate Detection	SINT	SSV	DuplicateDetection 属性的待定值。

属性	数据类型	指令	说明
Pending Embedded ResponseEnable	SINT	SSV	EmbeddedResponse 属性的待定值。
PendingEnable StoreFwd	SINT	SSV	EnableStoreFwd 属性的待定值。
PendingENQ TransmitLimit	SINT	SSV	ENQTransmitLimit 属性的待定值。
PendingEOT Suppression	SINT	SSV	EOTSuppression 属性的待定值。
PendingError Detection	SINT	SSV	ErrorDetection 属性的待定值。
PendingMaster Message Transmit	SINT	SSV	MasterMessageTransmit 属性的待定值。
PendingMax StationAddress	SINT	SSV	MaxStationAddress 属性的待定值。
PendingNAK ReceiveLimit	SINT	SSV	NAKReceiveLimit 属性的待定值。
PendingNormal PollGroupSize	INT	SSV	NormalPollGroupSize 属性的待定值。
PendingPolling Mode	SINT	SSV	PollingMode 属性的待定值。
PendingReply MessageWait	DINT	SSV	ReplyMessageWait 属性的待定值。
PendingSlavePoll Timeout	DINT	SSV	SlavePollTimeout 属性的待定值。
PendingStation Address	INT	SSV	StationAddress 属性的待定值。
PendingToken HoldFactory	SINT	SSV	TokenHoldFactor 属性的待定值。
PendingTransmit Retries	SINT	SSV	TransmitRetries 属性的待定值。

FaultLog 属性

FaultLog 对象提供有关控制器的故障信息。

属性	数据类型	指令	说明
MajorEvents	INT	GSV SSV	自上次复位此计数器以来发生的主要故障的数目。
MajorFaultBits	DINT	GSV SSV	各个位分别表示当前主要故障的不同原因。 位　含义 1　掉电 3　I/O 4　指令执行 (程序) 5　故障处理程序 6　看门狗 7　堆栈 8　模式更改 11　运动控制
MinorEvents	INT	GSV SSV	自上次复位此计数器以来发生的次要故障的数目。
MinorFaultBits	DINT	GSV SSV	各个位分别表示当前次要故障的不同原因。 位　含义 4　指令执行 (程序) 6　看门狗 9　串口 10　电池

Message 属性

可以通过 GSV/SSV 指令访问 Message 对象。指定信息标签名称可确定所需的 Message 对象。 Message 对象为设置和触发对等通信提供接口。此对象替代了 PLC-5 处理器的
MG 数据类型。

属性	数据类型	指令	说明
ConnectionPath	SINT[130]	GSV SSV	用于设置连接路径的数据。前两个字节 (低字节和高字节) 是连接路径的长度 (以字节为单位)。
ConnectionRate	DINT	GSV SSV	连接的请求数据包速率。
MessageType	SINT	GSV SSV	指定信息类型。 值 含义 0 未初始化
Port	SINT	GSV SSV	指示发送信息时应使用的端口。 值 含义 1 背板 2 串口
Timeout Multiplier	SINT	GSV SSV	确定何时应将连接视为超时而将其关闭。 值 含义 0 连接将在达到更新速率的 4 倍后超时 (默认值) 1 连接将在达到更新速率的 8 倍后超时 2 连接将在达到更新速率的 16 倍后超时
Unconnected Timeout	DINT	GSV SSV	所有未连接信息的超时 (以微秒为单位)。默认值为 30,000,000 微秒 (30 秒)。

Module 属性

Module 对象提供有关模块的状态信息。要选择特定 Module 对象，请将 GSV/SSV 指令的 “ 对象名称 ” (Object Name) 操作数设置为模块名称。指定的模块必须在控制器项目管理器的 “ I/O 组态 ” (I/O Configuration) 部分存在，且必须具有设备名称。

属性	数据类型	指令	说明
EntryStatus	INT	GSV	指明所规定映射条目的当前状态。执行比较操作时应当屏蔽低 12 位。仅第 12...15 位有效。 值　含义 16#0000　待机：控制器正在上电。 16#1000　发生故障：Module 对象与关联模块的连接发生故障。该值不应当用来确定模块是否发生故障，因为 Module 对象在尝试重新连接到模块时会周期性地保持此状态。而应当对 “ 正在运行 ” 状态 (16#4000) 进行测试。检查 FaultCode 是否等于 0 可确定模块是否发生故障。发生故障时，FaultCode 和 FaultInfo 属性将一直有效，直至故障状况得到纠正为止。 16#2000　正在验证：建立与模块的连接之前，Module 对象正在验证 Module 对象的完整性。 16#3000　正在连接：Module 对象正在发起与模块的连接。 16#4000　正在运行：已建立与模块的所有连接，数据正在传送。 16#5000　正在关闭：Module 对象正在关闭与模块的所有连接。 16#6000　已禁止：已禁止 Module 对象 (Mode 属性中的禁止位已置位)。 16#7000　正在等待：此 Module 对象依赖的拥有者未运行。
FaultCode	INT	GSV	发生故障时，识别模块故障的数字。
FaultInfo	DINT	GSV	提供有关 Module 对象故障代码的特定信息。
Firmware SupervisorStatus	INT	GSV	识别固件监控功能的当前工作状态。 值　含义 0　当前未执行模块更新 1　正在执行模块更新
ForceStatus	INT	GSV	指定强制的状态。 位　含义 0　已安装强制 (1 = 是， 0 = 否) 1　已使能强制 (1 = 是， 0 = 否)
Instance	DINT	GSV	提供此模块对象的实例编号。

属性	数据类型	指令	说明
LEDStatus	INT	GSV	指明控制器前面的 I/O 状态指示灯的当前状态。 值 含义 0 状态指示灯熄灭：没有为控制器组态 Module 对象。 (控制器项目管理器的“ I/O 组态 ”(I/O Configuration) 部分没有模块。) 1 红色闪烁：Module 对象均未运行。 2 绿色闪烁：至少有一个 Module 对象未运行。 3 绿色常亮：所有 Module 对象都在运行。 使用此属性时不输入对象名称，因为此属性应用于整个模块集合。
Mode	INT	GSV SSV	指定 Module 对象的当前模式。 位 含义 0 置位后，如果在控制器处于运行模式时 Module 对象连接发生故障，将导致生成主要故障。 2 如果置位，则在关闭与模块的所有连接后，Module 对象将进入“ 已禁止 ”状态。

Program 属性

Program 对象提供有关程序的状态信息。指定程序名称以确定所需的 Program 对象。

属性	数据类型	标准任务内的指令	安全任务内的指令	说明
DisableFlag	SINT	GSV SSV	无	控制此程序的执行。 值　含义 0　使能执行 1　禁止执行
Instance	DINT	GSV	GSV	提供此程序对象的实例编号。
LastScanTime	DINT	GSV SSV	无	上次执行此程序所花费的时间。时间以微秒为单位。
MajorFault Record	DINT[11]	GSV SSV	GSV SSV	记录此程序的主要故障 建议您创建用户自定义的结构来简化对 MajorFaultRecord 属性的访问：

名称	数据类型	格式	说明
TimeLow	DINT	十进制	故障时间戳值的低 32 位
TimeHigh	DINT	十进制	故障时间戳值的高 32 位
Type	INT	十进制	故障类型 (程序、 I/O 等)
Code	INT	十进制	故障的惟一代码 (取决于故障类型)
Info	DINT[8]	十六进制	故障的特定信息 (取决于故障类型和代码)

属性	数据类型	标准任务内的指令	安全任务内的指令	说明
MinorFault Record	DINT[11]	GSV SSV	GSV SSV	记录此程序的次要故障 建议您创建用户自定义的结构来简化对 MinorFaultRecord 属性的访问：
名称数据类型格式说明				
TimeLow	DINT	十进制	故障时间戳值的低 32 位	
TimeHigh	DINT	十进制	故障时间戳值的高 32 位	
Type	INT	十进制	故障类型 (程序、 I/O 等)	
Code	INT	十进制	故障的惟一代码 (取决于故障类型)	
Info	DINT[8]	十六进制	故障的特定信息 (取决于故障类型和代码)	
MaxScanTime	DINT	GSV SSV	无	此程序的记录的最长执行时间。时间以微秒为单位。
Name	字符串	GSV	GSV	程序的名称。

Routine 属性

Routine 对象提供有关例程的状态信息。指定例程名称可确定所需的 Routine 对象。

属性	数据类型	标准任务内的指令	安全任务内的指令	说明
Instance	DINT	GSV	GSV	提供此例程对象的实例编号。有效值为 0…65,535。
Name	字符串	GSV	GSV	例程的名称。
SFCPaused	INT	GSV	无	在 SFC 例程中，指示 SFC 是否暂停。 值　含义 0　SFC 未暂停 1　SFC 已暂停
SFCResuming	INT	GSV SSV	无	在 SFC 例程中，指示 SFC 是否正在恢复执行。 值　含义 0　SFC 未执行。在其中执行图表的扫描结束时，此属性会自动设置为 0 1　SFC 正在执行。步进计时器和 Action 计时器将保持其之前的值 (倘若如此组态)。流程图中止暂停后首次扫描时，此属性会自动设置为 1。

Safety 属性

安全控制器对象提供有关安全状态和安全签名的信息。SafetyTask 和 SafetyFaultRecord 属性可以采集有关不可恢复故障的信息。

请参见 《GuardLogix 控制器用户手册》 (出版号 1756-UM020)。

属性	数据类型	标准任务内的指令	安全任务内的指令	说明
SafetyLocked	SINT	GSV	无	指示控制器已安全锁定还是已安全解锁。
SafetyStatus	INT	GSV	无	安全状态可指定为： 值　含义 1000000000000000　安全任务正常。 1000000000000001　安全任务无法运行。 0000000000000000　伙伴丢失。 0000000000000001　伙伴不可用。 0000000000000010　硬件不兼容。 0000000000000011　固件不兼容。
SafetySignature Exists	SINT	GSV	GSV	指示是否存在安全任务签名。
SafetySignature ID	DINT	GSV	无	32 位标识号。
SafetySignature	字符串	GSV	无	32 位标识号。
SafetyTaskFault Record	DINT[11]	GSV	无	记录安全任务故障。

SerialPort 属性

SerialPort 对象提供串行通信端口的接口。

属性	数据类型	指令	说明
BaudRate	DINT	GSV	指定波特率。有效值为 110、300、600、1200、2400、4800、9600 和 19200 (默认值)。
ComDriverID	SINT	GSV	指定特定驱动程序。 值　含义 0xA2　DF1(默认值) 0xA3　ASCII
DataBits	SINT	GSV	指定每个字符的数据位数。 值　含义 7　7 个数据位 (仅限 ASCII) 8　8 个数据位 (默认值)
DCDDelay	INT	GSV	指定数据载波检测 (DCD) 在数据包出错前变为低的等待时间。延迟以 1 秒的数据包数计数。默认值为 0 个计数器。
Parity	SINT	GSV	指定奇偶校验。 值　含义 0　无奇偶校验 (默认值) 1　奇校验 (仅限 ASCII) 2　偶校验
RTSOffDelay	INT	GSV	传送完最后一个字符后关闭 RTS 行的延迟时间。有效值为 0...32,767。延迟以 20 毫秒为周期计数。默认值为 0 毫秒。
RTSSendDelay	INT	GSV	打开 RTS 行后传送信息的第一个字符的延迟时间。有效值为 0...32,767。延迟以 20 毫秒为周期计数。默认值为 0 毫秒。
StopBits	SINT	GSV	指定停止位的数目。 值　含义 1　1 个停止位 (默认值) 2　2 个停止位 (仅限 ASCII)
PendingBaudRate	DINT	SSV	BaudRate 属性的待定值。
PendingCOM DriverID	SINT	SSV	COMDriverID 属性的待定值。
PendingDataBits	SINT	SSV	DataBits 属性的待定值。
PendingDCD Delay	INT	SSV	DCDDelay 属性的待定值。
PendingParity	SINT	SSV	Parity 属性的待定值。
PendingRTSOff Delay	INT	SSV	RTSOffDelay 属性的待定值。
PendingRTSSend Delay	INT	SSV	RTSSendDelay 属性的待定值。
PendingStopBits	SINT	SSV	StopBits 属性的待定值。

Task 属性

Task 对象提供有关任务的状态信息。指定任务名称可确定所需的 Task 对象。

属性	数据类型	标准任务内的指令	安全任务内的指令	说明
DisableUpdate Outputs	DINT	GSV SSV	无	任务结束时使能或禁止对输出的处理。 值　含义 0　任务结束时使能对输出的处理 非零　任务结束时禁止对输出的处理
EnableTimeOut	DINT	GSV SSV	无	使能或禁止事件任务的超时功能。 值　含义 0　禁止超时功能 非零　使能超时功能
InhibitTask	DINT	GSV SSV	无	阻止任务执行。如果任务被禁止，控制器在从编程模式转换为运行模式或测试模式时，仍会预扫描该任务。 值　含义 0　使能任务 0(默认值) 非零　禁止 (禁止) 任务
Instance	DINT	GSV	GSV	提供此任务对象的实例编号。有效值为 0...31。
LastScanTime	DINT	GSV SSV	无	上次执行此任务所花费的时间。时间以微秒为单位。
MaximumInterval	DINT[2]	GSV SSV	无	任务的两次连续执行的最大时间间隔。 DINT[0] 包含该值的低 32 位； DINT[1] 包含该值的高 32 位。值为 0 表示该任务执行了 1 次或更少次。
MaximumScan Time	DINT	GSV SSV	无	此程序的记录的最长执行时间。时间以微秒为单位。
MinimumInterval	DINT[2]	GSV SSV	无	任务的两次连续执行的最小时间间隔。 DINT[0] 包含该值的低 32 位； DINT[1] 包含该值的高 32 位。值为 0 表示该任务执行了 1 次或更少次。
Name	字符串	GSV	GSV	任务的名称。
OverlapCount	DINT	GSV SSV	GSV SSV	任务仍在执行时触发该任务的次数。对事件或周期性任务有效。要清除计数，请将该属性设置为 0。
Priority	INT	GSV SSV	GSV	此任务与其它任务相比的相对优先权。有效值为 0...15。
Rate	DINT	GSV SSV	GSV	任务的两次执行时间间隔。时间以微秒为单位。

属性	数据类型	标准任务内的指令	安全任务内的指令	说明
StartTime	DINT[2]	GSV SSV	无	上次开始执行该任务时的 WALLCLOCKTIME 值。DINT[0] 包含该值的低 32 位；DINT[1] 包含该值的高 32 位。
Status	DINT	GSV SSV	无	有关任务的状态信息。一旦控制器对以下某个位置位，必须手动清除该位。 位　含义 0　EVENT 指令触发了任务 (仅限事件任务) 1　超时触发了任务 (仅限事件任务) 2　此任务发生了交迭
Watchdog	DINT	GSV SSV	GSV	执行与此任务相关的所有程序时的时间限制。时间以微秒为单位。 如果输入 0，将分配以下值： 时间　任务类型 0.5 秒　周期性 5.0 秒　连续性

WallClockTime 属性

WallClockTime 对象提供时间戳，控制器可使用该时间戳进行规划。

属性	数据类型	指令	说明
ApplyDST	SINT	GSV SSV	指示是否使能夏令时。 值含义 0 不按照夏令时进行调整 非零 按照夏令时进行调整
CSTOffset	DINT[2]	GSV SSV	相对于 CST 对象的 CurrentValue 的正向偏移 (有关协调系统时间，请参见第 190 页)。 DINT[0] 包含该值的低 32 位；DINT[1] 包含该值的高 32 位。值以微秒为单位。默认值为 0。
CurrentValue	DINT[2]	GSV SSV	时钟时间的当前值。 DINT[0] 包含该值的低 32 位；DINT[1] 包含该值的高 32 位。该值是自 1972 年 1 月 1 日 0000 时以来经历的微秒数。 CST 和 WALLCLOCKTIME 对象在控制器中是数学上相关的。例如，如果添加 CST CurrentValue 和 WALLCLOCKTIME CTSOffset，则结果就是 WALLCLOCKTIME CurrentValue。
DateTime	DINT[7]	GSV SSV	日期和时间。 值含义 DINT[0] 年 DINT[1] 月 (1...12) DINT[2] 日 (1...31) DINT[3] 时 (0...23) DINT[4] 分 (0...59) DINT[5] 秒 (0...59) DINT[6] 微秒 (0...999,999)
DSTAdjustment	INT	GSV SSV	按照夏令时调整的分钟数。
LocalDateTime	DINT[7]	GSV SSV	当前调整的本地时间。 值含义 DINT[0] 年 DINT[1] 月 (1...12) DINT[2] 日 (1...31) DINT[3] 时 (0...23) DINT[4] 分 (0...59) DINT[5] 秒 (0...59) DINT[6] 微秒 (0...999,999)
TimeZoneString	INT	GSV SSV	时间值的时区。

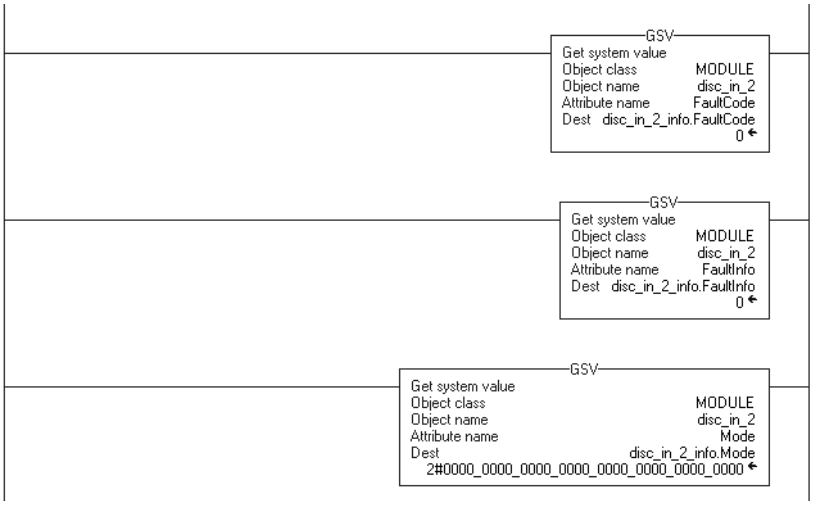
GSV/SSV 编程示例

获取故障信息

下面的示例使用 GSV 指令获取故障信息。

示例 **1**： 此示例从 I/O 模块 *disc_in_2* 中获取故障信息，并将数据放入用户定义的 *disc_in_2_info* 结构中。

梯形图

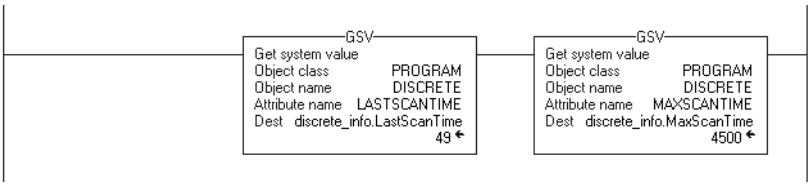


结构化文本

```
GSV(MODULE,disc_in_2,FaultCode,disc_in_2_info.FaultCode);

GSV(MODULE,disc_in_2,FaultInfo,disc_in_2_info.FaultInfo);

GSV(MODULE,disc_in_2,Mode,disc_in_2info.Mode);
```

示例 **2**： 此示例获取有关 *discrete* 程序的状态信息，并将数据放入用户定义的 *discrete_info* 结构中。

梯形图

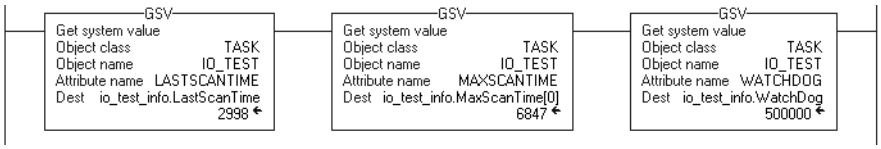


结构化文本

```
GSV(PROGRAM,DISCRETE,LASTSCANTIME,
     discrete_info.LastScanTime);

GSV(PROGRAM,DISCRETE,MAXSCANTIME,discrete_info.MaxScanT
ime);
```

示例 **3**： 此示例获取有关 IO_test 任务的状态信息，并将数据放入用户定义的 io_test_info 结构中。

梯形图



结构化文本

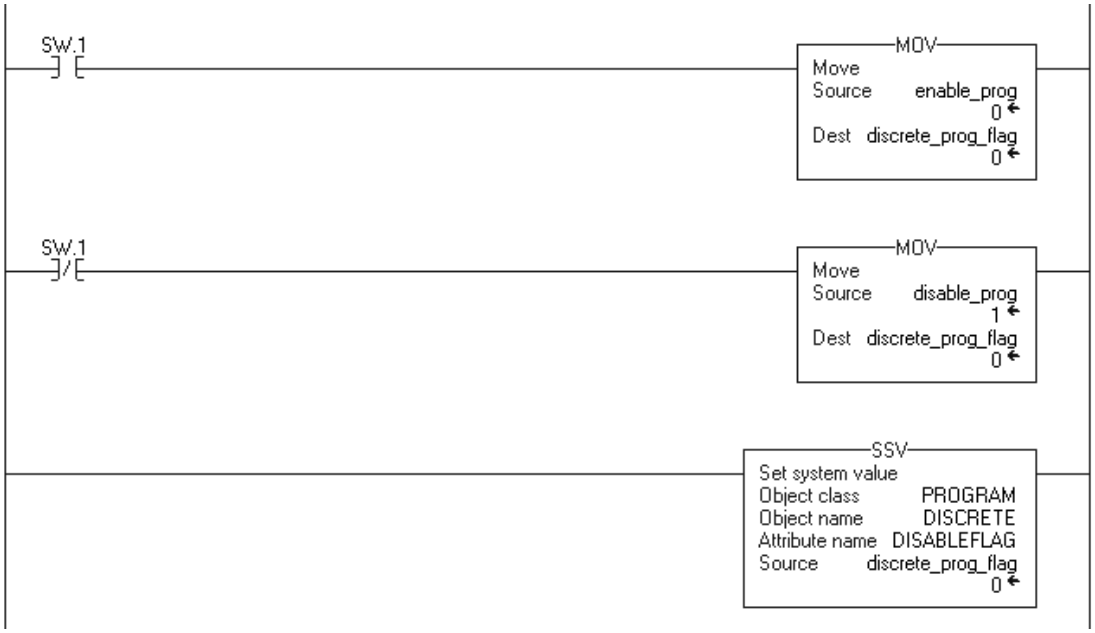
```
GSV(TASK,IO_TEST,LASTSCANTIME,io_test_info.LastScanTime);

GSV(TASK,IO_TEST,MAXSCANTIME,io_test_info.MaxScanTime);

GSV(TASK,IO_TEST,WATCHDOG,io_test_info.WatchDog);
```

设置使能和禁止标志

下面的示例使用 SSV 指令来使能或禁止程序。也可以使用此方法来使能或禁止 I/O 模块，这与 PLC-5 处理器使用禁止位类似。

示例： 根据 *SW.1* 的状态，将相应值放入 *discrete* 程序的 *disableflag* 属性中。

梯形图



结构化文本

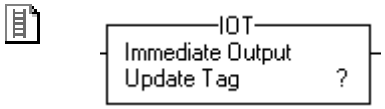
```
IF SW.1 THEN
    discrete_prog_flag := enable_prog;
ELSE
    discrete_prog_flag := disable_prog;
END_IF;
SSV(PROGRAM,DISCRETE,DISABLEFLAG,discrete_prog_flag);
```

立即输出 (IOT)

IOT 指令立即刷新指定的输出数据 (输出标签或生产者标签)。

操作数：

梯形图



操作数	类型	格式	说明
更新标签 (Update Tag)		标签	要刷新的标签，可以是下面一个： • I/O 模块的输出标签 • 生产者标签
			不要选择标签的成员或元素。例如， Local:5:0 正确，但 Local:5:0.Data 不正确。

结构化文本

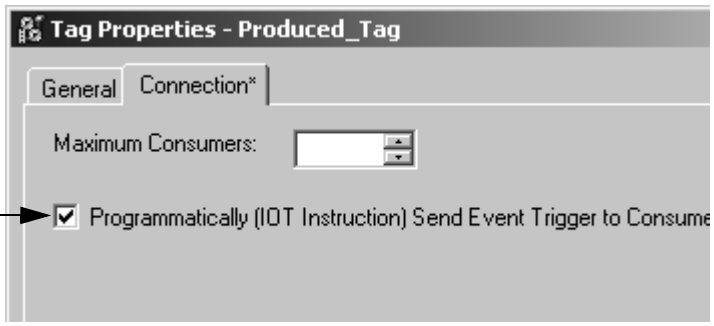
```
IOT(output_tag);
```

操作数与梯形图 IOT 指令的操作数相同。

说明： IOT 指令会覆盖输出连接的请求数据包间隔 (RPI)，并通过该连接发送刷新的数据。

- 输出连接是与 I/O 模块的输出标签或与生产者标签关联的连接。
- 如果该连接用于生产者标签，则 IOT 指令还会将事件触发器发送到消费者标签控制器。使 IOT 指令在消费者标签控制器中触发事件任务。

要使用 IOT 指令和生产者标签在消费者标签控制器中触发事件任务，请按以下方式组态生产者标签。



勾选此框。

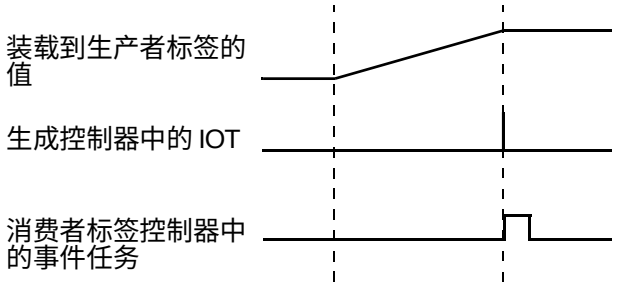
这可将标签组态为仅通过 IOT 指令刷新其事件触发器。

控制器之间的网络类型决定了消费者标签控制器何时通过 IOT 指令接收新数据和事件触发器。

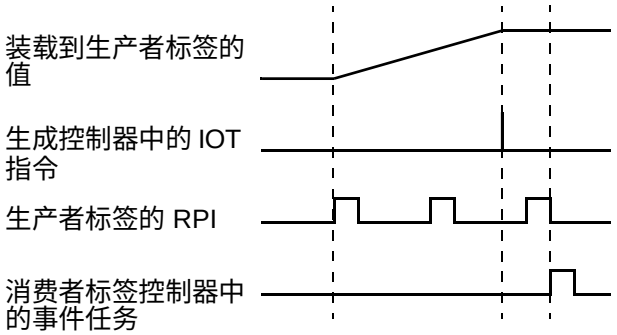
使用下列控制器	通过以下网络	使用设备接收数据和事件触发器的时间
ControlLogix	背板	立即
	EtherNet/IP 网络	立即
	ControlNet 网络	在消费者标签 (连接) 的实际数据包间隔 (API) 内
SoftLogix5800	只能通过 ControlNet 网络生产和使用标签。	在消费者标签 (连接) 的实际数据包间隔 (API) 内

下图对使用 IOT 指令通过 EtherNet/IP 和 ControlNet 网络接收数据进行了比较。

EtherNet/IP 网络



ControlNet 网络



算术状态标志： 不受影响

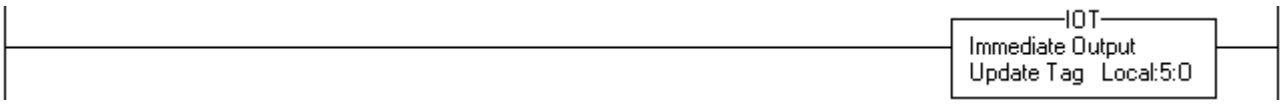
故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	指令： • 刷新指定标签的连接。 • 复位连接的 RPI 计时器。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例 **1**： 执行 IOT 指令，该指令立即将 Local:5:0 标签的值发送到输出模块。

梯形图

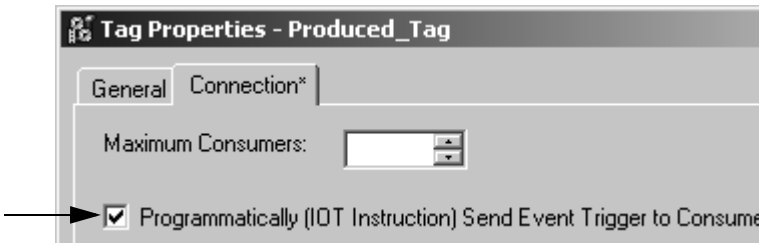


结构化文本

```
IOT (Local:5:O);
```

示例 **2**： 此控制器控制站点 24，并为下一个站 (站点 25) 生成数据。要使用 IOT 指令来指示新数据的传输，请按如下方式组态生产者标签：

组态 Produced_Tag 以通过 IOT 指令刷新其事件触发器。

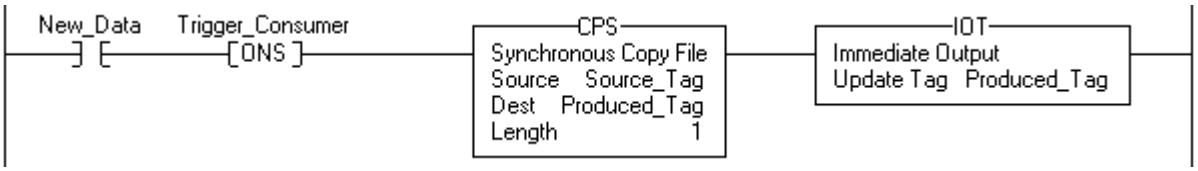


梯形图

如果 *New_Data* 为使能状态，则对于一次扫描，会发生以下情况：

CPS 指令将 *Produced_Tag* 设置为 *Source_Tag*。

IOT 指令刷新 *Produced_Tag*，并将此刷新发送到消费者标签控制器 (站点 25)。消费者标签控制器收到此刷新时，它会在该控制器中触发关联的事件任务。



结构化文本

```
IF New_Data AND NOT Trigger_Consumer THEN
    CPS (Source_Tag,Produced_Tag,1);
    IOT (Produced_Tag);
END_IF;
    Trigger_Consumer := New_Data;
```

比较指令
(CMP、 EQU、 GEQ、 GRT、 LEQ、 LES、 LIM、 MEQ、 NEQ)

简介

通过比较指令，可以利用一个表达式或一个特定的比较指令对值进行比较。

如果要	使用以下指令	在以下语言中可用	页码
基于表达式对值进行比较	CMP	梯形图 结构化文本(1)	214
检验两个值是否相等	EQU	梯形图 结构化文本(2) 功能块	219
检验一个值是否大于或等于另一个值	GEQ	梯形图 结构化文本 (1) 功能块	223
检验一个值是否大于另一个值	GRT	梯形图 结构化文本 (1) 功能块	227
检验一个值是否小于或等于另一个值	LEQ	梯形图 结构化文本 (1) 功能块	231
检验一个值是否小于另一个值	LES	梯形图 结构化文本 (1) 功能块	235
检验一个值是否介于其它两个值之间	LIM	梯形图 结构化文本 (1) 功能块	239
通过屏蔽码传递两个值，然后检验结果是否相等	MEQ	梯形图 结构化文本 (1) 功能块	245
检验一个值是否不等于另一个值	NEQ	梯形图 结构化文本 (1) 功能块	250

(1) 没有对等的结构化文本指令。使用其它结构化文本编程可获得相同结果。请参见指令说明。

(2) 没有对等的结构化文本指令。在表达式中使用操作符。

可以比较不同数据类型的值，如浮点型和整型。

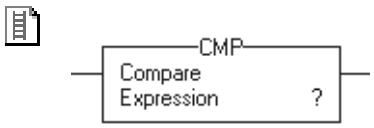
对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

比较 (CMP)

CMP 指令比较在表达式中指定的算术运算进行比较。

操作数：

梯形图



操作数	类型	格式	说明
表达式 (Expression)	SINT INT **DINT** **REAL** 字符串	立即数　标签	由标签和／或立即值 (被运算符隔开) 组成的表达式。
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		

结构化文本

结构化文本没有 CMP 指令，但可通过 IF...THEN 结构和表达式获得相同结果。

```
IF BOOL_expression THEN
    <statement>;
END_IF;
```

有关结构化文本中结构和表达式的语法的信息，请参见结构化文本语法。

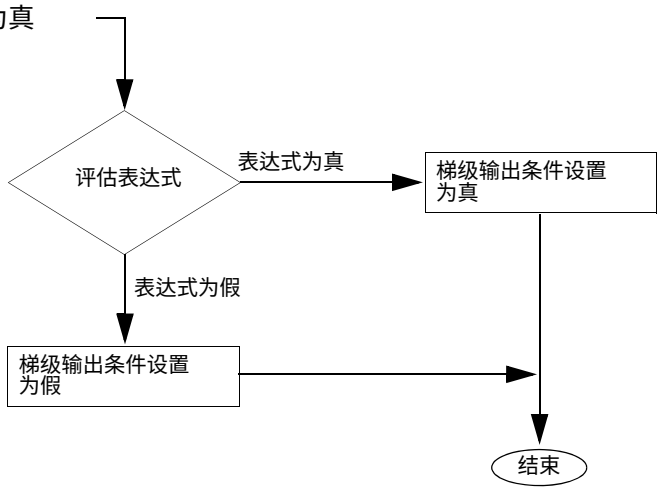
说明：　利用运算符、标签和立即值定义 CMP 表达式。使用圆括号 ( ) 定义更复杂表达式的各部分。

与其它比较指令相比，CMP 指令的执行速度略慢，所用的内存更多。CMP 指令的优点是它允许在一条指令中输入多个复杂的表达式。

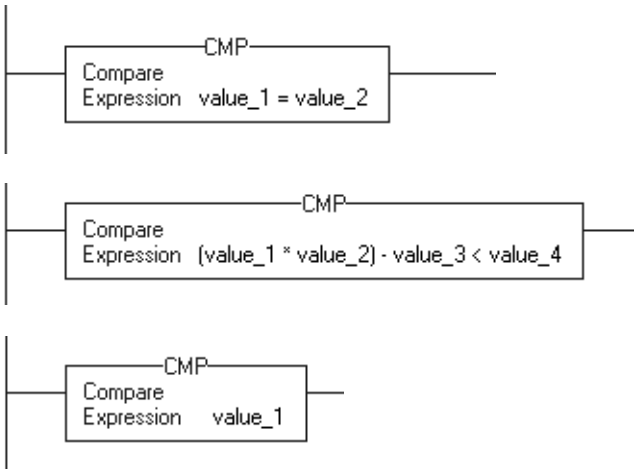
算术状态标志：　仅当表达式中包含可影响算术状态标志的运算符 (例如，+、–、*、/) 时， CMP 才会影响算术状态标志。

错误条件：　无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

示例： 如果 CMP 指令判定表达式为真，则梯级输出条件设置为真。



如果键入一个没有比较运算符的表达式 (如 *value_1 + value_2* 或 *value_1*)，指令将按如表中所述评估表达式。

如果表达式为	梯级输出条件设置为
非零	真
零	假

CMP 表达式

CMP 指令中表达式的编写方法与 FSC 指令中表达式的编写方法相同。请查看以下部分，了解以上两种指令通用的有效运算符、格式和运算顺序。

有效运算符

运算符：	说明	最佳类型
+	加	DINT、 REAL
-	减 / 取反	DINT、 REAL
*	乘	DINT、 REAL
/	除	DINT、 REAL
=	等于	DINT、 REAL
<	小于	DINT、 REAL
<=	小于等于	DINT、 REAL
>	大于	DINT、 REAL
>=	大于等于	DINT、 REAL
<>	不等于	DINT、 REAL
**	指数 (x 的 y 次幂)	DINT、 REAL
ABS	绝对值	DINT、 REAL
ACS	反余弦	REAL
AND	按位与	DINT
ASN	反正弦	REAL
ATN	反正切	REAL
COS	余弦	REAL

运算符：	说明	最佳类型
DEG	弧度转角度	DINT、 REAL
FRD	BCD 转整数	DINT
LN	自然对数	REAL
LOG	以 10 为底的对数	REAL
MOD	求模除法	DINT、 REAL
NOT	按位取反	DINT
OR	按位或	DINT
RAD	角度转弧度	DINT、 REAL
SIN	正弦	REAL
SQR	平方根	DINT、 REAL
TAN	正切	REAL
TOD	整数转 BCD	DINT
TRN	截断	DINT、 REAL
XOR	按位异或	DINT

格式表达式

对于表达式中使用的每一个运算符，都需要提供一个或两个操作数 (标签或立即值)。有关表达式中运算符和操作数的格式，请遵循下表中的规定 .

运算符作用的内容	使用的格式	示例
一个操作数	运算符 (操作数)	ABS(*tag_a*)
两个操作数	操作数 _a 运算符 操作数 _b	• *tag_b + 5* • *tag_c* AND *tag_d* • *(tag_e ** 2)* MOD *(tag_f / tag_g)*

确定运算顺序

写入表达式中的运算将由指令按照规定的顺序执行，不一定按照写入的顺序执行。通过用圆括号括起一些内容，可以更改运算顺序，这样指令就会先执行括号内的运算，后执行其它运算。

相同顺序的运算从左到右执行。

顺序	运算
1.	( )
2.	ABS、 ACS、 ASN、 ATN、 COS、 DEG、FRD、LN、LOG、RAD、SIN、 SQR、 TAN、 TOD、 TRN
3.	**
4.	– (取反)， NOT
5.	*、 /、 MOD
6.	<, <=, >, >=, =
7.	– (减)， +
8.	AND
9.	XOR
10.	OR

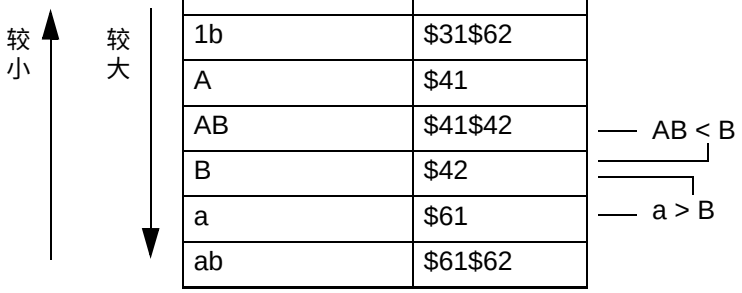
在表达式中使用字符串

使用梯形图或结构化文本表达式来比较字符串数据类型。在表达式中使用字符串时，请遵循以下原则：

- 通过表达式可比较两个字符串标签。
- 不能将 ASCII 字符直接输入到表达式中。
- 只允许使用下列运算符。

运算符	说明
=	等于
<	小于
<=	小于等于
>	大于
>=	大于等于
<>	不等于

- 如果字符串的字符匹配，则字符串相等。
- ASCII 字符区分大小写。大写的“ A ”($41) **不**等于小写的“ a ”($61)。
- 字符的十六进制值决定了一个字符串小于还是大于另一个字符串。有关字符的十六进制代码，请参见本手册的封底。
- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。



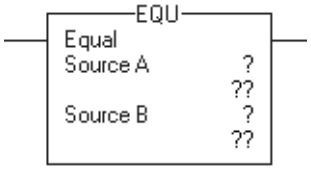
ASCII 字符	十六进制码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

等于 (EQU)

EQU 指令用于检验 Source A 是否等于 Source B。

操作数：

梯形图



操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source B 进行比较的值
源 B (Source B)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source A 进行比较的值

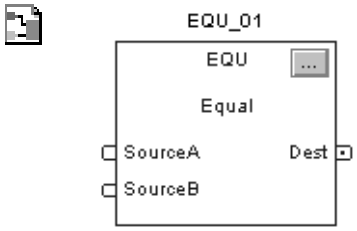
- 如果键入 SINT 或 INT 标签，值将通过符号扩展转换为 DINT 值。
- 很少存在绝对相等的 REAL 值。如果需要确定两个 REAL 值是否相等，请使用 LIM 指令。
- 字符串数据类型包括：
  - 默认的 STRING 数据类型。
  - 所创建的任何新的字符串数据类型。
- 要检验字符串的字符，请输入 Source A 和 Source B 的字符串标签。

结构化文本

```
IF sourceA = sourceB THEN
    <statements>;
```

在表达式中，等号 “ = ” 用作运算符。该表达式评估 *sourceA* 是否等于 *sourceB*。

有关结构化文本中表达式的语法的信息，请参见结构化文本语法。



功能块

操作数	类型	格式	说明
EQU 标签	FBD_COMPARE	结构	EQU 结构

FBD_COMPARE 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	与 SourceB 进行比较的值。 任何浮点数都有效
SourceB	REAL	与 SourceA 进行比较的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 EQU 指令的梯级输出条件。

说明： 使用 EQU 指令比较两个数或两个由 ASCII 字符构成的字符串。比较字符串时：

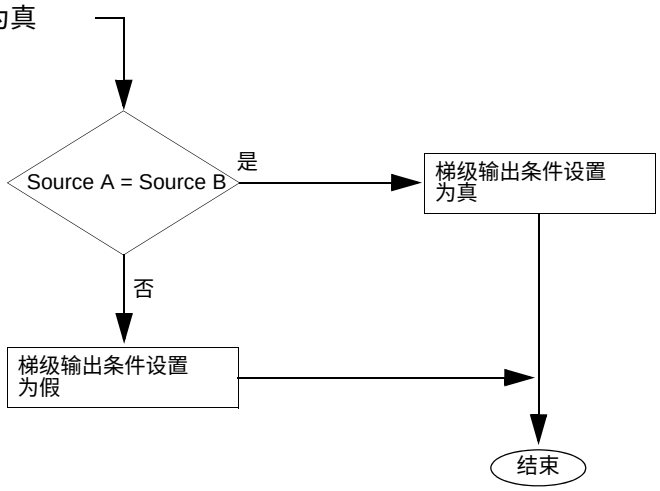
- 如果字符串的字符匹配，则字符串相等。
- ASCII 字符区分大小写。大写的“ A ”($41) 不等于小写的“ a ”($61)。

算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真 	
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

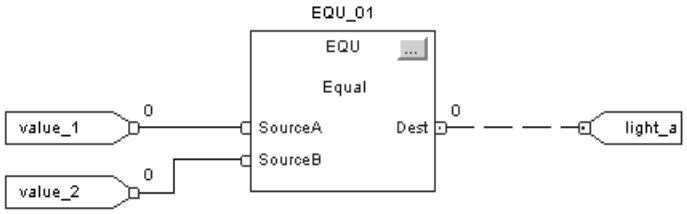
示例： 如果 *value_1* 等于 *value_2*，置位 *light_a*。如果 *value_1* 不等于 *value_2*，清零 *light_a*。

梯形图

结构化文本

```
light_a := (value_1 = value_2);
```

功能块

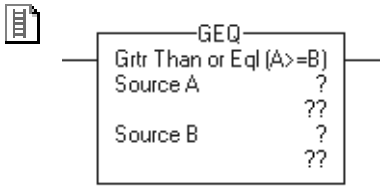


大于等于 (GEQ)

GEQ 指令用于检验 Source A 是否大于等于 Source B。

操作数：

梯形图



操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source B 进行比较的值
源 B (Source B)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source A 进行比较的值

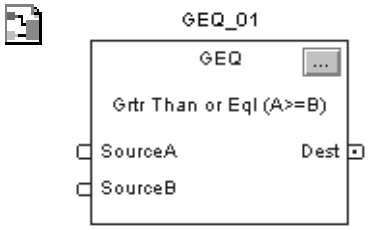
- 如果键入 SINT 或 INT 标签，值将通过符号扩展转换为 DINT 值。
- 字符串数据类型包括：
  - 默认的 STRING 数据类型。
  - 所创建的任何新的字符串数据类型。
- 要检验字符串的字符，请输入 Source A 和 Source B 的字符串标签。

结构化文本

```
IF sourceA >= sourceB THEN
    <statements>;
```

在表达式中，放在一起的大于号和等于号 “ >= ” 用作运算符。该表达式评估 *sourceA* 是否大于等于 *sourceB*。

有关结构化文本中表达式的语法的信息，请参见结构化文本语法。



功能块

操作数	类型	格式	说明
GEQ 标签	FBD_COMPARE	结构	GEQ 结构

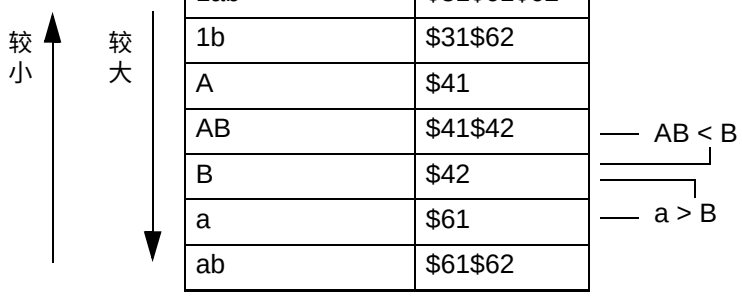
FBD_COMPARE 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	与 SourceB 进行比较的值。 任何浮点数都有效
SourceB	REAL	与 SourceA 进行比较的值。 任何浮点数都有效
输出参数	**数据类型**	**说明**
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 GEQ 指令的梯级输出条件。

说明： GEQ 指令用于检验 Source A 是否大于等于 Source B。

比较字符串时：

- 字符的十六进制值决定了一个字符串小于还是大于另一个字符串。有关字符的十六进制代码，请参见本手册的封底。
- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。



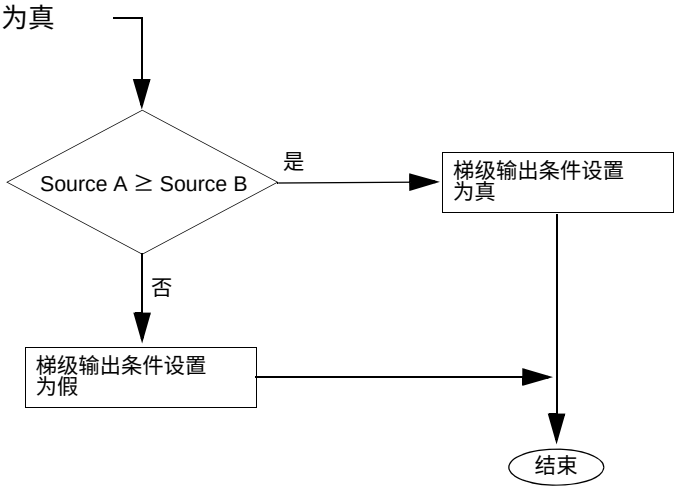
ASCII 字符	十六进制码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

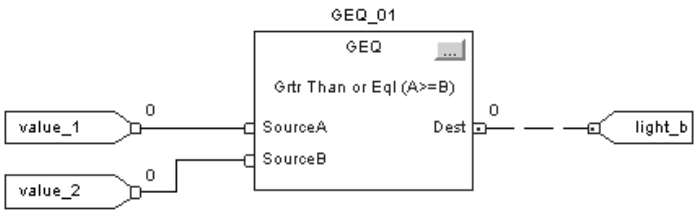
示例： 如果 *value_1* 大于或等于 *value_2*，置位 *light_b*。如果 *value_1* 小于 *value_2*，清零 *light_b*。

梯形图

结构化文本

```
light_b := (value_1 >= value_2);
```

功能块

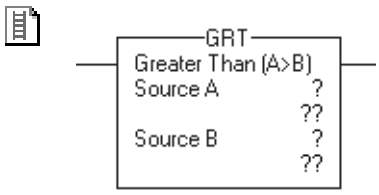


大于 (GRT)

GEQ 指令用于检验 Source A 是否大于 Source B。

操作数：

梯形图



操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL** 字符串	立即数 标签	与 Source B 进行比较的值
源 B (Source B)	SINT INT **DINT** **REAL** 字符串	立即数 标签	与 Source A 进行比较的值

- 如果键入 SINT 或 INT 标签，值将通过符号扩展转换为 DINT 值。
- 字符串数据类型包括：
  - 默认的 STRING 数据类型。
  - 所创建的任何新的字符串数据类型。
- 要检验字符串的字符，请输入 Source A 和 Source B 的字符串标签。

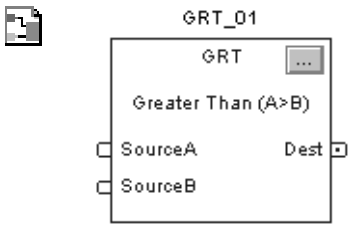
结构化文本

```
IF sourceA > sourceB THEN
    <statements>;
```

在表达式中，大于号 “ > ” 用作运算符。该表达式评估 *sourceA* 是否大于 *sourceB*。

有关结构化文本中表达式的语法的信息，请参见结构化文本语法。

功能块



操作数	类型	格式	说明
GRT 标签	FBD_COMPARE	结构	GRT 结构

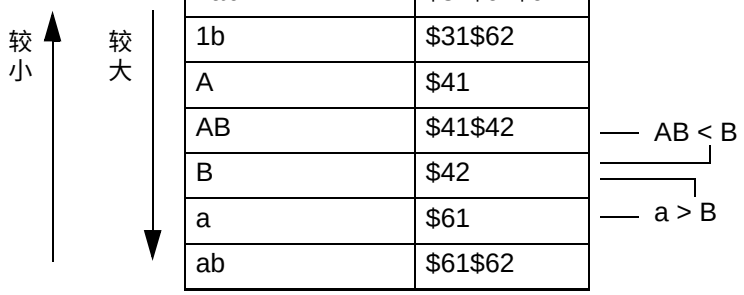
FBD_COMPARE 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	与 SourceB 进行比较的值。 任何浮点数都有效
SourceB	REAL	与 SourceA 进行比较的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 GRT 指令的梯级输出条件。

说明： GEQ 指令用于检验 Source A 是否大于 Source B。

比较字符串时：

- 字符的十六进制值决定了一个字符串小于还是大于另一个字符串。有关字符的十六进制代码，请参见本手册的封底。
- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。



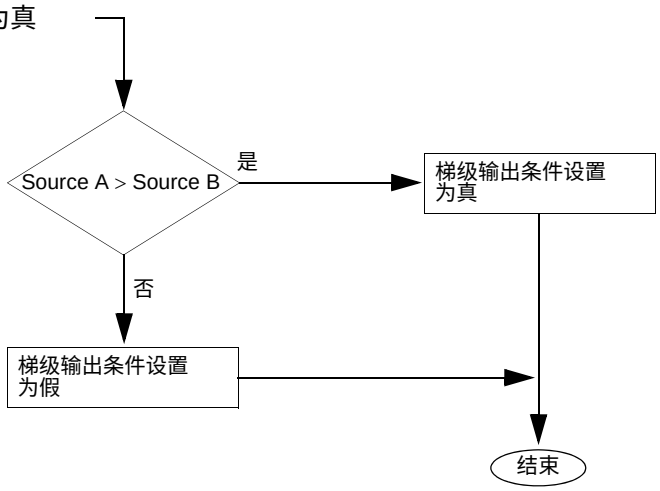
ASCII 字符	十六进制码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

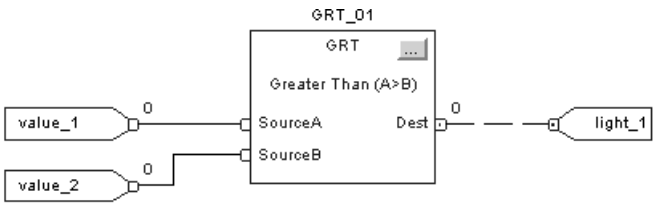
示例： 如果 *value_1* 大于 *value_2*，置位 *light_1*。如果 *value_1* 小于或等于 *value_2*，清零 **l***ight_1*。

梯形图

结构化文本

```
light_1 := (value_1 > value_2);
```

功能块

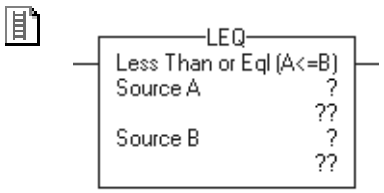


小于等于 (LEQ)

LEQ 指令用于检验 Source A 是否小于或等于 Source B。

操作数：

梯形图



操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source B 进行比较的值
源 B (Source B)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source A 进行比较的值

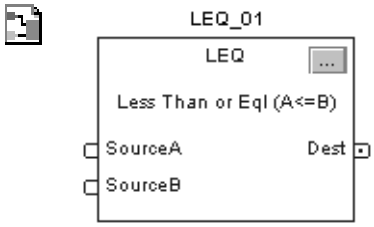
- 如果键入 SINT 或 INT 标签，值将通过符号扩展转换为 DINT 值。
- 字符串数据类型包括：
  - 默认的 STRING 数据类型。
  - 所创建的任何新的字符串数据类型。
- 要检验字符串的字符，请输入 Source A 和 Source B 的字符串标签。

结构化文本

```
IF sourceA <= sourceB THEN
    <statements>;
```

在表达式中，放在一起的小于号和等于号 “ <= ” 用作运算符。该表达式评估 *sourceA* 是否小于
等于 *sourceB*。

有关结构化文本中表达式的语法的信息，请参见结构化文本语法。



功能块

操作数	类型	格式	说明
LEQ 标签	FBD_COMPARE	结构	LEQ 结构

FBD_COMPARE 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	与 SourceB 进行比较的值。 任何浮点数都有效
SourceB	REAL	与 SourceA 进行比较的值。 任何浮点数都有效
输出参数	**数据类型**	**说明**
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 LEQ 指令的梯级输出条件。

说明： LEQ 指令用于检验 Source A 是否小于或等于 Source B。

比较字符串时：

- 字符的十六进制值决定了一个字符串小于还是大于另一个字符串。有关字符的十六进制代码，请参见本手册的封底。
- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。

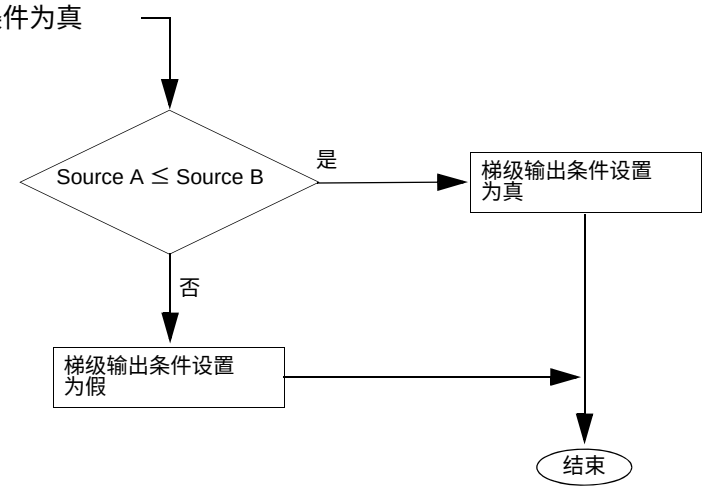
ASCII 字符	十六进制码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

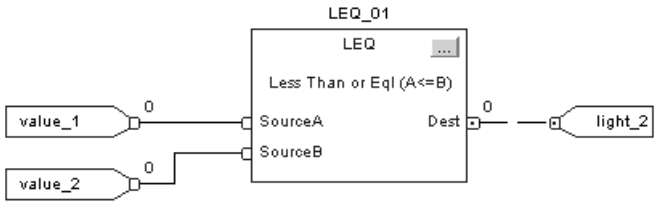
示例： 如果 *value_1* 小于或等于 *value_2*，置位 *light_2*。如果 *value_1* 大于 *value_2*，清零 *light_2*。

梯形图

结构化文本

```
light_2 := (value_1 <= value_2);
```

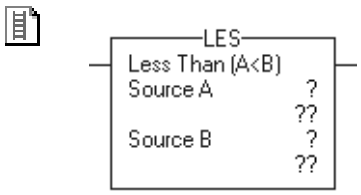
功能块



小于 (LES)

LES 指令用于检验 Source A 是否小于 Source B。

操作数：



梯形图

操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source B 进行比较的值
源 B (Source B)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source A 进行比较的值

- 如果键入 SINT 或 INT 标签，值将通过符号扩展转换为 DINT 值。
- 字符串数据类型包括：
  - 默认的 STRING 数据类型。
  - 所创建的任何新的字符串数据类型。
- 要检验字符串的字符，请输入 Source A 和 Source B 的字符串标签。

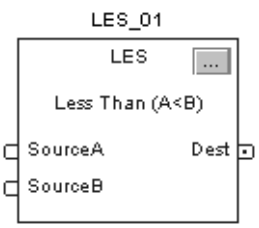
结构化文本

```
IF sourceA < sourceB THEN
    <statements>;
```

在表达式中，小于号 “ < ” 用作运算符。该表达式评估 *sourceA* 是否小于 *sourceB*。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
LES 标签	FBD_COMPARE	结构	LES 结构

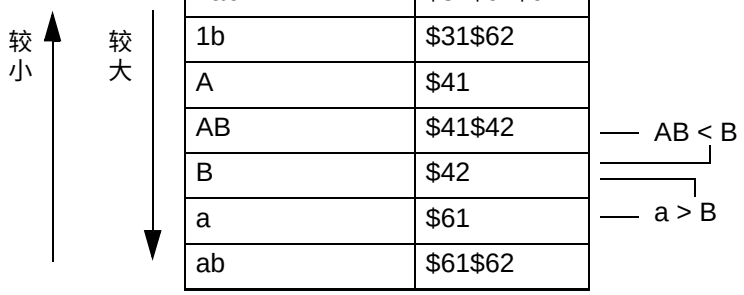
FBD_COMPARE 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	与 SourceB 进行比较的值。 任何浮点数都有效
SourceB	REAL	与 SourceA 进行比较的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 LES 指令的梯级输出条件。

说明： LES 指令用于检验 Source A 是否小于 Source B。

比较字符串时：

- 字符的十六进制值决定了一个字符串小于还是大于另一个字符串。有关字符的十六进制代码，请参见本手册的封底。
- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。



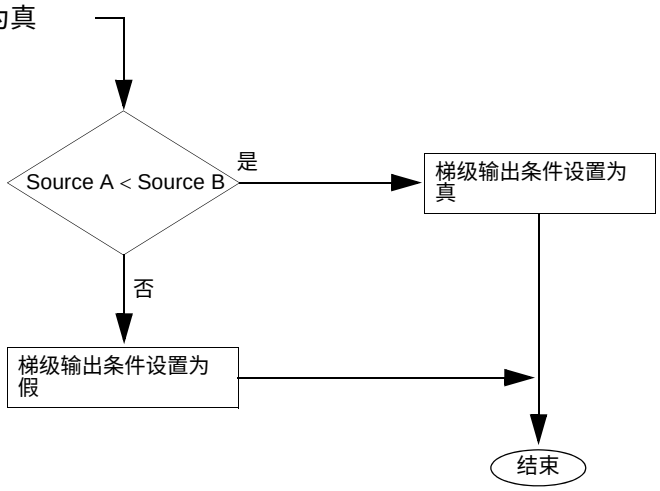
ASCII 字符	十六进制码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

功能块

条件：	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 为假	清零 EnableOut。
EnableIn 为真	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

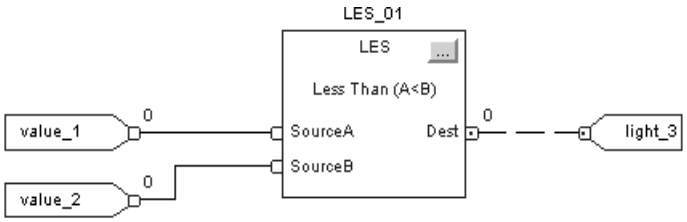
示例： 如果 *value_1* 小于 *value_2*，置位 *light_3*。如果 *value_1* 大于或等于 *value_2*，清零 *light_3*。

梯形图

结构化文本

```
light_3 := (value_1 < value_2);
```

功能块

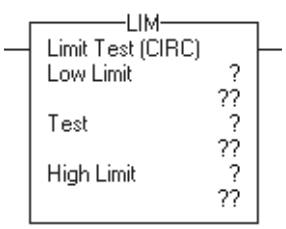


限值 (LIM)

LIM 指令用于检验测试值是否在上下限范围之间。

操作数：

梯形图

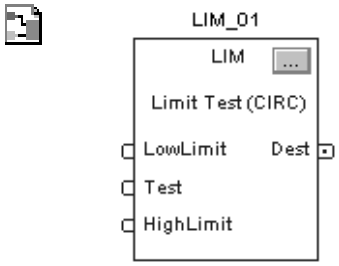


操作数	类型	格式	说明
下限 (Low limit)	SINT INT **DINT** **REAL**	立即数 标签	下限值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
测试 (Test)	SINT INT **DINT** **REAL**	立即数 标签	要检验的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
上限 (High limit)	SINT INT **DINT** **REAL**	立即数 标签	上限值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		

结构化文本

结构化文本没有 LIM 指令，但可通过结构化文本获得相同结果。

```
IF (LowLimit <= HighLimit AND
    (Test >= LowLimit AND Test <= HighLimit)) OR
    (LowLimit >= HighLimit AND
    (Test <= LowLimit OR Test >= HighLimit)) THEN

          <statement>;

END_IF;
```



功能块

操作数	类型	格式	说明
LIM 标签	FBD_LIMIT	结构	LIM 结构

FBD_LIMIT 结构

输入参数	数据类型	说明
EnableIn	BOOL	如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则如 “ 执行 ” 下所述执行该指令。 默认为置位状态。
LowLimit	REAL	下限值。 任何浮点数都有效
Test	REAL	与限值进行比较的值。 任何浮点数都有效
输出参数	**数据类型**	**说明**
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 LIM 指令的梯级输出条件。
HighLimit	REAL	上限值。 任何浮点数都有效

说明： LIM 指令用于检验测试值是否在上下限范围之间。

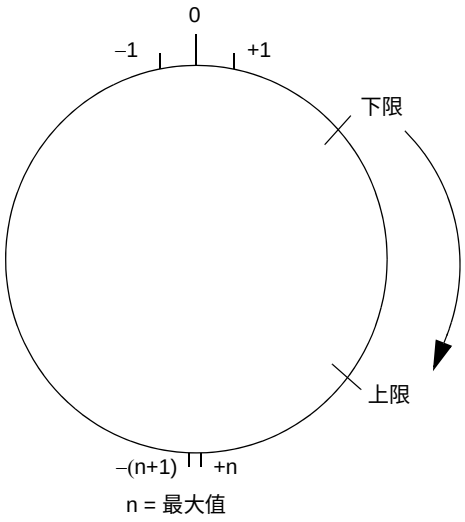
如果下限	并且测试值	则梯级输出条件为
≤上限	等于或介于上下限之间	真
	不等于或位于上下限范围之外	假
≥上限	等于或位于上下限范围之外	真
	不等于或位于上下限范围之内	假

置位最高有效位后，带符号的整数将从最大正数 “ 翻滚 ” 到最大负数。例如，在 16 位整数 (INT 类型) 中，最大正数是 32,767，其十六进制形式为 16#7FFF(位 0 …14 均已置位)。如果将这个数加 1，则结果是 16#8000(位 15 置位)。对于带符号的整数，十六进制的 16#8000 等于十进制的 -32,768。从该值开始加 1，直到全部 16 个位均已置位，此时将得到 16#FFFF，它等于十进制的 -1。

这可以表示为一条圆形数轴 (请参见下面的图)。 LIM 指令开始于下限，顺时针递增，直到达到上限为止。从下限到上限的顺时针范围内的任何测试值都会将梯级输出条件设置为真。从上限到下限的顺时针范围内的任何测试值都会将梯级输出条件设置为假。

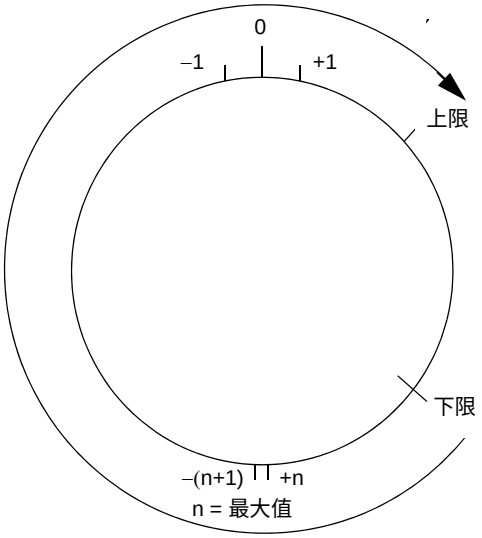
下限≤上限

如果测试值等于或介于上下限之间，则指令为真



下限≥上限

如果测试值等于或位于上下限范围之外，则指令为真

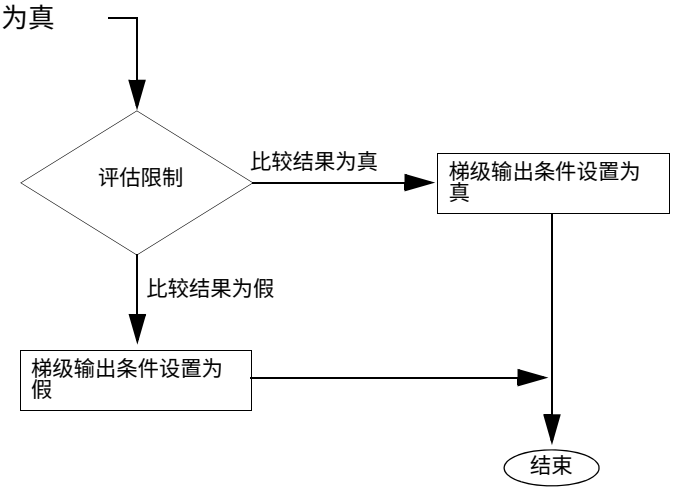


算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真 	
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

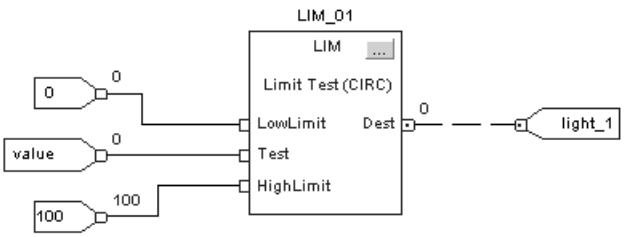
示例 **1**： 下限≤上限：
如果 0 ≤ *value* ≥ 100，置位 *light_1*。如果 *value* < 0 或 *value* >100，清零 *light_1*。

梯形图

结构化文本

```
IF (value <= 100 AND(value >= 0 AND value <= 100)) OR
    (value >= 100 AND value <= 0 OR value >= 100)) THEN

         light_1 := 1;

ELSE

         light_1 := 0;

END_IF;
```

功能块



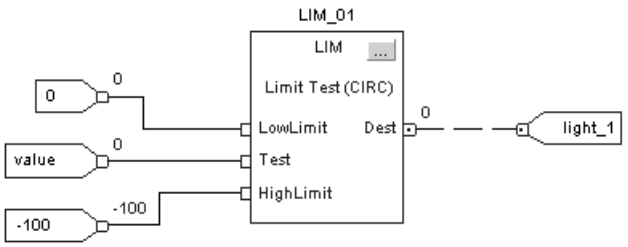
示例 **2**： 下限≥上限：
如果 *value* ≥ 0 或 *value* ≤ –100，置位 *light_1*。如果 *value* < 0 或 *value* >–100，清零 *light_1*。

梯形图

结构化文本

```
IF (0 <= -100 AND value >= 0 AND value <= -100)) OR
    (0 >= -100 AND(value <= 0 OR value >= -100)) THEN

          light_1 := 1;

ELSE

          light_1 := 0;

END_IF;
```

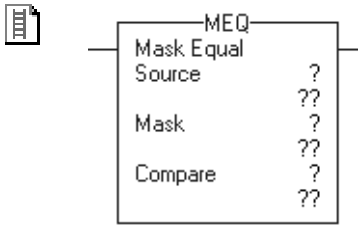
功能块



屏蔽码等于 (MEQ)

MEQ 指令通过屏蔽码传递 Source 和 Compare 值，然后对结果进行比较。

操作数：



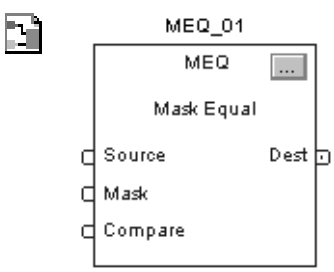
梯形图

操作数	类型	格式	说明
源 (Source)	SINT INT **DINT**	立即数　标签	与 Compare 进行比较的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
屏蔽码 (Mask)	SINT INT **DINT**	立即数　标签	定义要阻止或传递的位
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
比较 (Compare)	SINT INT **DINT**	立即数　标签	与 Source 进行比较的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		

结构化文本

结构化文本没有 MEQ 指令，但可通过结构化文本获得相同结果。

```
IF (Source AND Mask) = (Compare AND Mask) THEN
    <statement>;
END_IF;
```



功能块

操作数	类型	格式	说明
MEQ 标签	FBD_MASK_EQUAL	结构	MEQ 结构

FBD_MASK_EQUAL 结构

输入参数	数据类型	说明
EnableIn	BOOL	如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则如 “ 执行 ” 下所述执行该指令。 默认为置位状态。
Source	DINT	与 Compare 进行比较的值。 任何整数都有效
Mask	DINT	定义要阻止 (屏蔽) 的位。 任何整数都有效
Compare	DINT	比较值。 任何整数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 MEQ 指令的梯级输出条件。

说明： 屏蔽码中的 “ 1 ” 表示该数据位已传递。屏蔽码中的 “ 0 ” 表示该数据位已阻止。通常， Source、 Mask 和 Compare 的值为同一种数据类型。

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

输入立即数屏蔽码值

键入屏蔽码时，编程软件将默认为十进制值。如果想使用另一种格式键入屏蔽码，请在值的前面加上正确的前缀。

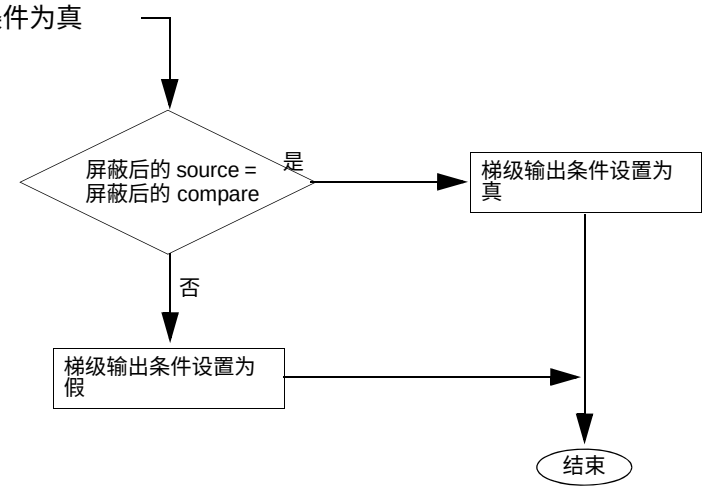
前缀	说明
16#	十六进制 例如， 16#0F0F
8#	八进制 例如， 8#16
2#	二进制 例如， 2#00110011

算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例 **1**： 如果屏蔽后的 *value_1* 等于屏蔽后的 *value_2*，置位 *light_1*。如果屏蔽后的 *value_1* 不等于屏蔽后的 *value_2*，清零 *light_1*。该示例显示屏蔽后的值相等。屏蔽码中的 0 可以阻止指令比较该位 (在示例中用 x 表示)。

value_1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
mask_1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
屏蔽后的 value_1	0	1	0	1	0	1	0	1	1	1	1	1	x	x	x	x

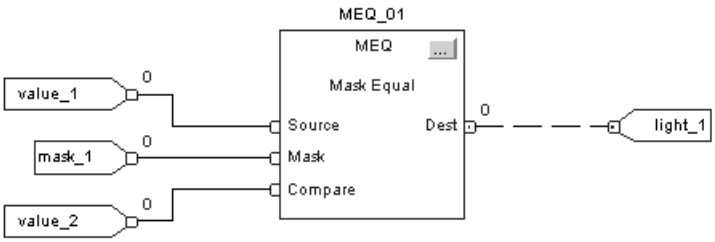
value_2	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0
mask_1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
屏蔽后的 value_2	0	1	0	1	0	1	0	1	1	1	1	1	x	x	x	x

梯形图

结构化文本

```
light_1 := ((value_1 AND mask_1)=(value_2 AND mask_2));
```

功能块



示例 **2**： 如果屏蔽后的 *value_1* 等于屏蔽后的 *value_2*，置位 light_1。如果屏蔽后的 *value_1* 不等于屏蔽后的 *value_2*，清零 *light_1*。该示例显示屏蔽后的值不相等。屏蔽码中的 0 可以阻止指令比较该位 (在示例中用 x 表示)。

value_1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
mask_1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
屏蔽后的 value_1	x	x	x	x	x	x	x	x	x	x	x	x	1	1	1	1

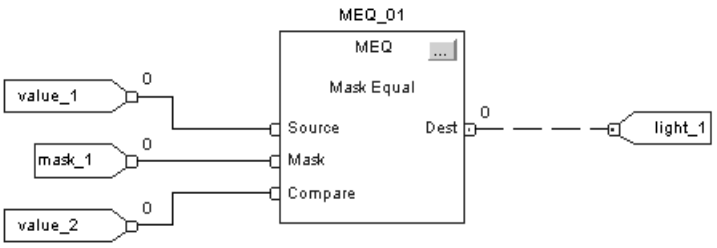
value_2	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0
mask_1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
屏蔽后的 value_2	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0

梯形图

结构化文本

```
light_1 := ((value_1 AND mask_1)=(value_2 AND mask_2));
```

功能块

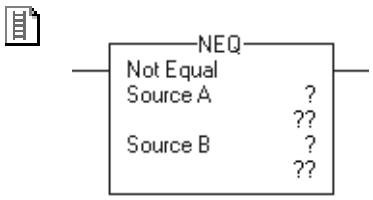


不等于 (NEQ)

NEQ 指令用于检验 Source A 是否不等于 Source B。

操作数：

梯形图



操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source B 进行比较的值
源 B (Source B)	SINT INT **DINT** **REAL** 字符串	立即数　标签	与 Source A 进行比较的值

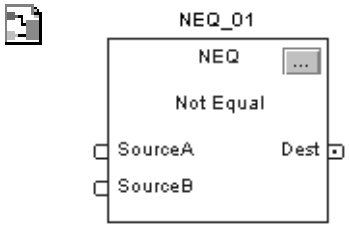
- 如果键入 SINT 或 INT 标签，值将通过符号扩展转换为 DINT 值。
- 字符串数据类型包括：
  - 默认的 STRING 数据类型。
  - 所创建的任何新的字符串数据类型。
- 要检验字符串的字符，请输入 Source A 和 Source B 的字符串标签。

结构化文本

```
IF sourceA <> sourceB THEN
    <statements>;
```

在表达式中，放在一起的小于号和大于号 “ <> ” 用作运算符。 该表达式评估 *sourceA* 是否不等于 *sourceB*。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。



功能块

操作数	类型	格式	说明
NEQ 标签	FBD_COMPARE	结构	NEQ 结构

FBD_COMPARE 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	与 SourceB 进行比较的值。 任何浮点数都有效
SourceB	REAL	与 SourceA 进行比较的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	BOOL	指令的结果。这等效于梯形图 NEQ 指令的梯级输出条件。

说明： NEQ 指令用于检验 Source A 是否不等于 Source B。

比较字符串时：

- 只要字符串中存在不匹配的字符，字符串就不相等。
- ASCII 字符区分大小写。大写的“ A ”($41) 不等于小写的“ a ”($61)。

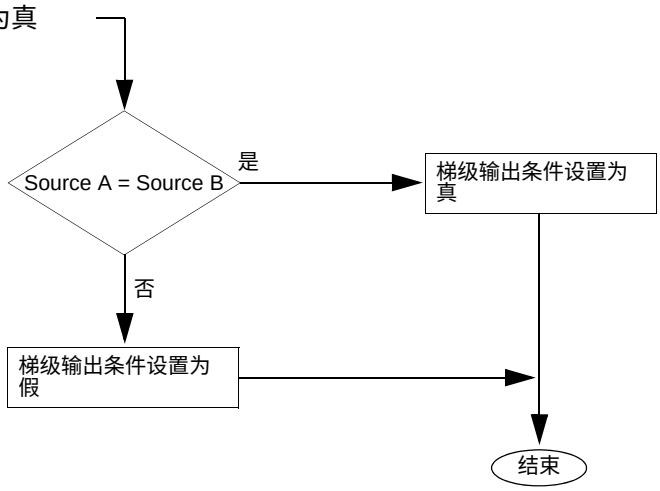
ASCII 字符	十六进制码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

算术状态标志： 不受影响

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

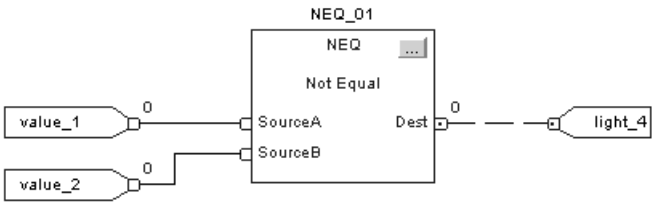
示例： 如果 *value_1* 不等于 *value_2*，置位 *light_4*。如果 *value_1* 等于 *value_2*，清零 *light_4*。

梯形图

结构化文本

```
light_4 := (value_1 <> value_2);
```

功能块



注：

计算 / 数学指令
(CPT、 ADD、 SUB、 MUL、 DIV、 MOD、 SQR、 SQRT、 NEG、 ABS)

简介

计算 / 数学指令可利用表达式或特定的算术指令来进行算术运算。

如果要	使用以下指令	在以下语言中可用	页码
计算表达式的值	CPT	梯形图 结构化文本(1)	256
将两个值相加	ADD	梯形图 结构化文本(2) 功能块	260
将两个值相减	SUB	梯形图 结构化文本 (2) 功能块	263
将两个值相乘	MUL	梯形图 结构化文本 (2) 功能块	266
将两个值相除	DIV	梯形图 结构化文本 (2) 功能块	269
确定一个值被另一个值除后所得的余数	MOD	梯形图 结构化文本 (2) 功能块	274
计算值的平方根	SQR SQRT(3)	梯形图 结构化文本 功能块	278
将值的符号变成相反的符号	NEG	梯形图 结构化文本 (2) 功能块	282
取值的绝对值	ABS	梯形图 结构化文本 功能块	285

(1) 没有对等的结构化文本指令。使用其它结构化文本编程可获得相同结果。请参见指令说明。

(2) 没有对等的结构化文本指令。在表达式中使用操作符。

(3) 仅限结构化文本。

可以混合使用不同数据类型，但可能会造成精度降低和舍入错误，而且指令的执行时间也会变长。请检查 S:V 位，查看结果是否被截断。

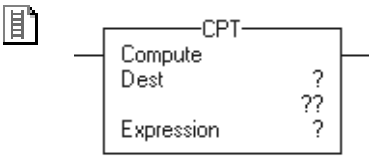
对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

计算 (CPT)

CPT 指令用于执行在表达式中定义的算术运算。

操作数：

梯形图



操作数	类型	格式	说明
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签
表达式 (Expression)	SINT INT **DINT** **REAL**	立即数　标签	由标签和／或立即值 (被运算符隔开) 组成的表达式
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		

结构化文本

结构化文本没有 CPT 指令，但可通过赋值语句和表达式获得相同结果。

```
destination := numeric_expression;
```

有关结构化文本中的赋值和表达式的语法的信息，请参见结构化文本编程。

说明：　CPT 指令用于执行在表达式中定义的算术运算。使能后， CPT 指令将计算表达式的值，然后将结果放到 Destination 中。

与其它计算 / 数学比较指令相比， CPT 指令的执行速度略慢，所用的内存更多。CPT 指令的优点是它允许在一条指令中输入多个复杂的表达式。

提 示　对表达式的长度没有限制。

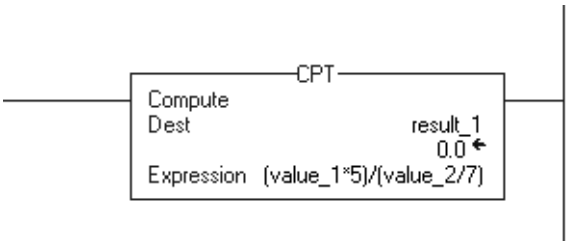
算术状态标志：　算术状态标志将受到影响。

错误条件：　无

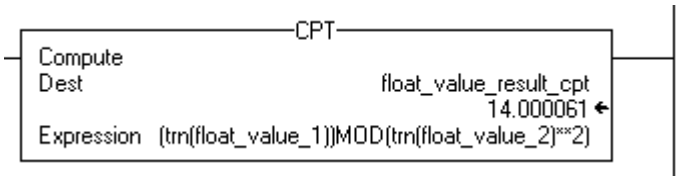
执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	指令将计算表达式的值，然后将结果放在 Destination 中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

示例 **1**： 使能后， CPT 指令将分别计算 *value_1* 乘以 5 和 *value_2* 除以 7 的结果，再用第一个结果除以第二个结果，并将最终结果放到 *result_1* 中。



示例 **2**： 使能后，CPT 指令将截断 *float_value_1* 和 *float_value_2*，然后将截断后的 *float_value_2* 平方，再用所得的结果去除截断后的 *float_value_1*，然后将相除所得的余数存储在 *float_value_result_cpt* 中。



有效运算符

运算符	说明	最佳类型
+	加	DINT、 REAL
-	减 / 取反	DINT、 REAL
*	乘	DINT、 REAL
/	除	DINT、 REAL
**	指数 (x 的 y 次幂)	DINT、 REAL
ABS	绝对值	DINT、 REAL
ACS	反余弦	REAL
AND	按位与	DINT
ASN	反正弦	REAL
ATN	反正切	REAL
COS	余弦	REAL
DEG	弧度转角度	DINT、 REAL
FRD	BCD 转整数	DINT

运算符	说明	最佳类型
LN	自然对数	REAL
LOG	以 10 为底的对数	REAL
MOD	求模除法	DINT、 REAL
NOT	按位取反	DINT
OR	按位或	DINT
RAD	角度转弧度	DINT、 REAL
SIN	正弦	REAL
SQR	平方根	DINT、 REAL
TAN	正切	REAL
TOD	整数转 BCD	DINT
TRN	截断	DINT、 REAL
XOR	按位异或	DINT

格式表达式

对于表达式中使用的每一个运算符，都需要提供一个或两个操作数 (标签或立即值)。有关表达式中运算符和操作数的格式，请遵循下表中的规定。

运算符涉及的操作数的个数	使用的格式	示例
一个操作数	运算符 (操作数)	ABS(*tag_a*)
两个操作数	操作数 _a 运算符 操作数 _b	· *tag_b* + 5 · *tag_c* AND *tag_d* · (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

确定运算顺序

写入表达式中的运算将由指令按照规定的顺序执行，不一定按照写入的顺序执行。通过用圆括号括起一些内容，可以更改运算顺序，这样指令就会先执行括号内的运算，后执行其它运算。

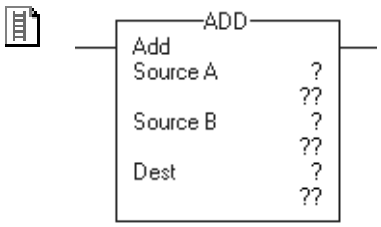
相同顺序的运算从左到右执行。

顺序	运算
1.	( )
2.	ABS、 ACS、 ASN、 ATN、 COS、 DEG、FRD、LN、LOG、RAD、SIN、 SQR、 TAN、 TOD、 TRN
3.	**
4.	– (取反)， NOT
5.	*、 /、 MOD
6.	– (减)， +
7.	AND
8.	XOR
9.	OR

加 (ADD)

ADD 指令用于将 Source A 与 Source B 相加并将结果放到 Destination 中。

操作数：



梯形图

操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL**	立即数 标签	与 Source B 相加的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
源 B (Source B)	SINT INT **DINT** **REAL**	立即数 标签	与 Source A 相加的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

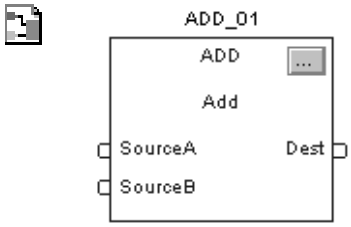
结构化文本

```
dest := sourceA + sourceB;
```

在表达式中，加号 “ + ” 用作运算符。该表达式将 *sourceA* 与 *sourceB* 相加，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
ADD 标签	FBD_MATH	结构	ADD 结构

FBD_MATH 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	与 SourceB 相加的值。 任何浮点数都有效
SourceB	REAL	与 SourceA 相加的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： ADD 指令用于将 Source A 与 Source B 相加并将结果放到 Destination 中。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

梯形图

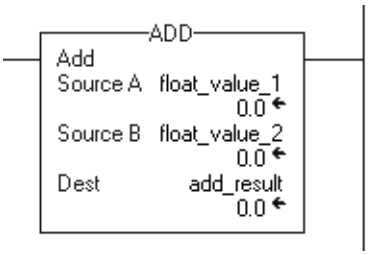
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	Destination = Source A + Source B 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 将 *float_value_1* 与 *float_value_2* 相加，然后将结果放在 *add_result* 中。

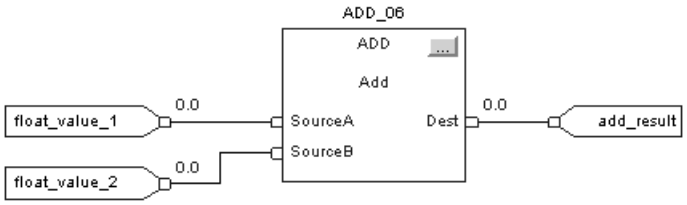
梯形图



结构化文本

```
add_result := float_value_1 + float_value_2;
```

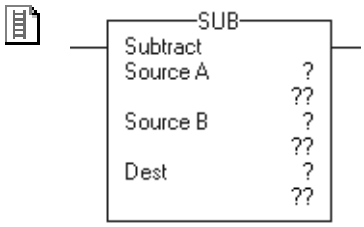
功能块



减 (SUB)

SUB 指令用于从 Source A 中减去 Source B 并将结果放在 Destination 中。

操作数：



梯形图

操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL** SINT 或 INT 标签将通过符号扩展转换为 DINT 值。	立即数 标签	从其减去 Source B 的值
源 B (Source B)	SINT INT **DINT** **REAL** SINT 或 INT 标签将通过符号扩展转换为 DINT 值。	立即数 标签	从 Source A 中减掉的值
Destination	SINT INT **DINT** **REAL**	标签	要存储结果的标签

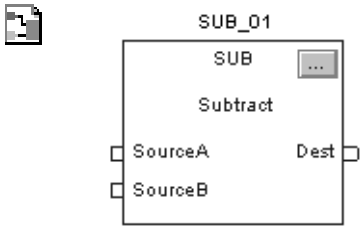
结构化文本

```
dest := sourceA - sourceB;
```

在表达式中，减号 “ – ” 用作运算符。该表达式从 *sourceA* 中减去 *sourceB*，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
SUB 标签	FBD_MATH	结构体	SUB 结构

FBD_MATH 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	REAL	从其减去 SourceB 的值。 任何浮点数都有效
SourceB	REAL	从 SourceA 中减掉的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： SUB 指令用于从 Source A 中减去 Source B 并将结果放在 Destination 中。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

梯形图

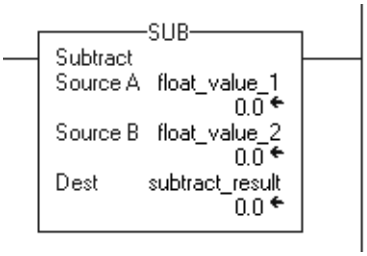
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	Destination = Source B - Source A 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 从 *float_value_1* 中减去 *float_value_2*，然后将结果放在 *subtract_result* 中。

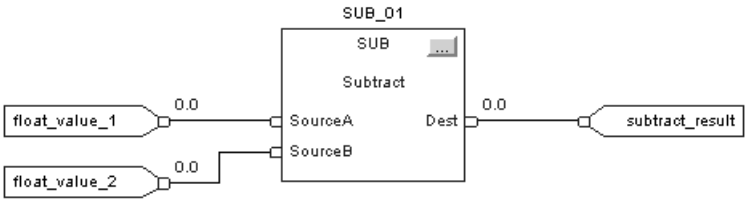
梯形图



结构化文本

```
subtract_result := float_value_1 - float_value_2;
```

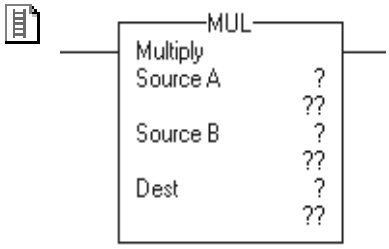
功能块



乘 (MUL)

MUL 指令用于将 Source A 与 Source B 相乘，并将结果放在 Destination 中。

操作数：



梯形图

操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL**	立即数　标签	被乘数的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
源 B (Source B)	SINT INT **DINT** **REAL**	立即数　标签	乘数的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

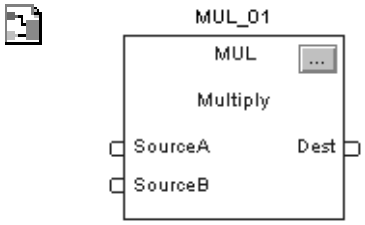
结构化文本

```
dest := sourceA * sourceB;
```

在表达式中，乘号 “ * ” 用作运算符。该表达式将 *sourceA* 与 *sourceB* 相乘，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
MUL 标签	FBD_MATH	结构	MUL 结构

FBD_MATH 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 A (Source A)	REAL	被乘数的值。 任何浮点数都有效
源 B (Source B)	REAL	乘数的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： MUL 指令用于将 Source A 与 Source B 相乘，并将结果放在 Destination 中。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

梯形图

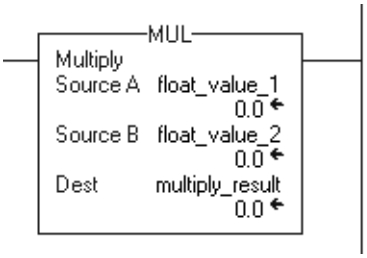
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	Destination = Source B x Source A 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 将 *float_value_1* 与 *float_value_2* 相乘，然后将结果放在 *multiply_result* 中。

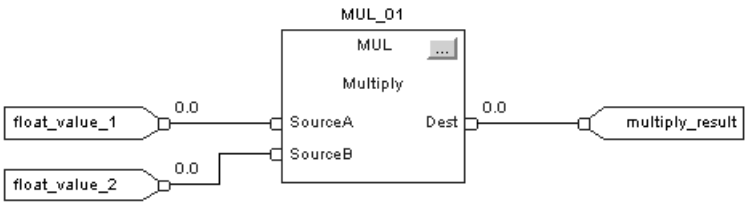
梯形图



结构化文本

```
multiply_result := float_value_1 * float_value_2;
```

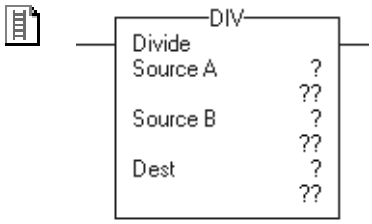
功能块



除 (DIV)

DIV 指令用 Source B 去除 Source A，并将结果放在 Destination 中。

操作数：



梯形图

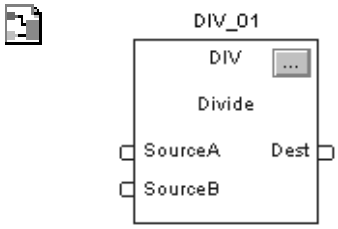
操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL**	立即数 标签	被除数的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
源 B (Source B)	SINT INT **DINT** **REAL**	立即数 标签	除数的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

结构化文本

```
dest := sourceA / sourceB;
```

在表达式中，除号 “ / ” 用作运算符。该表达式用 *sourceB* 去除 *sourceA*，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。



功能块

操作数	类型	格式	说明
DIV 标签	FBD_MATH	结构	DIV 结构

FBD_MATH 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 A (Source A)	REAL	被除数的值。 任何浮点数都有效
源 B (Source B)	REAL	除数的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 如果 Destination 不是 REAL 类型，指令将按照下述方法处理结果的小数部分：

如果 source A	则结果的小数部分	示例		
和 Source B *不是* REAL 类型	截断	源 A (Source A)	DINT	5
		源 B (Source B)	DINT	3
		Destination	DINT	1
或 Source B 是 REAL 类型	舍入	源 A (Source A)	REAL	5.0
		源 B (Source B)	DINT	3
		Destination	DINT	2

如果 Source B(除数) 是零：

- 将出现一个次要错误：
  - 类型 4：程序错误
  - 代码 4：算术溢出
- 如下设置 Destination：

如果 source B 是零并且	并且 Destination 是	而且结果是	则 Destination 设置为
所有操作数都是整数 (SINT、INT 或 DINT)	→	→	源 A (Source A)
至少一个操作数是 REAL	SINT、 INT 或 DINT	正数	-1
		负数	0
	REAL	正数	1.$(正无穷)
		负数	-1.$(负无穷)

要检测可能出现的除数为零的情况，请检查次要错误位 (S:MINOR)。
请参见 《Logix5000 控制器通用编程手册》 (出版号 1756-PM001)。

算术状态标志： 算术状态标志将受到影响。

错误条件：

次要错误的发生条件	错误类型	错误代码
除数是零	4	4

执行：

梯形图

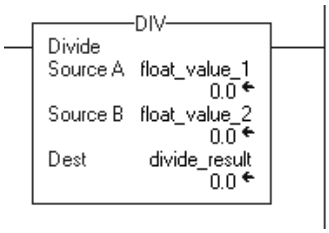
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	Destination = Source A / Source B 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例 **1**： 用 *float_value_2* 去除 *float_value_1*，然后将结果放在 *divide_result* 中。

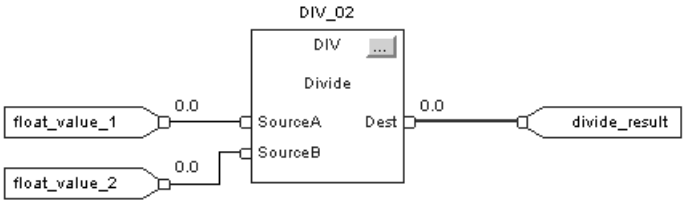
梯形图



结构化文本

```
divide_result := float_value_1 / float_value_2;
```

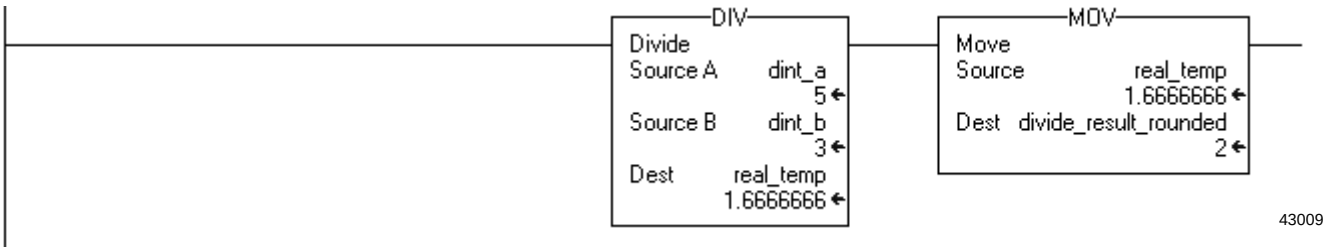
功能块



示例 **2**： DIV 和 MOV 指令配合使用，可以将两个整数相除、舍入结果并将结果放在整型标签中：

- DIV 指令用 *dint_b* 去除 *dint_a*。
- 要舍入结果， Destination 需要是 REAL 标签。 (如果 Destination 是整型标签 (SINT、 INT 或 DINT)，指令将截断结果。)
- MOV 指令将舍入后的结果 (*real_temp*) 从 DIV 移动到 *divide_result_rounded*。
- 由于 *divide_result_rounded* 是 DINT 标签，所以 *real_temp* 的值将被舍入并放在 DINT Destination 中。

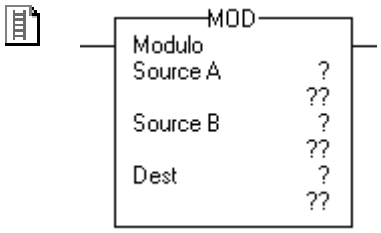
梯形图



求模 (MOD)

MOD 指令用 Source B 去除 Source A，并将余数放在 Destination 中。

操作数：



梯形图

操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT** **REAL**	立即数　标签	被除数的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
源 B (Source B)	SINT INT **DINT** **REAL**	立即数　标签	除数的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

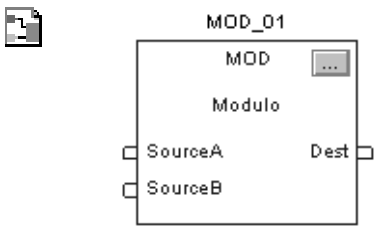
结构化文本

```
dest := sourceA MOD sourceB;
```

在表达式中， MOD 用作运算符。该表达式用 *sourceB* 去除 *sourceA*，并将余数存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
MOD 标签	FBD_MATH	结构	MOD 结构

FBD_MATH 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 A (Source A)	REAL	被除数的值。 任何浮点数都有效
源 B (Source B)	REAL	除数的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 如果 Source B(除数) 是零：

- 将出现一个次要错误：
  - 类型 4：程序错误
  - 代码 4：算术溢出
- 如下设置 Destination：

如果 source B 是零并且	并且 Destination 是	而且结果是	则 Destination 设置为
所有操作数都是整数 (SINT、INT 或 DINT)	→	→	源 A (Source A)
至少一个操作数是 REAL	SINT、 INT 或 DINT	正数	-1
		负数	0
	REAL	正数	1.$(正无穷)
		负数	-1.$(负无穷)

要检测可能出现的除数为零的情况，请检查次要错误位 (S:MINOR)。
请参见 《Logix5000 控制器通用编程手册》 (出版号 1756-PM001)。

算术状态标志： 算术状态标志将受到影响。

错误条件：

次要错误的发生条件	错误类型	错误代码
除数是零	4	4

执行：

梯形图

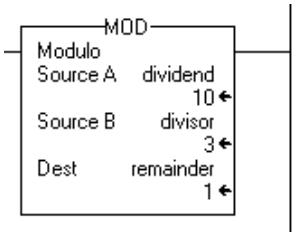
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	Destination = Source A　(TRN (Source A / Source B) * Source B) 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
后扫描	不执行任何操作。

示例： 用 *divisor* 去除 *dividend*，并将余数放在 *remainder* 中。在本示例中，3 除 10 得 3 余 1。

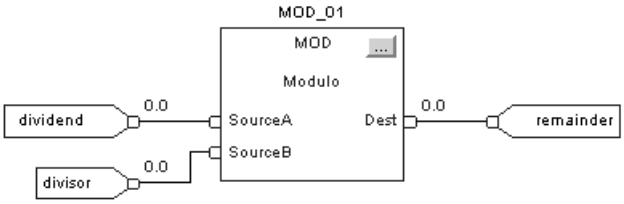
梯形图



结构化文本

```
remainder := dividend MOD divisor;
```

功能块

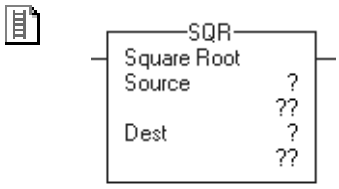


平方根 (SQR)

SQR 指令用于计算 Source 的平方根并将结果放在 Destination 中。

操作数：

梯形图



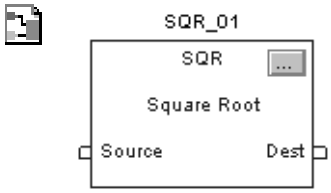
操作数	类型	格式	说明
源 (Source)	SINT INT **DINT** **REAL**	立即数 标签	求该值的平方根
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

结构化文本

```
dest := SQRT(source);
```

使用 SQRT 作为函数。该表达式计算 *source* 的平方根，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。



功能块

操作数	类型	格式	说明
SQR 标签	FBD_MATH_ADVANCED	结构	SQR 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
Source	REAL	求该值的平方根。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 如果 Destination 不是 REAL 类型，指令将按照下述方法处理结果的小数部分：

如果 Source	则结果的小数部分	示例		
不是 REAL	截断	Source	DINT	3
		Destination	DINT	1
是 REAL	舍入	Source	REAL	3.0
		Destination	DINT	2

如果 Source 是负数，则指令将取 Source 的绝对值，然后再计算平方根。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

梯形图

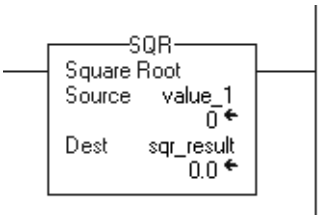
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	$Destination = \sqrt{Source}$ 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value_1* 的平方根，并将结果放在 *sqr_result* 中。

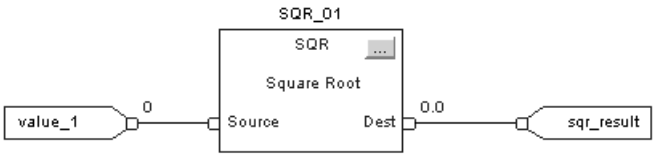
梯形图



结构化文本

```
sqr_result := SQRT(value_1);
```

功能块

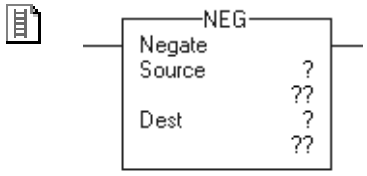


取反 (NEG)

NEG 指令用于改变 Source 的符号，并将结果放在 Destination 中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT **DINT** **REAL**	立即数　标签	要取反的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

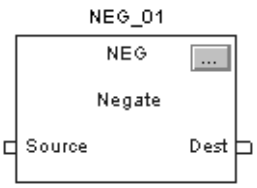
结构化文本

```
dest := -source;
```

在表达式中，负号 “ – ” 用作运算符。该表达式更改 *source* 的符号，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
NEG 标签	FBD_MATH_ADVANCED	结构	NEG 结构

FBD_MATH 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
Source	REAL	要取反的值 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 如果对负值取反，则结果为正数。如果对正值取反，则结果为负数。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

梯形图

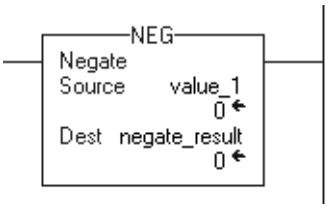
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	Destination = 0 – Source 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 更改 *value_1* 的符号，并将结果放在 *negate_result* 中。

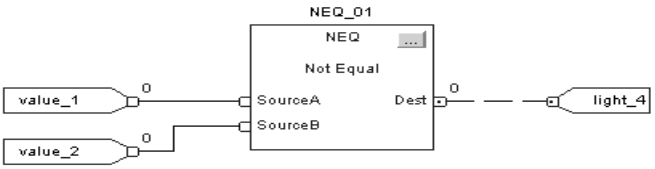
梯形图



结构化文本

```
negate_result := -value_1;
```

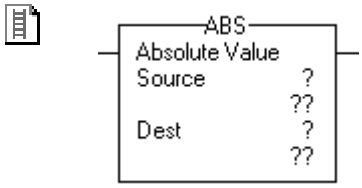
功能块



绝对值 (ABS)

ABS 指令用于取 Source 的绝对值，并将结果放在 Destination 中。

操作数：



梯形图

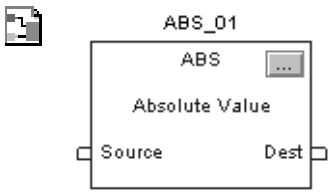
操作数	类型	格式	说明
源 (Source)	SINT INT **DINT** **REAL**	立即数 标签	要取绝对值的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

结构化文本

```
dest := ABS(source);
```

使用 ABS 作为函数。该表达式计算 *source* 的绝对值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。



功能块

操作数	类型	格式	说明
ABS 标签	FBD_MATH_ADVANCED	结构	ABS 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
Source	REAL	要取绝对值的值。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： ABS 指令用于取 Source 的绝对值，并将结果放在 Destination 中。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

梯形图

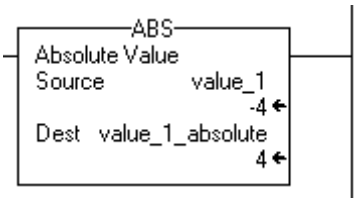
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	Destination = | Source | 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 将 *value_1* 的绝对值放在 *value_1_absolute* 中。在本示例中， -4 的绝对值是 +4。

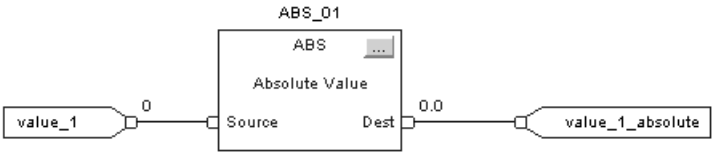
梯形图



结构化文本

```
value_1_absolute := ABS(value_1);
```

功能块



注：

移动 / 逻辑指令

(MOV、 MVM、 BTD、 MVMT、 BTDT、 CLR、 SWPB、 AND、OR、XOR、NOT、BAND、BOR、BXOR、BNOT)

简介

可以混合使用不同数据类型，但可能会造成精度降低和舍入错误，而且指令的执行时间也会变长。请检查 S:V 位，查看结果是否被截断。

对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

移动指令用于修改和移动位。

如果要	使用以下指令	在以下语言中可用	页码
复制一个值	MOV	梯形图 结构化文本(1)	291
复制整数的特定部分	MVM	梯形图	293
复制功能块中某个整数的特定部分	MVMT	结构化文本 功能块	296
在一个整数内或在整数之间移动位	BTD	梯形图	299
在功能块的一个整数内或整数之间移动位	BTDT	结构化文本 功能块	302
将值清零	CLR	结构化文本 (1) 梯形图	305
重新排列 INT、 DINT 或 REAL 标签的字节	SWPB	梯形图 结构化文本	307

(1) 没有对等的结构化文本指令。使用其它结构化文本编程可获得相同结果。请参见指令说明。

逻辑指令可以对位进行运算。

如果要	使用以下指令	在以下语言中可用	页码
按位与运算	按位与 &(1)	梯形图 结构化文本(2) 功能块	311
按位或运算	按位或	梯形图 结构化文本 (2) 功能块	314
按位异或运算	按位异或	梯形图 结构化文本 (2) 功能块	318
按位非运算	按位非	梯形图 结构化文本 (2) 功能块	322
对八个布尔输入进行逻辑与运算	布尔型与运算 (BAND)	结构化文本 (2) 功能块	325
对八个布尔输入进行逻辑或运算	布尔型或运算 (BOR)	结构化文本 (2) 功能块	328
对两个布尔输入执行异或运算。	布尔异或 (BXOR)	结构化文本 (2) 功能块	331
对布尔输入取反。	布尔型非运算 (BNOT)	结构化文本 (2) 功能块	334

(1) 仅限结构化文本。

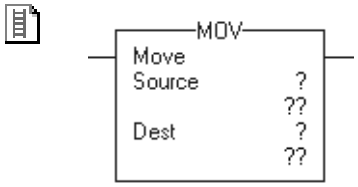
(2) 在结构化文本中， AND、 OR、 XOR 和 NOT 运算可以是按位运算，也可以是逻辑运算。

移动 (MOV)

MOV 指令用于将 Source 复制到 Destination。 Source 保持不变。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT **DINT** **REAL**	立即数 标签	要移动 (复制) 的值
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要存储结果的标签

结构化文本

```
dest := source;
```

将赋值语句 “ := ” 和表达式结合使用。该赋值语句将 *source* 的值移动到 *dest*。

有关结构化文本中表达式和赋值的语法的信息，请参见结构化文本编程。

说明： MOV 指令用于将 Source 复制到 Destination。 Source 保持不变。

算术状态标志： 算术状态标志将受到影响。

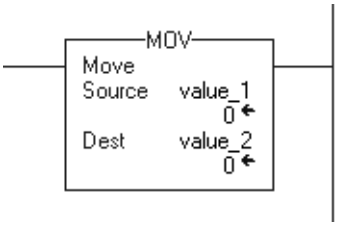
错误条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	该指令将 Source 复制到 Destination。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

示例： 将 *value_1* 中的数据移动到 *value_2*。

梯形图



结构化文本

```
value_2 := value _1;
```

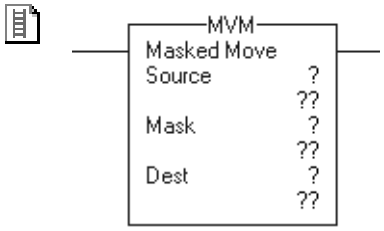
屏蔽移动 (MVM)

MVM 指令用于将 Source 复制到 Destination，并允许部分数据被屏蔽。

该指令在结构化文本和功能块中以 MVMT 形式提供，请参见第 296 页。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT **DINT**	立即数 标签	要移动的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
屏蔽码 (Mask)	SINT INT **DINT**	立即数 标签	要阻止或传递的位
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT**	标签	要存储结果的标签

结构化文本

```
dest := (Dest AND NOT (Mask))
    OR (Source AND Mask);
```

该指令在结构化文本中以 MVMT 形式提供。也可以在表达式中将按位逻辑组合起来，然后将结果分配给 Destination。该表达式可以对 *Source* 执行屏蔽移动。

有关结构化文本中表达式和赋值的语法的信息，请参见结构化文本编程。

说明：MVM 指令使用 Mask 来传递或阻止 Source 数据位。屏蔽码中的“ 1 ”表示该数据位已传递。屏蔽码中的“ 0 ”表示该数据位已阻止。

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

键入立即数屏蔽码值

键入屏蔽码时，编程软件将默认为十进制值。如果想使用另一种格式键入屏蔽码，请在值的前面加上正确的前缀。

前缀	说明
16#	十六进制 例如， 16#0F0F
8#	八进制 例如， 8#16
2#	二进制 例如， 2#00110011

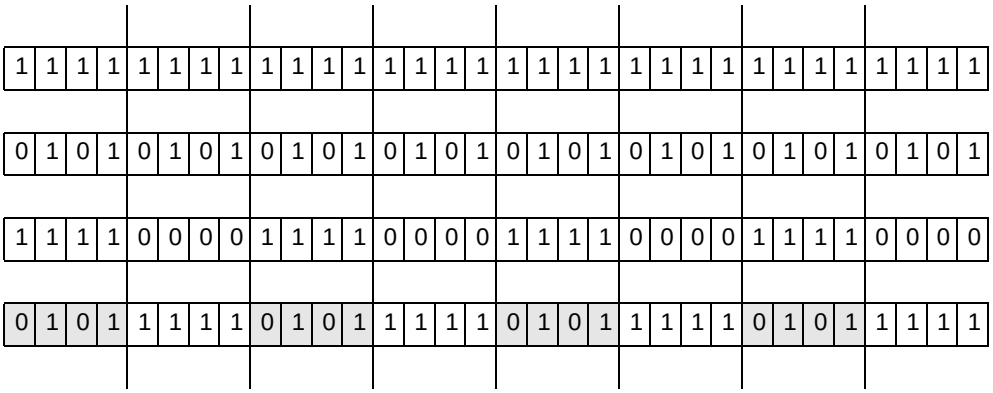
算术状态标志　算术状态标志将受到影响。

错误条件　无

执行：

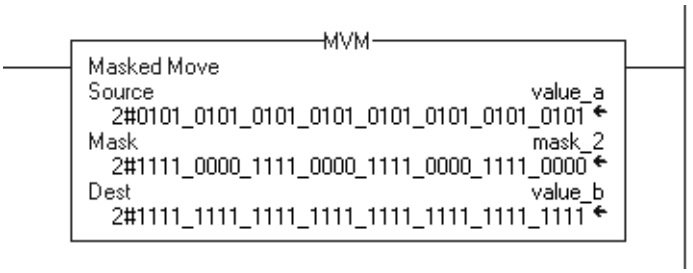
条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	该指令通过 Mask 传递 Source，并将结果复制到 Destination 中。Destination 中未屏蔽的位将保持不变。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

示例： 将数据从 *value_a* 复制到 *value_b*，同时允许数据被屏蔽 (0 可以屏蔽 *value_a* 中的数据)。



有阴影的框显示的是在 *value_b* 中发生变化的位。

梯形图



结构化文本

```
value_b := (value_b AND NOT (mask_2)) OR
    (value_a AND mask_2);
```

带目标屏蔽移动 (MVMT)

MVMT 指令先将 Target 复制到 Destination。然后用屏蔽后的 Source 同 Destination 相比较，并对 Destination 进行必要改动。 Target 和 Source 保持不变。

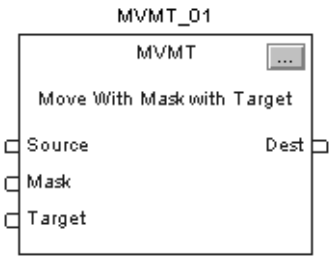
该指令在梯形图中以 MVM 形式提供，请参见第 293 页。

操作数：

```
MVMT(MVMT_tag);
```

结构化文本

变量	类型	格式	说明：
MVMT 标签	FBD_MASKED_MOVE	结构	MVMT 结构



功能块

操作数	类型	格式	说明
MVMT 标签	FBD_MASKED_MOVE	结构	MVMT 结构

FBD_MASKED_MOVE 结构

输入参数	数据类型	说明
EnableIn	BOOL	功能块 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本 无影响。执行该指令。
Source	DINT	要基于 Mask 的值移动到 Destination 的输入值。 任何整数都有效
Mask	DINT	要从 Source 移动到 Dest 的位的屏蔽码。所有设置为 1 的位都可使相应的位从 Source 移动到 Dest。所有设置为 0 的位都可使相应的位不从 Source 移动到 Dest。 任何整数都有效
Target	DINT	通过 Mask 移动 Source 位前要移动到 Dest 的输入值。 任何整数都有效

输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	DINT	屏蔽移动指令的结果。将为该输出设置算术状态标志。

说明： 使能后， MVMT 指令将使用 Mask 来传递或阻止 Source 数据位。屏蔽码中的“ 1 ”表示该数据位已传递。屏蔽码中的“ 0 ”表示该数据位已阻止。

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

键入立即数屏蔽码值

键入屏蔽码时，编程软件将默认为十进制值。如果想使用另一种格式键入屏蔽码，请在值的前面加上正确的前缀。

前缀	说明
16#	十六进制 例如， 16#0F0F
8#	八进制 例如， 8#16
2#	二进制 例如， 2#00110011

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

条件	功能块操作	结构化文本操作
预扫描	不执行任何操作。	不执行任何操作。
指令第一次扫描	不执行任何操作。	不执行任何操作。
指令第一次运行	不执行任何操作。	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	N/A
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。	EnableIn 始终置位。 执行该指令。
后扫描	不执行任何操作。	不执行任何操作。

示例：　**1.** 将 Target 复制到 Dest。

Target	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Dest	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

2. 屏蔽 Source，并将其与 Dest 比较。在 Dest 中进行必要改变。Source 和 Target 将保持不变。屏蔽码中的 0 可以阻止指令比较该位 (在示例中用 x 表示)。

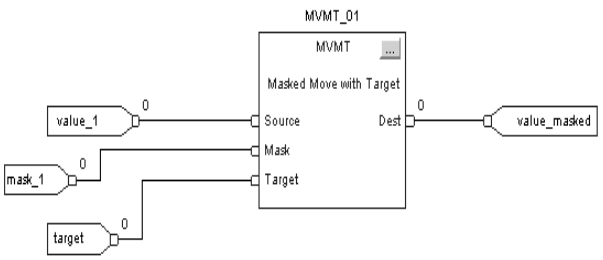
Source	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Mask1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
Dest	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1	1

有阴影的框显示的是发生改变的位。

结构化文本

```
MVMT_01.Source := value_1;
MVMT_01.Mask := mask1;
MVMT_01.Target := target;

MVMT(MVMT_01);

value_masked := MVMT_01.Dest;
```

功能块



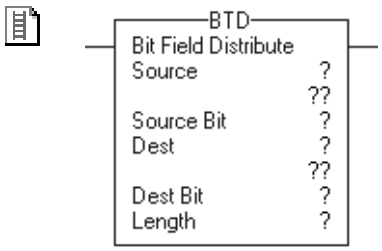
位域分配 (BTD)

BTD 指令用于从 Source 中复制指定的位，然后将这些位转移到合适的位置，再将这些位写入 Destination 中。

该指令在结构化文本和功能块中以 BTDT 形式提供。
请参见第 302 页。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT **DINT**	立即数 标签	含有要移动的位的标签
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
源位 (Source bit)	DINT	立即数 (0-31 DINT) (0-15 INT) (0-7 SINT)	从此开始移动的位的编号 (最低位编号) 必须位于 Source 数据类型的有效范围内
目标 (Destination)	SINT INT **DINT**	标签	向其中移动位的标签
目标位 (Destination bit)	DINT	立即数 (0-31 DINT) (0-15 INT) (0-7 SINT)	从此开始从 Source 中复制位的位的编号 (最低位编号) 必须位于 Destination 数据类型的有效范围内
长度 (Length)	DINT	立即值 (1-32)	要移动的位的个数

说明：使能后， BTD 指令将从 Source 中复制一组位到 Destination。这组位将通过 Source bit(该组的最低位编号) 和 Length(要复制的位的个数) 来标识。Destination bit 标识 Destination 中的起始最低位编号。Source 保持不变。

如果位域的长度超过 Destination，指令将不会保存超出的位。超出的任何位都不会附加到下一个字上。

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

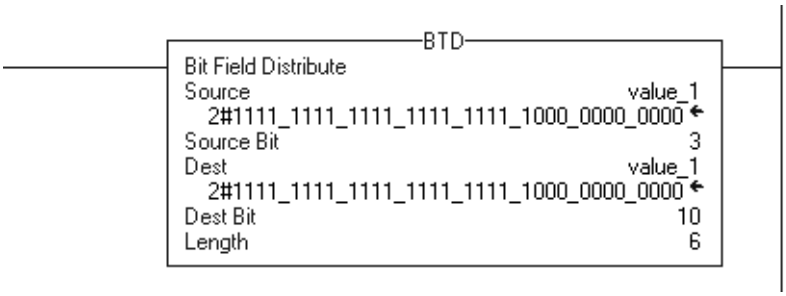
算术状态标志：不受影响

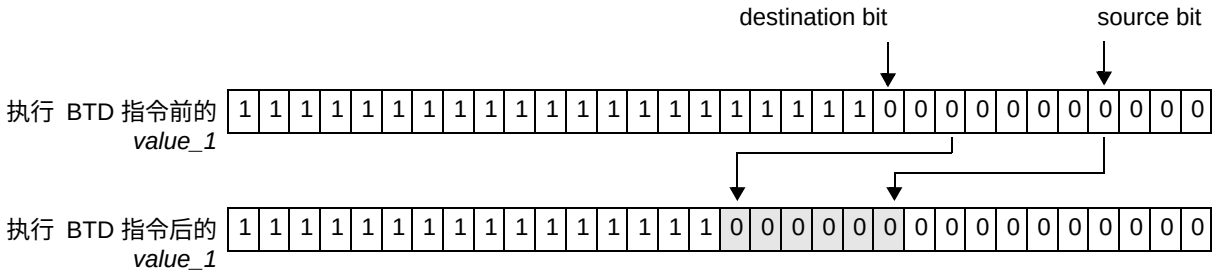
错误条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	该指令复制 Source 位并将其转移到 Destination。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

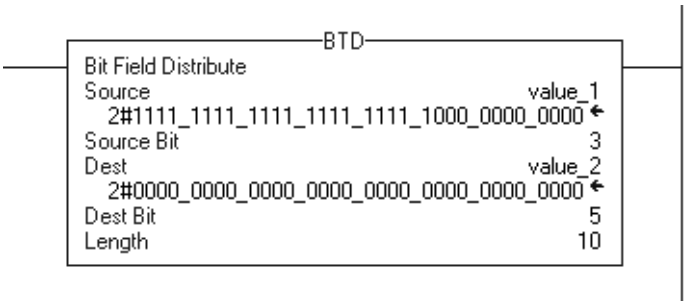
示例 **1**： 使能后， BTD 指令将在 *value_1* 内移动位。

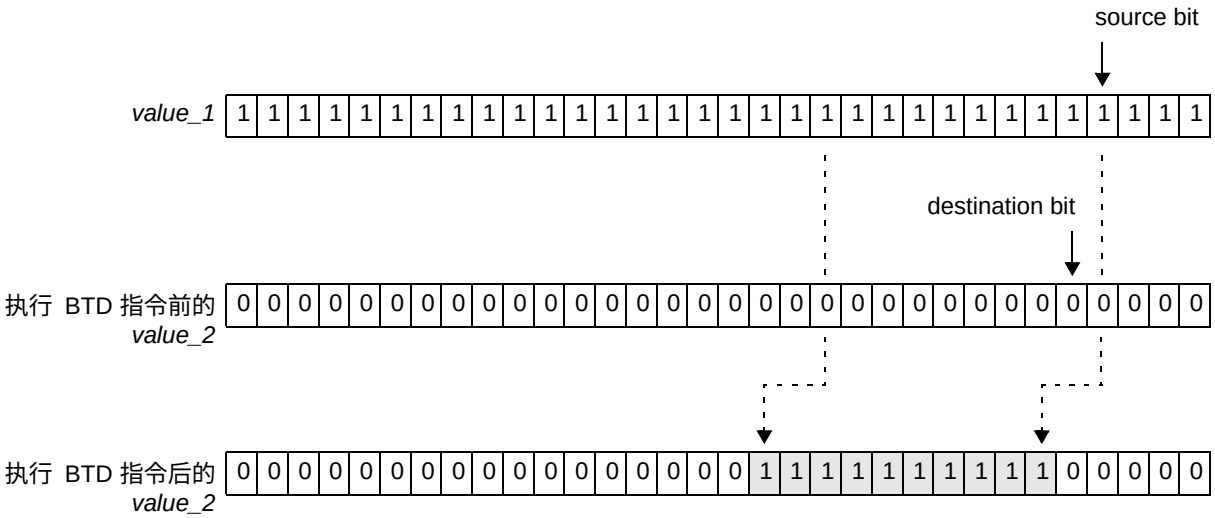




有阴影的框显示的是在 *value_1* 中发生变化的位。

示例 **2**： 使能后， BTD 指令将 *value_1* 中的 10 个位移动到 *value_2*。





有阴影的框显示的是在 *value_2* 中发生变化的位。

带目标的位域分配 (BTDT)

BTDT 指令首先将 Target 复制到 Destination。然后，该指令从 Source 复制指定的位，并将这些位转移到合适的位置，再将这些位写入 Destination。 Target 和 Source 保持不变。

该指令在梯形图中以 BTD 形式提供，请参见第 299 页。

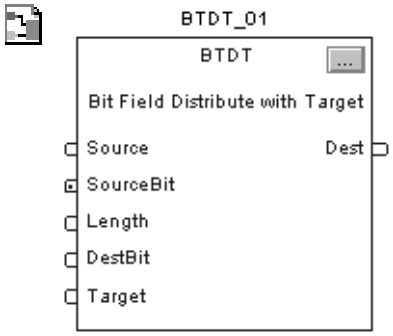
操作数：

结构化文本

```
BTDT(BTDT_tag);
```

变量	类型	格式	说明
BTDT 标签	FBD_BIT_FIELD_DISTRIBUTE	结构	BTDT 结构

功能块



操作数	类型	格式	说明
BTDT 标签	FBD_BIT_FIELD_DISTRIBUTE	结构	BTDT 结构

FBD_BIT_FIELD_DISTRIBUTE 结构

输入参数	数据类型	说明
EnableIn	BOOL	功能块： 如果为清零状态，则不执行该指令，并且不更新输出。 如果为置位状态，则执行该指令。 默认为置位状态。 结构化文本： 无影响。执行该指令。
Source	DINT	含有要移动到 Destination 的位的输入值。 任何整数都有效
SourceBit	DINT	Source 中位的位置 (开始移动时的最低位编号)。 有效值 = 0…31
Length	DINT	要移动的位的个数 有效值 = 1…32

输入参数	数据类型	说明
DestBit	DINT	Dest 中位的位置 (要开始将位复制到其中的最低位编号)。 有效值 = 0…31
Target	DINT	从 Source 移动位之前要移动到 Dest 的输入值。 任何整数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	DINT	位移动运算的结果。将为该输出设置算术状态标志。

说明： 使能后， BTD 指令将从 Source 中复制一组位到 Destination。这组位将通过 Source bit(该组的最低位编号) 和 Length(要复制的位的个数) 来标识。 Destination bit 标识 Destination 中的起始最低位编号。 Source 保持不变。

如果位域的长度超过 Destination，指令将不会保存超出的位。超出的任何位都不会附加到下一个字上。

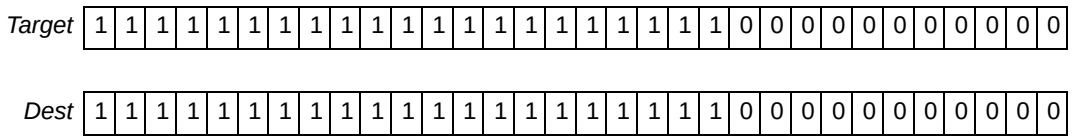
算术状态标志： 算术状态标志将受到影响

错误条件： 无

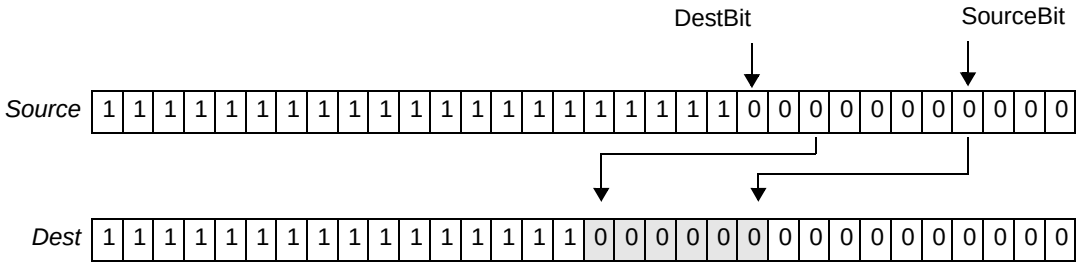
执行：

条件	功能块操作	结构化文本操作
预扫描	不执行任何操作。	不执行任何操作。
指令第一次扫描	不执行任何操作。	不执行任何操作。
指令第一次运行	不执行任何操作。	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut，指令不执行任何操作，并且不更新输出。	N/A
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。	EnableIn 始终置位。 执行该指令。
后扫描	不执行任何操作。	不执行任何操作。

示例：　**1.** 控制器将 Target 复制到 Dest 中。



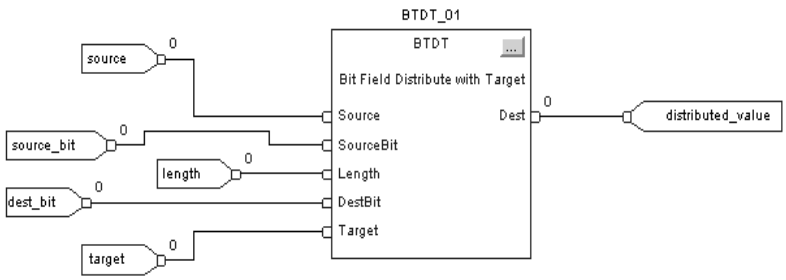
2. SourceBit 和 Length 指定 Source 中的哪些位被复制到 Dest 中 (从 DestBit 开始)。 Source 和 Target 将保持不变。



结构化文本

```
BTDT_01.Source := source;
BTDT_01.SourceBit := source_bit;
BTDT_01.Length := length;
BTDT_01.DestBit := dest_bit;
BTDT_01.Target := target;

BTDT(BTDT_01);

distributed_value := BTDT_01.Dest;
```

功能块

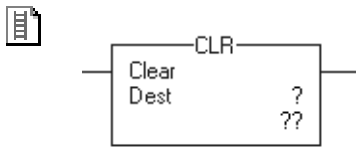


清零 (CLR)

CLR 指令用于对 Destination 的所有位清零。

操作数：

梯形图



操作数	类型	格式	说明
目标 (Destination)	SINT INT **DINT** **REAL**	标签	要清零的标签

结构化文本

```
dest := 0;
```

结构化文本没有 CLR 指令。而是将 0 赋给要清零的标签。该赋值语句将对 *dest* 清零。

有关结构化文本中表达式和赋值语句的语法的信息，请参见结构化文本编程。

说明：　CLR 指令用于对 Destination 的所有位清零。

算术状态标志：　算术状态标志将受到影响。

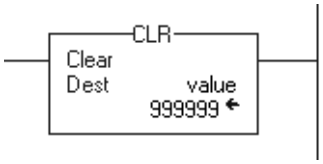
错误条件：　无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	指令将对 Destination 清零。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

示例： 将 *value* 的所有位清为 0。

梯形图



结构化文本

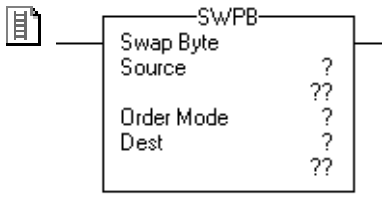
```
value := 0;
```

交换字节 (SWPB)

SWPB 指令用于重新排列值的字节。

操作数：

梯形图



操作数	类型	格式	说明		
源 (Source)	INT DINT REAL	标签	包含要重新排列的字节的标签		
顺序模式 (Order Mode)			如果 Source 为	并且想要将字节改为以下形式 (每个字母代表一个不同的字节)	则选择
			INT	N/A	任意选项
			DINT REAL	ABCD DCBA	REVERSE(或键入 0)
				ABCD CDAB	WORD(或键入 1)
				ABCD BADC	HIGH/LOW(或键入 2)
目标 (Destination)	INT DINT REAL	标签	要存储按新顺序排列的字节的标签		
			如果 Source 为	则 Destination 必须为	
			INT	INT	
				DINT	
			DINT	DINT	
			REAL	REAL	

结构化文本

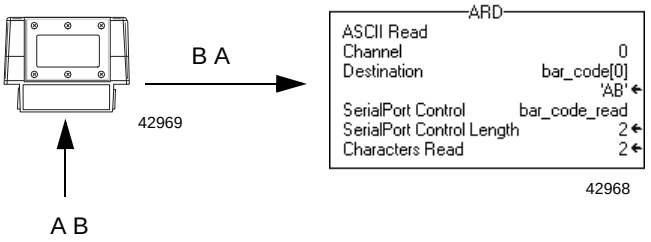
```
SWPB(Source,OrderMode,Dest);
```

操作数与梯形图 SWPB 指令的操作数相同。如果选择 HIGH/LOW 顺序模式，则输入为 HIGHLOW 或 HIGH_LOW(不带斜线)。

说明： SWPB 指令可重新排列 Source 的字节顺序。该指令会将结果放在 Destination 中。

读取或写入 ASCII 字符时，通常*不*需要交换字符。 ASCII 读写指令 (ARD、 ARL、 AWA、 AWT) 将自动交换字符，如下所示。

条码阅读器



标签名称	值	格式	类型
bar_code[0]	AB	ASCII	INT

算术状态标志： 不受影响

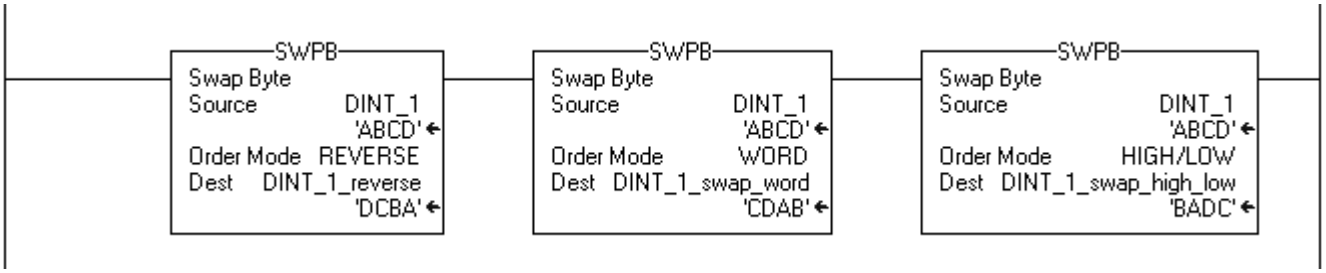
错误条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	指令将重新排列指定的字节。	指令将重新排列指定的字节。
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例 **1**： 以下三条 SWPB 指令中的每一条都将根据不同的顺序模式对 *DINT_1* 的字节重新排序。显示格式是 ASCII，每个字符表示一个字节。每条指令都将按照新顺序将字节放在一个不同的 Destination 中。

梯形图



结构化文本

```
SWPB(DINT_1,REVERSE,DINT_1_reverse);
SWPB(DINT_1,WORD,DINT_1_swap_word);
SWPB(DINT_1,HIGHLOW,DINT_1_swap_high_low);
```

示例 **2**： 以下示例反转数组的每个元素中的字节。有关含有本示例的 RSLogix 5000 项目，请打开 RSLogix 5000\Projects\Samples 文件夹中的 Swap_Bytes_in_Array.ACD 文件。

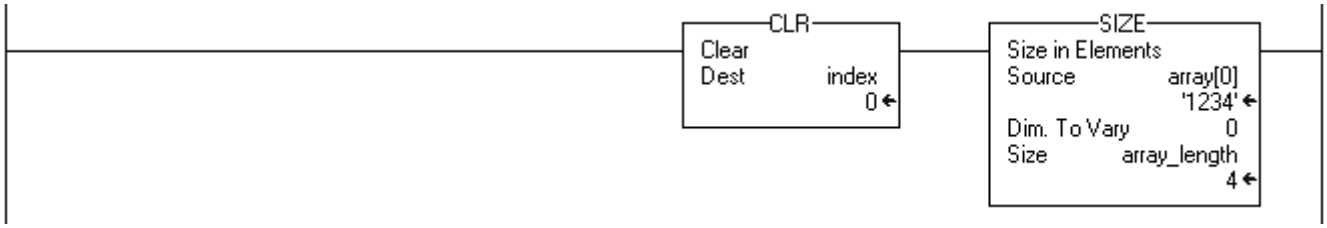
1. 初始化标签。 SIZE 指令将计算出 *array* 中的元素数，并将该值存储在 *array_length* 中。后续指令使用该值来确定例程何时作用于数组中的所有元素。

2. 反转 *array* 的一个元素中的字节。
   - SWPB 指令将反转编号由 *index* 的值指示的元素的字节。例如，当 *index* 等于 0 时， SWPB 指令将作用于 *array[0]*。
   - ADD 指令递增 *index* 值。下次指令执行时， SWPB 指令将作用于 *array* 中的下一个元素。

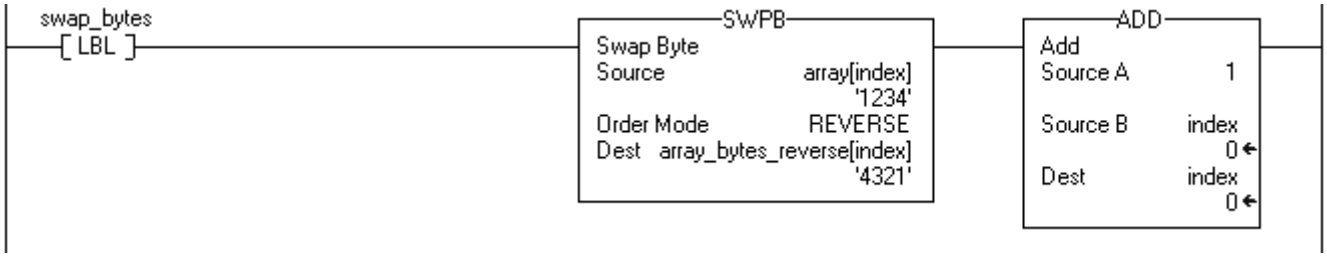
3. 确定 SWPB 指令是否已作用于数组中的所有元素。
   - 如果 *index* 小于数组的元素数 (*array_length*)，将继续作用于数组中的下一个元素。
   - 如果 *index* 等于 *array_length*，则说明 SWPB 已经作用于数组中的所有元素。

梯形图

初始化标签。



反转字节。



确定 SWPB 指令是否已作用于数组中的所有元素。

结构化文本

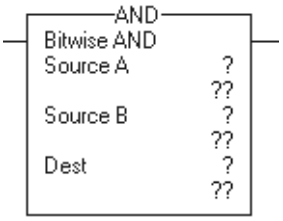
```
index := 0;
SIZE (array[0],0,array_length);
REPEAT
 SWPB(array[index],REVERSE,array_bytes_reverse[index]);
 index := index + 1;
UNTIL(index >= array_length)END_REPEAT;
```

按位与 (AND)

AND 指令利用 Source A 和 Source B 中的位执行按位与运算，并将结果放在 Destination 中。

要执行逻辑与运算，请参见第 325 页。

操作数：



梯形图

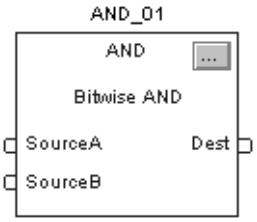
操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT**	立即数 标签	与 Source B 作与运算的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
源 B (Source B)	SINT INT **DINT**	立即数 标签	与 Source A 作与运算的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT**	标签	存储结果

```
dest := sourceA AND sourceB
```

结构化文本

在表达式中， AND 或符号 “ & ” 用作运算符。该表达式将评估 *sourceA* AND *sourceB* 的结果。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。



功能块

操作数	类型	格式	说明
AND 标签	FBD_LOGICAL	结构	AND 结构

FBD_LOGICAL 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	DINT	与 SourceB 作与运算的值。 任何整数都有效
SourceB	DINT	与 SourceA 作与运算的值。 任何整数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	DINT	指令的结果。将为该输出设置算术状态标志。

说明： 使能后，指令将执行与运算：

如果 Source A 中的位是	并且 Source B 中的位是	则 Destination 中的位是
0	0	0
0	1	0
1	0	0
1	1	1

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

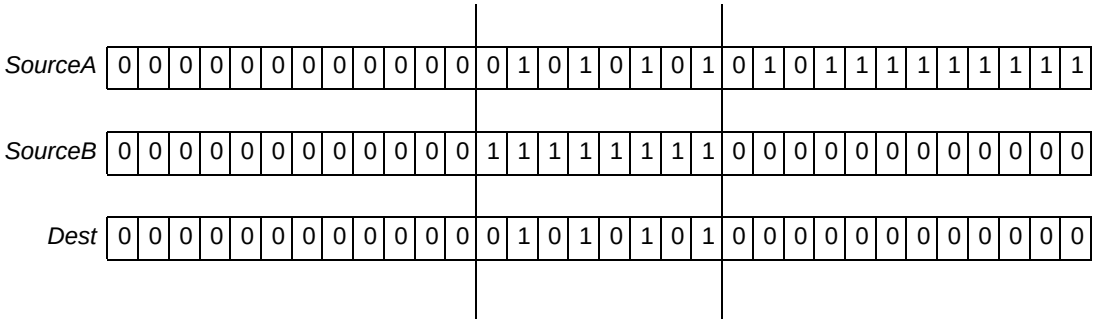
梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	该指令执行按位与运算。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

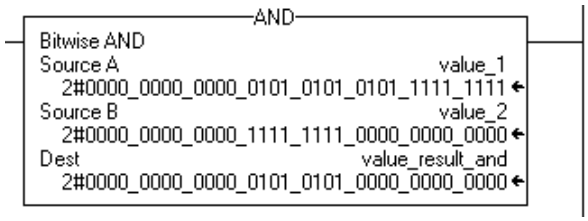
功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 使能后，AND 指令将对 SourceA 和 SourceB 执行按位与运算，并将结果放在 Dest 中。



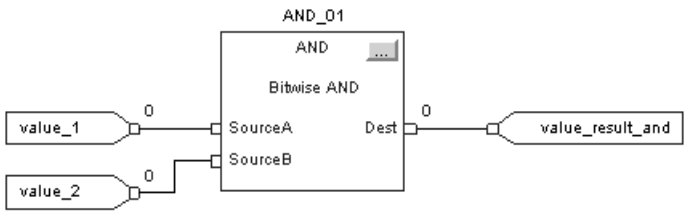
梯形图



结构化文本

```
value_result_and := value_1 AND value_2;
```

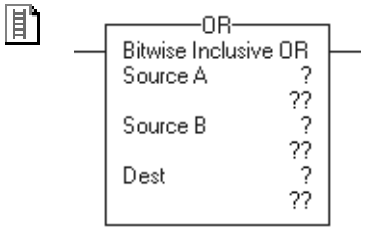
功能块



按位或 (OR)

OR 指令利用 Source A 和 Source B 中的位执行按位或运算，并将结果放在 Destination 中。

操作数：



梯形图

操作数	类型	格式	说明
源 A (Source A)	SINT INT **DINT**	立即数 标签	与 Source B 作或运算的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
源 B (Source B)	SINT INT **DINT**	立即数 标签	与 Source A 作或运算的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
目标 (Destination)	SINT INT **DINT**	标签	存储结果

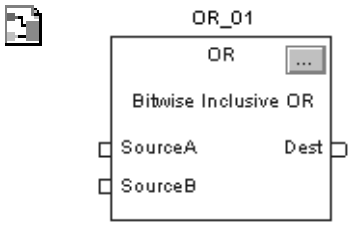
结构化文本

```
dest := sourceA OR sourceB
```

在表达式中， OR 用作运算符。该表达式将评估 *sourceA* OR *sourceB* 的结果。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式：	说明
OR 标签	FBD_LOGICAL	结构	OR 结构

FBD_LOGICAL 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	DINT	与 SourceB 作或运算的值。 任何整数都有效
SourceB	DINT	与 SourceA 作或运算的值。 任何整数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	DINT	指令的结果。将为该输出设置算术状态标志。

说明： 使能后，指令将执行或运算。

如果 Source A 中的位是	并且 Source B 中的位是	则 Destination 中的位是
0	0	0
0	1	1
1	0	1
1	1	1

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

算术状态标志 算术状态标志将受到影响。

错误条件： 无

执行：

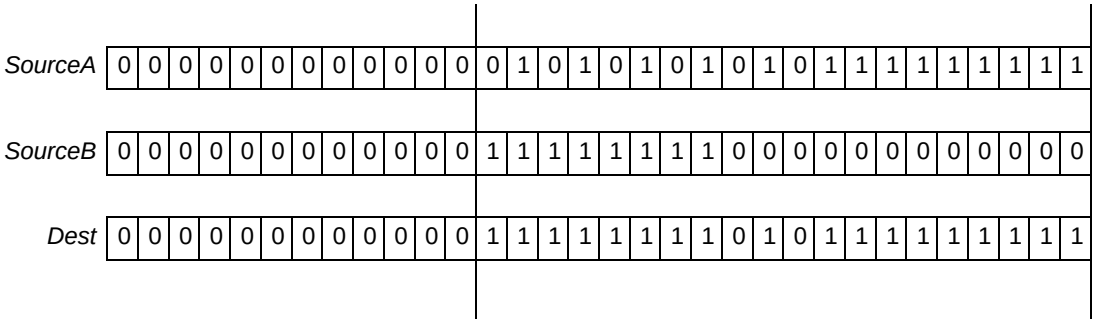
梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	该指令执行按位或运算。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

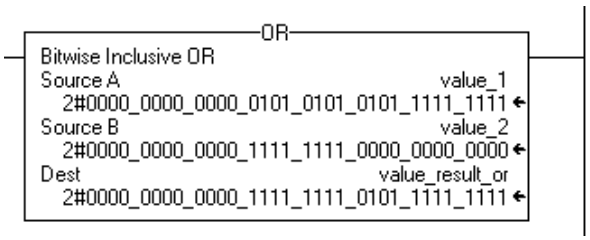
功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 使能后， OR 指令将对 SourceA 和 SourceB 执行按位或运算，并将结果放在 Dest 中。



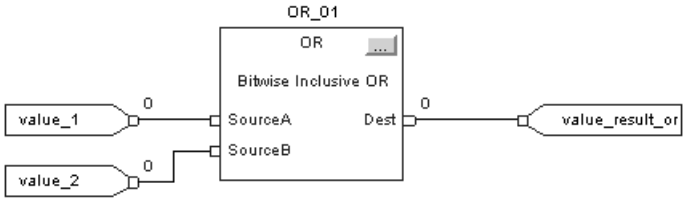
梯形图



结构化文本

```
value_result_or := value_1 OR value_2;
```

功能块


OR_01
OR
...
Bitwise Inclusive OR
value_1
0
SourceA
Dest
0
value_result_or
SourceB
value_2
0

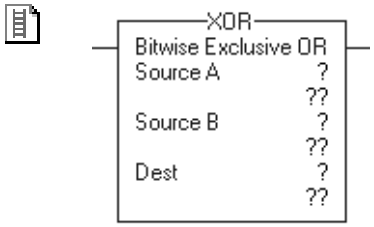
按位异或 (XOR)

XOR 指令利用 Source A 和 Source B 中的位执行按位异或运算，并将结果放在 Destination 中。

要执行逻辑异或运算，请参见第 331 页。

操作数：

梯形图



操作数	类型	格式	说明
源 A (Source A)	SINT INT DINT	立即数 标签	与 Source B 作异或运算的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
源 B (Source B)	SINT INT DINT	立即数 标签	与 Source A 作异或运算的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
目标 (Destination)	SINT INT DINT	标签	存储结果

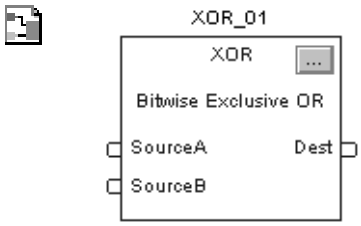
结构化文本

```
dest := sourceA XOR sourceB
```

在表达式中， XOR 用作运算符。该表达式将评估 *sourceA* XOR *sourceB* 的结果。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
XOR 标签	FBD_LOGICAL	结构	XOR 结构

FBD_LOGICAL 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
SourceA	DINT	与 SourceB 作异或运算的值。 任何整数都有效
SourceB	DINT	与 SourceA 作异或运算的值。 任何整数都有效
输出参数：	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	DINT	指令的结果。将为该输出设置算术状态标志。

说明： 使能后，指令将执行异或运算。

如果 Source A 中的位是	并且 Source B 中的位是	则 Destination 中的位是
0	0	0
0	1	1
1	0	1
1	1	0

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

算术状态标志 算术状态标志将受到影响。

错误条件： 无

执行：

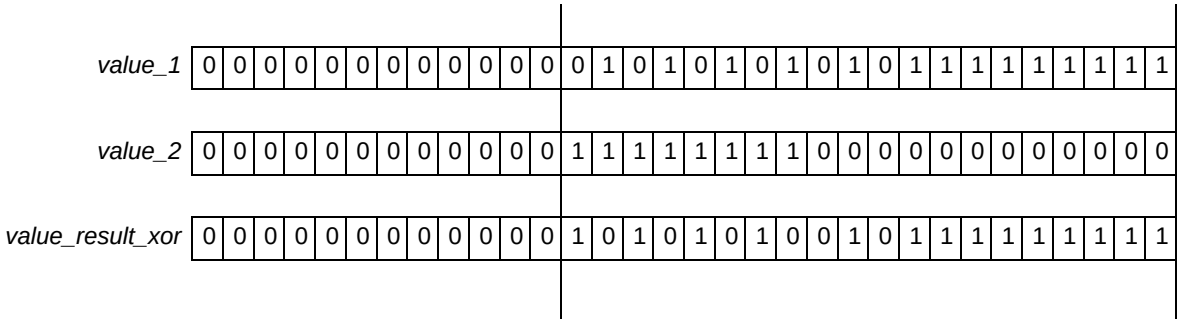
梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	该指令执行按位或运算。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

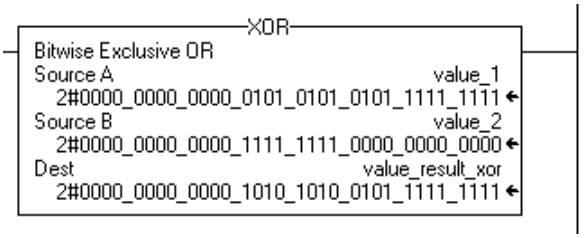
功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 使能后，XOR 指令将对 SourceA 和 SourceB 执行按位异或运算，并将结果放在目标标签中。



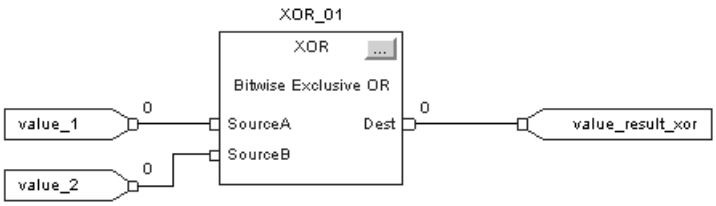
梯形图



结构化文本

```
value_result_xor := value_1 XOR value_2;
```

功能块


XOR_01
XOR
...
Bitwise Exclusive OR
value_1
0
SourceA
Dest
0
value_result_xor
SourceB
value_2
0

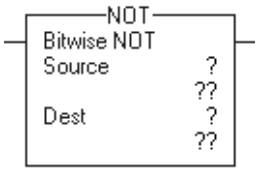
按位非 (NOT)

NOT 指令利用 Source 中的位执行按位非运算，并将结果放在 Destination 中。

要执行逻辑非运算，请参见第 334 页。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT	立即数 标签	要进行非运算的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
目标 (Destination)	SINT INT DINT	标签	存储结果

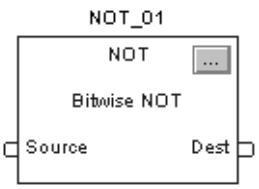
结构化文本

```
dest := NOT source
```

在表达式中，NOT 用作运算符。该表达式将评估 NOT *source* 的结果。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
NOT 标签	FBD_LOGICAL	结构	NOT 结构

FBD_LOGICAL 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态
Source	DINT	要进行非运算的值。 任何整数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
Dest	DINT	指令的结果。将为该输出设置算术状态标志。

说明： 使能后，指令将执行非运算。

如果 Source 中的位是	则 Destination 中的位是
0	1
1	0

如果混合使用整数数据类型，指令将使用 0 来填充较小整数数据类型的高位，使其与最大数据类型的尺寸相同。

算术状态标志： 算术状态标志将受到影响。

错误条件： 无

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	该指令执行按位非运算。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

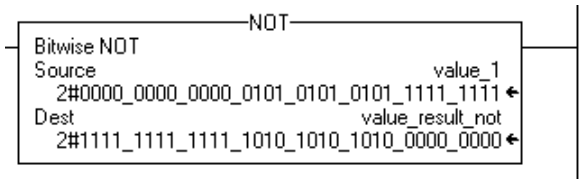
功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 使能后， NOT 指令将对 Source 执行按位非运算，并将结果放在 Dest 中。

value_1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
value_result_not	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0

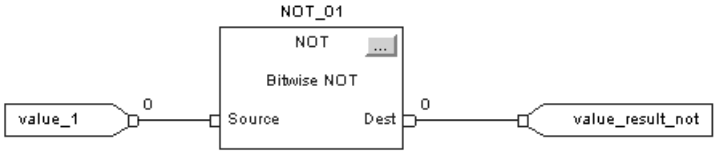
梯形图



结构化文本

```
value_result_not := NOT value_1;
```

功能块



布尔型与运算 (BAND)

BAND 指令可对八个布尔输入进行逻辑与运算。

要执行按位与运算，请参见第 311 页。

操作数：

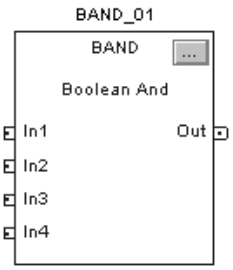
结构化文本

```
IF operandA AND operandB THEN
    <statement>;
END_IF;
```

在表达式中， AND 或符号 “ & ” 用作运算符。操作数必须是 BOOL 值或可赋值为 BOOL 值的表达式。该表达式评估 *operandA* 和 *operandB* 是否均已置位 (真)。

有关结构化文本中表达式的语法的信息，请参见附录 B。

功能块



操作数	类型	格式	说明
BAND 标签	FBD_BOOLEAN_AND	结构	BAND 结构

FBD_BOOLEAN_AND 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
In1	BOOL	第一个布尔输入。 默认为置位状态。
In2	BOOL	第二个布尔输入。 默认为置位状态。
In3	BOOL	第三个布尔输入。 默认为置位状态。
In4	BOOL	第四个布尔输入。 默认为置位状态。
In5	BOOL	第五个布尔输入。 默认为置位状态。
In6	BOOL	第六个布尔输入。 默认为置位状态。

输入参数	数据类型	说明
In7	BOOL	第七个布尔输入。 默认为置位状态。
In8	BOOL	第八个布尔输入。 默认为置位状态。
输出参数	数据类型	说明
EnableOut	BOOL	使能输出。
Out	BOOL	指令的输出。

说明： BAND 指令可对八个布尔输入进行与运算。如果未使用输入，则默认为置位 (1)。

Out = In1 AND In2 AND In3 AND In4 AND In5 AND In6 AND In7 AND In8

算术状态标志： 不受影响

错误条件： 无

执行：

条件	功能块操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

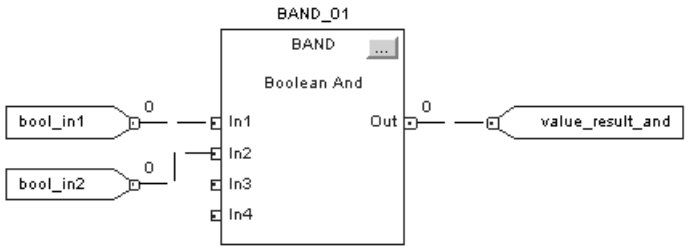
示例 **1**： 在本示例中，对 *bool_in1* 和 *bool_in2* 进行与运算，并将结果放在 *value_result_and* 中。

如果 BOOL_IN1 是	如果 BOOL_IN2 是	则 VALUE_RESULT_AND 是
0	0	0
0	1	0
1	0	0
1	1	1

结构化文本

```
value_result_and := bool_in1 AND bool_in2;
```

功能块



示例 **2**： 如果 *bool_in1* 和 *bool_in2* 都已置位 (真)，则 *light1* 置位 (开启)。否则， *light1* 将被清零 (关闭)。

结构化文本

```
IF bool_in1 AND bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

布尔型或运算 (BOR)

BOR 指令可对八个布尔输入进行逻辑或运算。

要执行按位或运算，请参见第 314 页。

操作数：

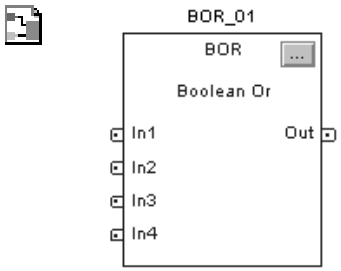
结构化文本

```
IF operandA OR operandB THEN
    <statement>;
END_IF;
```

在表达式中， OR 用作运算符。操作数必须是 BOOL 值或可赋值为 BOOL 值的表达式。该表达式评估 *operandA* 或 *operandB* 或两者是否已置位 (真)。

有关结构化文本中表达式的语法的信息，请参见附录 B。

功能块



操作数	类型	格式	说明
BOR 标签	FBD_BOOLEAN_OR	结构	BOR 结构

FBD_BOOLEAN_OR 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
In1	BOOL	第一个布尔输入。 默认为清零状态。
In2	BOOL	第二个布尔输入。 默认为清零状态。
In3	BOOL	第三个布尔输入。 默认为清零状态。
In4	BOOL	第四个布尔输入。 默认为清零状态。
In5	BOOL	第五个布尔输入。 默认为清零状态。
In6	BOOL	第六个布尔输入。 默认为清零状态。
In7	BOOL	第七个布尔输入。 默认为清零状态。

输入参数	数据类型	说明
In8	BOOL	第八个布尔输入。 默认为清零状态。
输出参数	数据类型	说明
EnableOut	BOOL	使能输出。
Out	BOOL	指令的输出。

说明： BOR 指令可对八个布尔输入进行或运算。如果未使用输入，则默认为清零 (0)。

Out = In1 OR In2 OR In3 OR In4 OR In5 OR In6 OR In7 OR In8

算术状态标志： 不受影响

错误条件： 无

执行：

条件	功能块操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

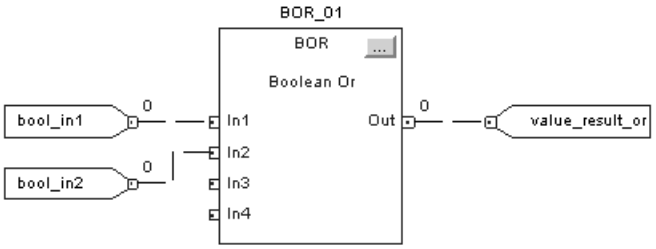
示例 **1**： 在本示例中，对 *bool_in1* 和 *bool_in2* 进行或运算，并将结果放在 *value_result_or* 中。

如果 BOOL_IN1 是	如果 BOOL_IN2 是	则 VALUE_RESULT_OR 是
0	0	0
0	1	1
1	0	1
1	1	1

结构化文本

```
value_result_or := bool_in1 OR bool_in2;
```

功能块



示例 **2**： 在本示例中，如果满足以下条件， *light1* 将被置位 (开启)：

- 仅 *bool_in1* 置位 (真)。
- 仅 *bool_in2* 置位 (真)。
- *bool_in1* 和 *bool_in2* 均已置位 (真)。

否则， *light1* 将被清零 (关闭)。

结构化文本

```
IF bool_in1 OR bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

布尔异或 (BXOR)

BXOR 用于对两个布尔输入进行异或运算。

要执行按位异或运算，请参见第 318 页。

操作数：

结构化文本

```
IF operandA XOR operandB THEN
    <statement>;
END_IF;
```

在表达式中， XOR 用作运算符。操作数必须是 BOOL 值或可赋值为 BOOL 值的表达式。该表达式评估是否仅 *operandA* 或仅 *operandB* 置位 (真)。

有关结构化文本中表达式的语法的信息，请参见附录 B。

功能块

操作数	类型	格式	说明
BXOR 标签	FBD_BOOLEAN_XOR	结构	BXOR 结构

FBD_BOOLEAN_XOR 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
In1	BOOL	第一个布尔输入。 默认为清零状态。
In2	BOOL	第二个布尔输入。 默认为清零状态。
输出参数	数据类型	说明
EnableOut	BOOL	使能输出。
Out	BOOL	指令的输出。

说明： BXOR 指令用于对两个布尔输入进行异或运算。

Out = In1 XOR In2

算术状态标志： 不受影响

错误条件： 无

执行：

条件	功能块操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

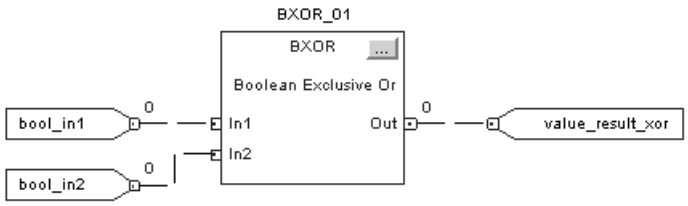
示例 **1**： 在本示例中，对 *bool_in1* 和 *bool_in2* 进行异或运算，并将结果放在 *value_result_xor* 中。

如果 BOOL_IN1 是	如果 BOOL_IN2 是	则 VVALUE_RESULT_XOR 是
0	0	0
0	1	1
1	0	1
1	1	0

结构化文本

```
value_result_xor := bool_in1 XOR bool_in2;
```

功能块



示例 **2**： 在本示例中，如果满足以下条件， *light1* 将被置位 (开启)：

- 仅 *bool_in1* 置位 (真)。
- 仅 *bool_in2* 置位 (真)。

否则， *light1* 将被清零 (关闭)。

结构化文本

```
IF bool_in1 XOR bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

布尔型非运算 (BNOT)

BNOT 指令用于对布尔输入取反。

要执行按位非运算，请参见第 322 页。

操作数：

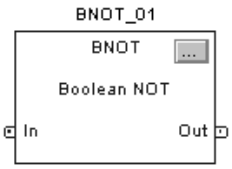
结构化文本

```
IF NOT operand THEN
    <statement>;
END_IF;
```

在表达式中， NOT 用作运算符。操作数必须是 BOOL 值或可赋值为 BOOL 值的表达式。该表达式评估 *operand* 是否被清零 (假)。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
BNOT 标签	FBD_BOOLEAN_NOT	结构	BNOT 结构

FBD_BOOLEAN_NOT 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
In	BOOL	指令的输入。 默认为置位状态。
输出参数	数据类型	说明：
EnableOut	BOOL	使能输出。
Out	BOOL	指令的输出。

说明：　BNOT 指令用于对布尔输入取反。

Out = NOT In

算术状态标志：　不受影响

错误条件：　无

执行：

条件	功能块操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

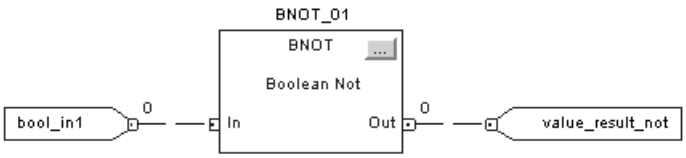
示例 **1**： 在本示例中，对 *bool_in1* 进行取反运算，并将结果放在 *value_result_not* 中。

如果 BOOL_IN1 是	则 VALUE_RESULT_NOT 是
0	1
1	0

结构化文本

```
value_result_not := NOT bool_in1;
```

功能块



示例 **2**： 如果 *bool_in1* 被清零，则 *light1* 也将被清零 (关闭)。否则，*light1* 将被置位 (开启)。

结构化文本

```
IF NOT bool_in1 THEN
    light1 := 0;
ELSE
    light1 := 1;
END_IF;
```

注：

数组 (文件)/ 综合指令
(FAL、 FSC、 COP、 CPS、 FLL、 AVE、 SRT、 STD、 SIZE)

简介

文件 / 综合指令用于对数据数组进行运算。

如果要	使用以下指令	在以下语言中可用	页码
对数组中的值执行算术、逻辑、移位和函数运算	FAL	梯形图 结构化文本(1)	343
搜索和比较数组中的值	FSC	梯形图	354
将一个数组中的内容复制到另一个数组	COP	梯形图 结构化文本	363
不中断地将一个数组中的内容复制到另一个数组	CPS	梯形图 结构化文本	363
用特定数据填充数组	FLL	梯形图 结构化文本 (1)	369
计算一个数组中所有值的平均值	AVE	梯形图 结构化文本 (1)	373
将一个维度的数组数据按升序排列	SRT	梯形图 结构化文本	378
计算一个数组中所有值的标准偏差	STD	梯形图 结构化文本 (1)	383
获取数组某一维度的尺寸	SIZE	梯形图 结构化文本	389

(1) 没有对等的结构化文本指令。使用其它结构化文本编程可获得相同结果。请参见指令说明。

可以混合使用不同数据类型，但可能会造成精度降低和舍入错误，而且指令的执行时间也会变长。请检查 S:V 位，查看结果是否被截断。

对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

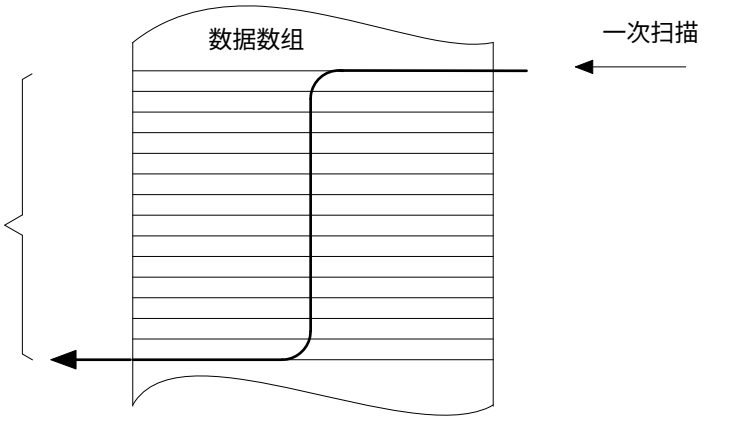
选择操作模式

对于 FAL 和 FSC 指令，通过模式来决定控制器如何分配数组操作。

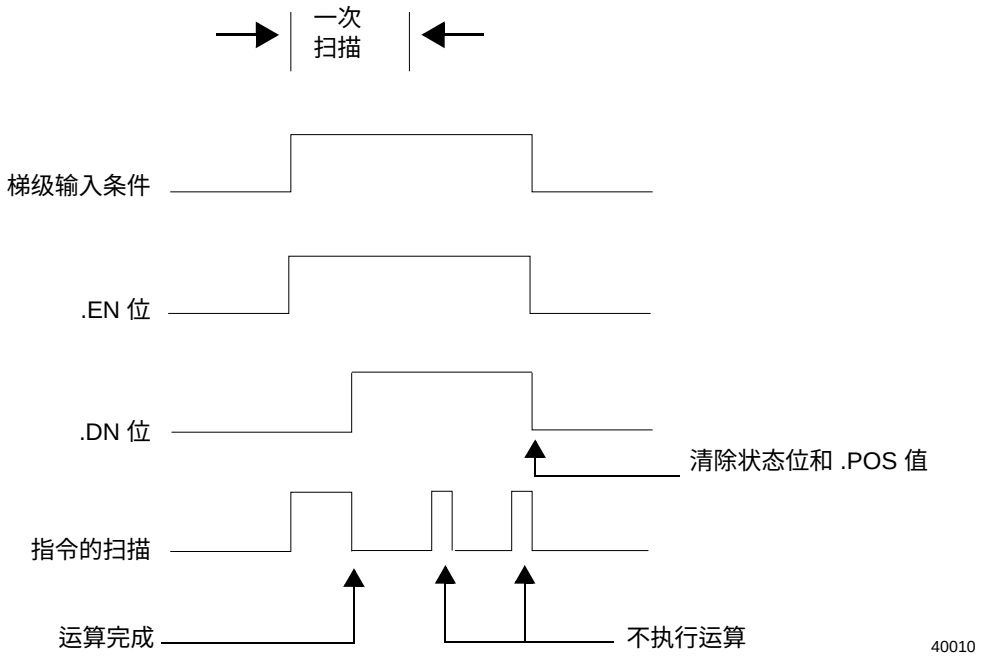
要执行的操作	选择的模式
对数组中所有指定的元素进行运算，然后继续执行下一条指令	全部
将数组运算分配到若干次扫描中 键入每次扫描要运算的元素个 (1-2147483647)	数目
每次梯级输入条件从假变为真时，对数组的一个元素进行处理	增量

“ 所有 ” 模式

在 “ 所有 ” 模式中，将对数组中所有指定的元素进行操作，然后再继续执行下一条指令。当指令的梯级输入条件从假变为真时，开始进行操作。控制结构中的位置 (.POS) 值指向指令当前使用的数组元素。当 .POS 值等于 .LEN 值时，操作停止。



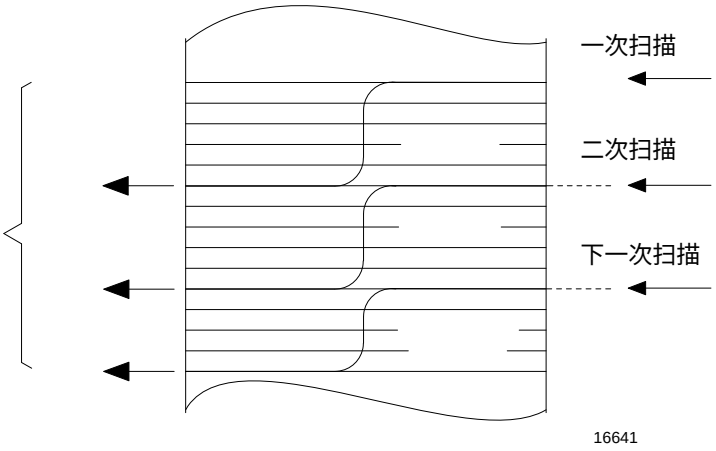
下面的时间表说明了状态位与指令操作之间的关系。当指令执行完成时，.DN 位置位。梯级输入条件为假时，.DN 位、.EN 位和 .POS 值清零。只有这样，梯级输入条件从假到真的转换才能再次触发指令执行。



数量模式

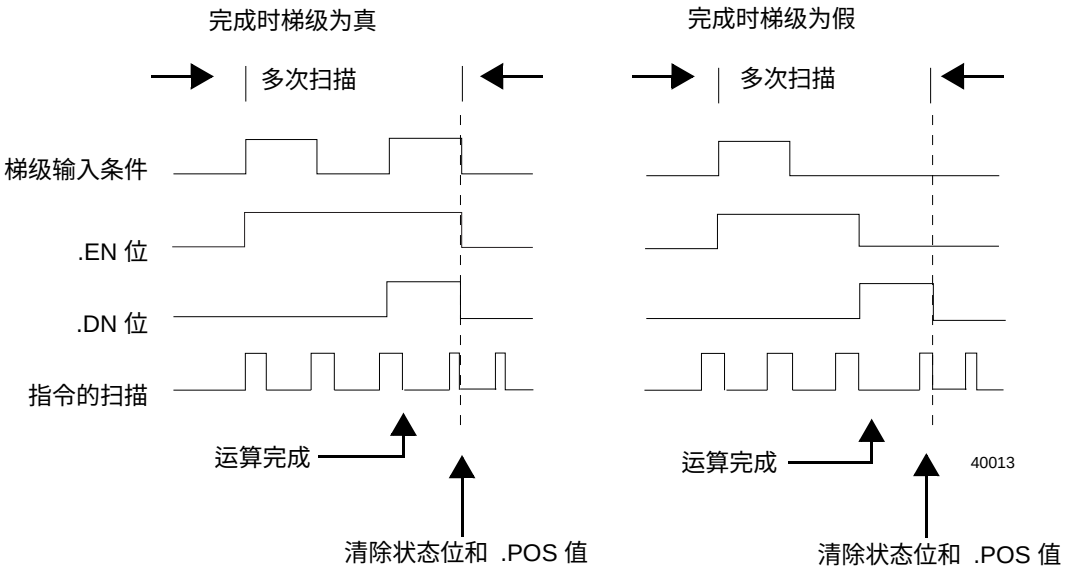
数量模式将数组运算分配到若干次扫描中。处理非时间性数据或大量数据时，此模式非常有用。键入每次扫描要运算的元素个数，这样可缩短扫描时间。

当梯级输入条件从假变为真时，将触发执行。指令将在每次扫描时执行，扫描次数为完成整个数组运算所需的次数。一旦触发，梯级输入条件即使反复改变，也不会中断指令的执行。



重要事项　在 .DN 位置位之前，应避免使用数量模式下数组指令操作的结果。

下面的时间表说明了状态位与指令操作之间的关系。当指令执行完成时， .DN 位置位。

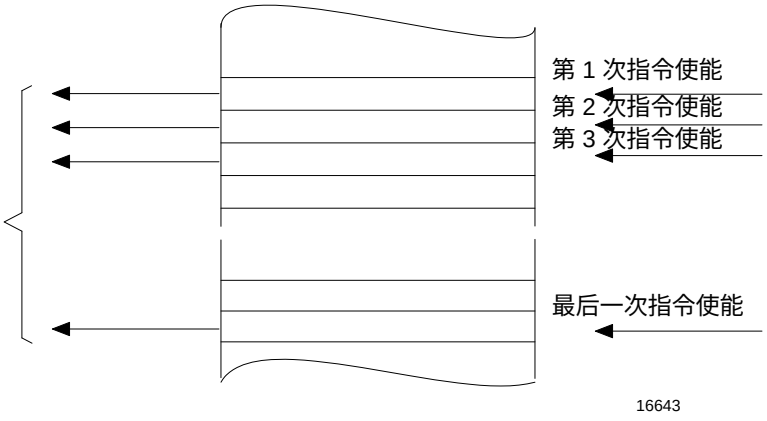


如果完成时梯级输入条件为真，.EN 位和 .DN 位将处于置位状态，直到梯级输入条件变为假。当梯级输入条件变为假时，这些状态位和 .POS 值将被清零。

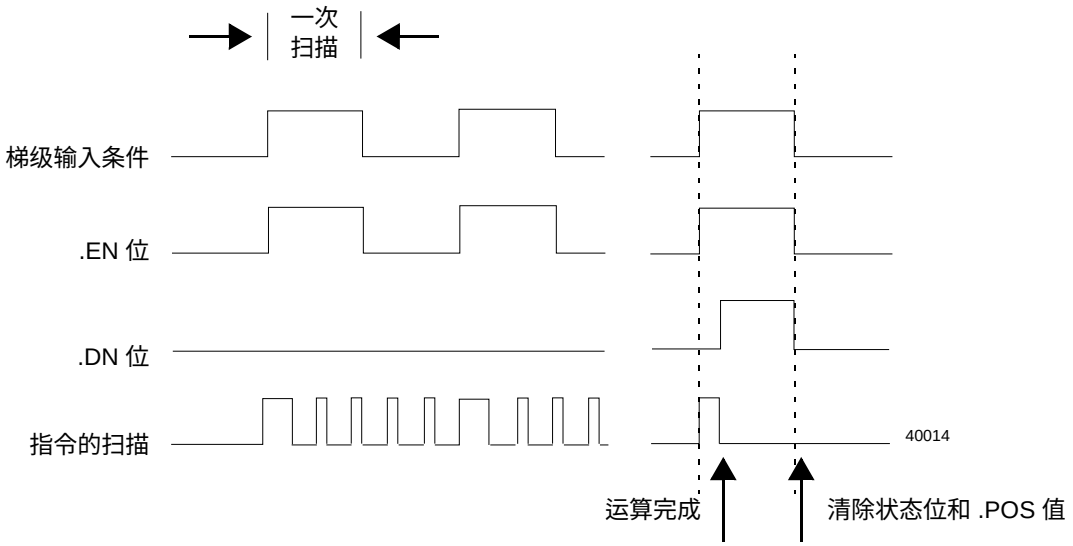
如果完成时梯级输入条件为假，.EN 位将立即清零。在 .EN 位清零后进行一次扫描，.DN 位和 .POS 值将被清零。

增量模式

在增量模式下，每次梯级输入条件从假变为真时，将对数组的一个元素进行处理。



下面的时间表说明了状态位与指令操作之间的关系。只有在梯级输入条件从假变为真的扫描中才会执行。每次执行时，只对数组的一个元素进行处理。如果梯级输入条件在多次扫描期间保持为真，则指令只在第一次扫描期间执行。



当梯级输入条件为真时， .EN 位置位。处理完数组中的最后一个元素后， .DN 位将置位。处理完最后一个元素并且梯级输入条件变为假后， .EN 位、 .DN 位和 .POS 值将清零。

当数量模式每次扫描处理一个元素时，其与增量模式的不同之处在于：

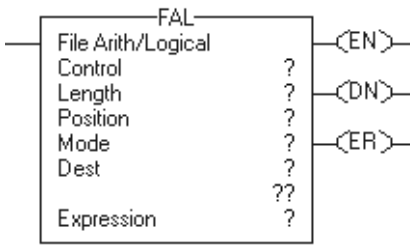
- 数量模式只需梯级输入条件从假到真跳变一次即开始执行，且每次扫描可处理任意个数的元素。指令在每次扫描都将继续处理指定个数的元素，直至完成，而与梯级输入条件的状态无关。
- 增量模式需要梯级输入条件从假变为真才能处理数组中的一个元素。

文件算术逻辑 (FAL)

FAL 指令对数组内存储的数据进行复制、算术、逻辑及函数运算操作。

操作数：

梯形图



操作数	类型	格式	说明
Control	CONTROL	标签	操作的控制结构
Length	DINT	立即数	数组中要处理的元素个数
Position	DINT	立即数	数组中的当前元素 初始值通常为 0
Mode	DINT	立即数	如何分配操作 选择 INC、 ALL 或键入一个数字
Destination	SINT INT **DINT** **REAL**	标签	要存储结果的标签
Expression	SINT INT **DINT** **REAL**	立即数 标签	由标签和／或立即值 (被运算符隔开) 组成的表达式
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		

结构化文本

结构化文本没有 FAL 指令，但可以通过 SIZE 指令和 FOR...DO 或其它循环结构来实现相同的结果。

```
SIZE(destination,0,length-1);
FOR position = 0 TO length DO
    destination[position] := numeric_expression;
END_FOR;
```

有关结构化文本中结构语法的信息，请参见结构化文本编程。

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 FAL 指令是否使能。
.DN	BOOL	指令对最后一个元素进行运算后 (.POS = .LEN)，完成位被置位。
.ER	BOOL	如果表达式发生溢出 (S:V 置位)，错误位置位。指令停止执行直到程序清零 .ER 位。 .POS 值包含产生溢出的元素的位置。
.LEN	DINT	长度用于指定 FAL 指令操作的数组元素数目。
.POS	DINT	位置包含指令正在访问的当前元素的位置。

说明： FAL 指令对数组执行的运算与 CPT 指令对元素执行的运算相同。

从第 350 页开始的示例显示了如何使用 .POS 值对数组执行逐步操作。 如果 Destination 表达式中的下标超出范围， FAL 指令将产生一个主要故障 (类型 4，代码 20)。

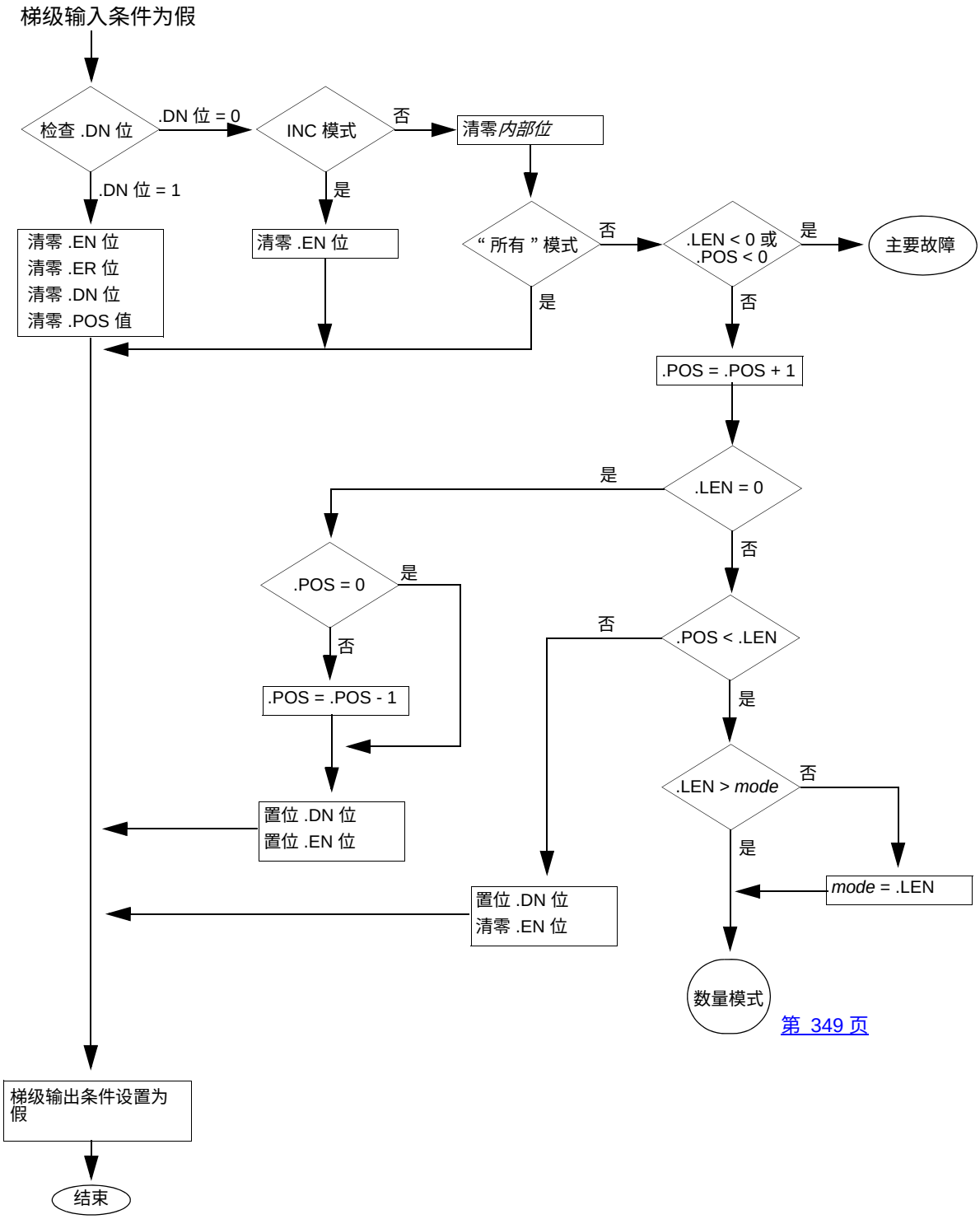
算术状态标志： 算术状态标志将受到影响。

故障条件：

出现主要故障的条件	故障类型	故障代码
下标超出范围	4	20
.POS < 0 或 .LEN < 0	4	21

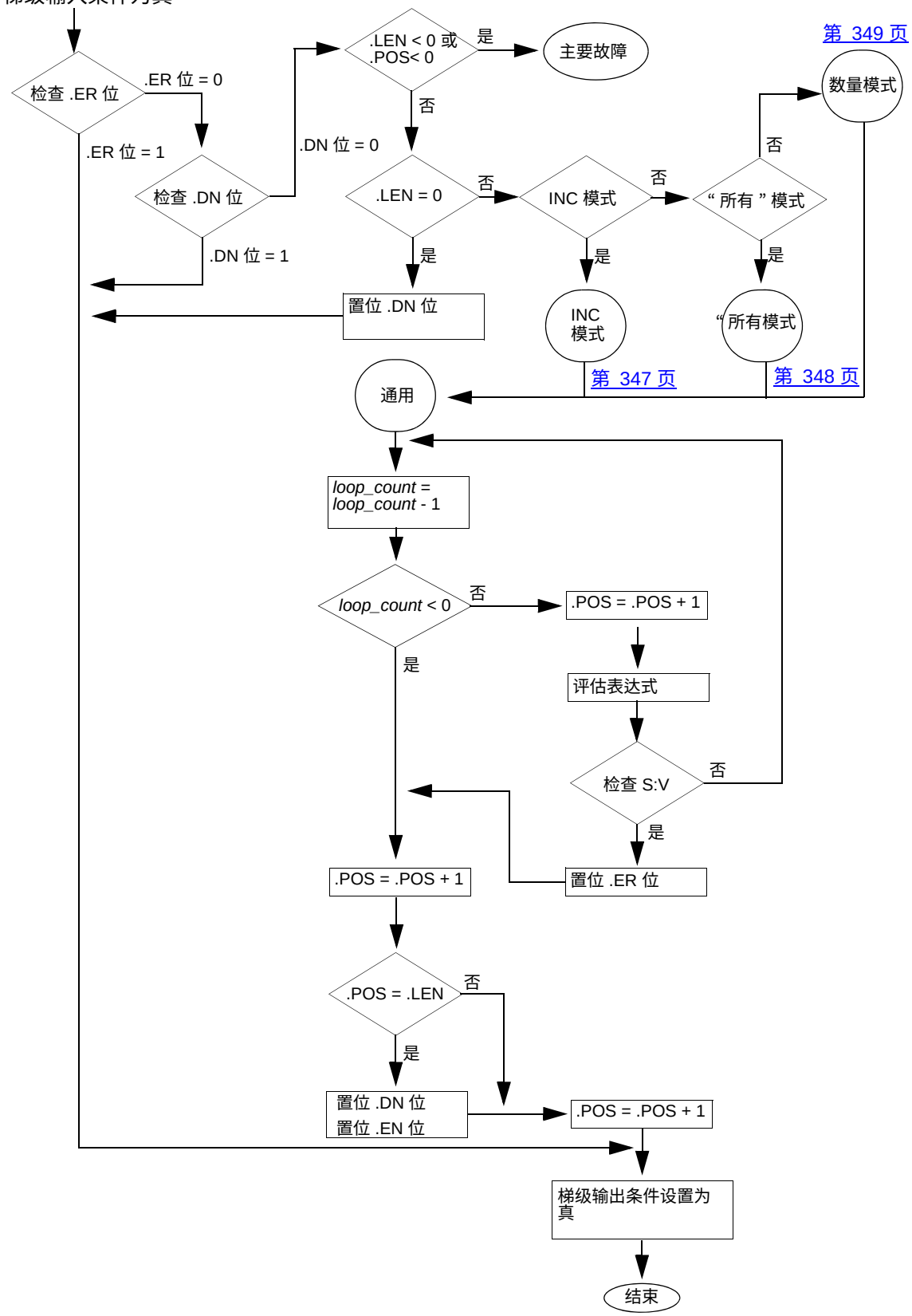
执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。

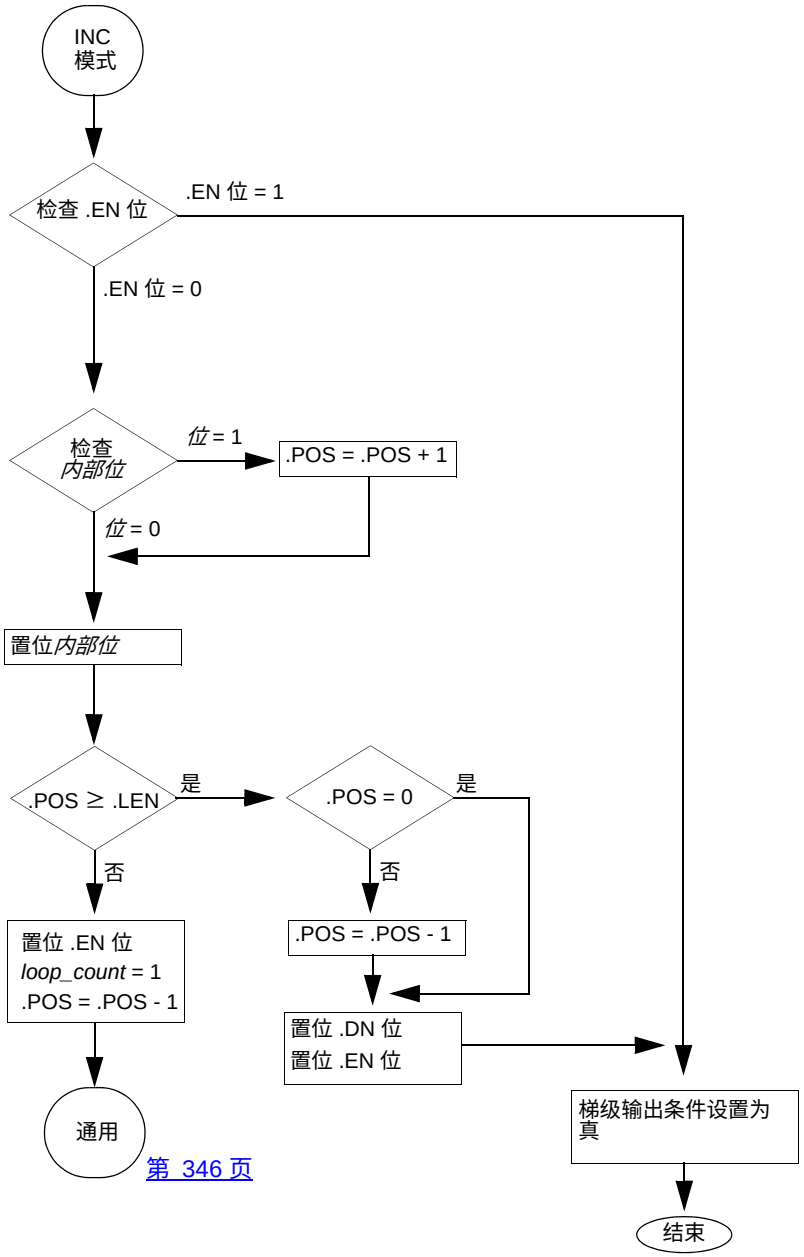


条件	梯形图操作

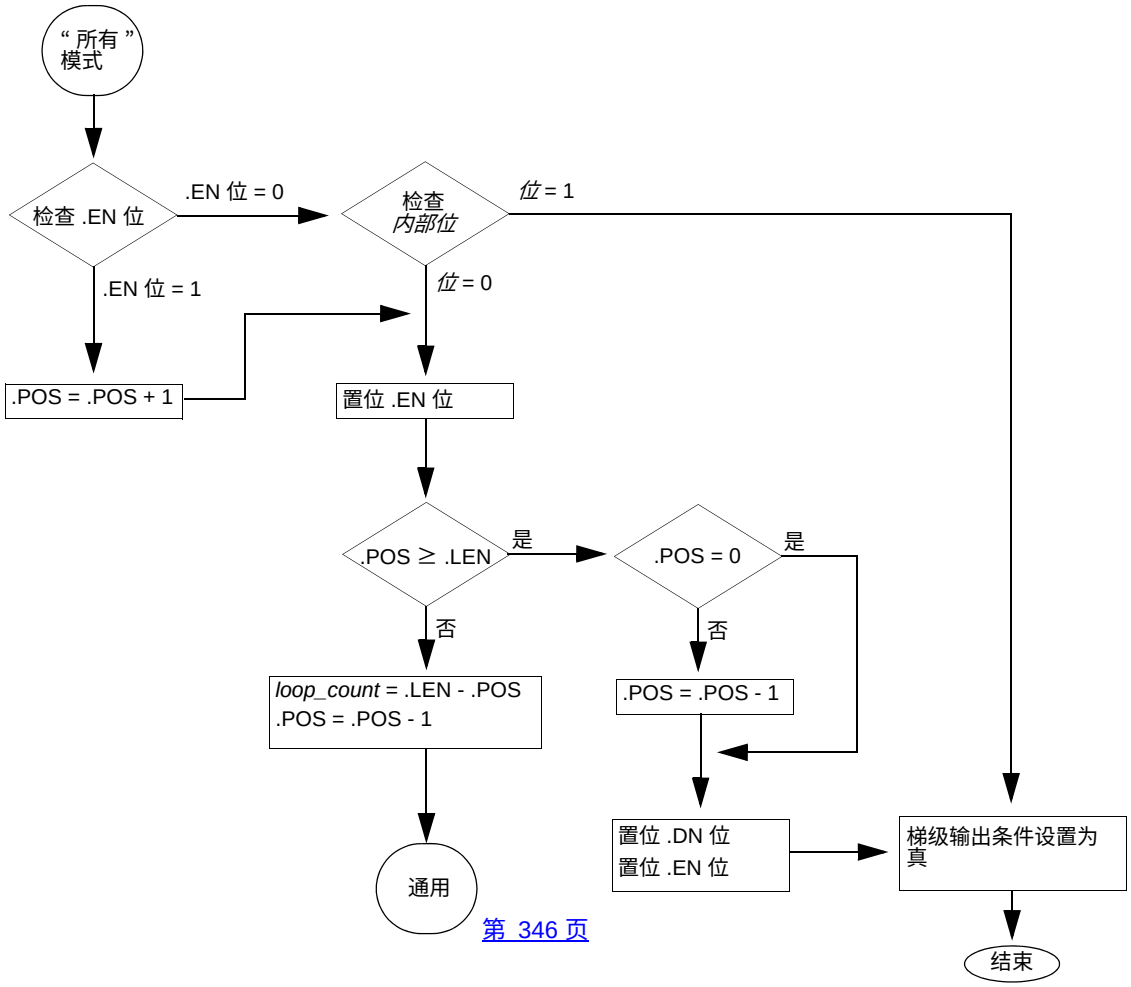
梯级输入条件为真



条件	梯形图操作

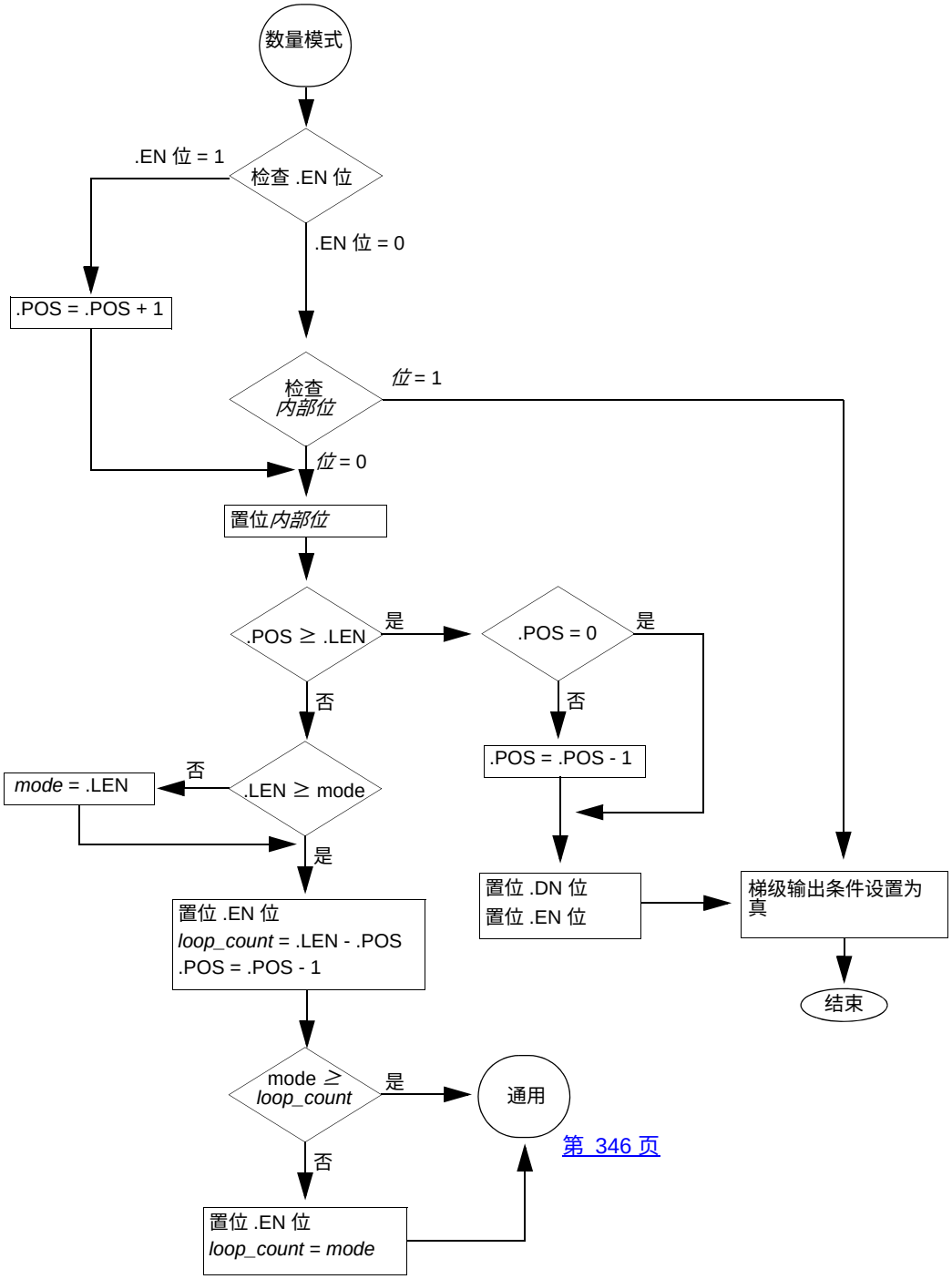

INC
模式
检查 .EN 位
.EN 位 = 1
.EN 位 = 0
检查
内部位
位 = 1
.POS = .POS + 1
位 = 0
置位内部位
.POS ≥ .LEN
是
.POS = 0
是
否
否
置位 .EN 位
loop_count = 1
.POS = .POS - 1
.POS = .POS - 1
置位 .DN 位
置位 .EN 位
通用
第 346 页
梯级输出条件设置为真
结束

条件	梯形图操作



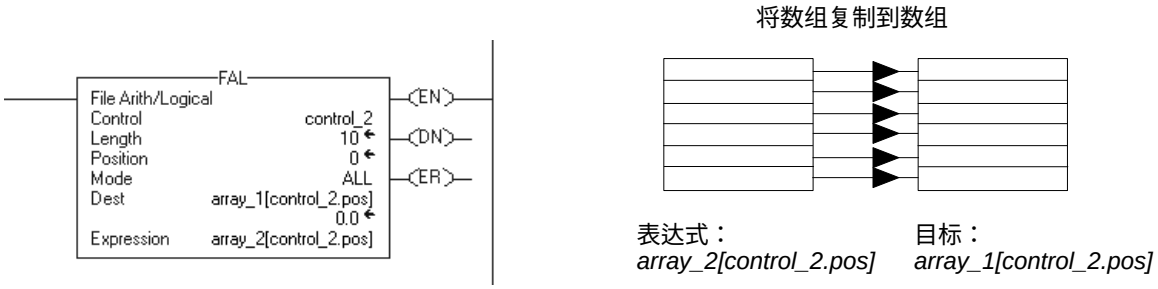
第 346 页

条件	梯形图操作

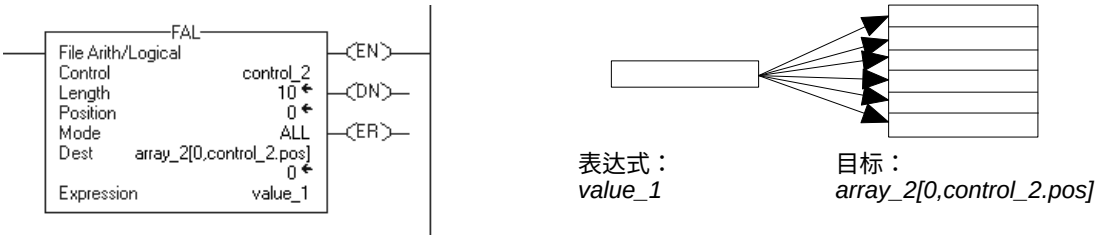


后扫描	梯级输出条件设置为假。

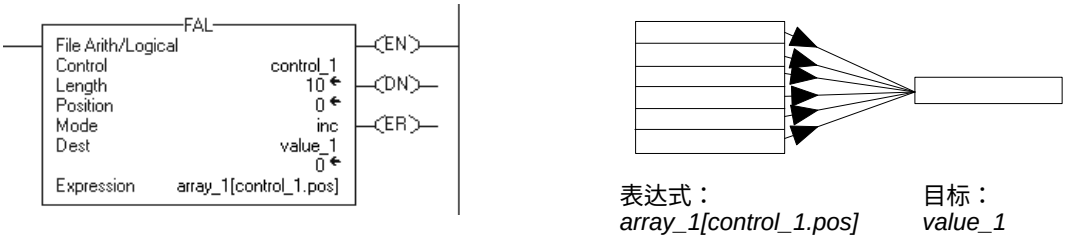
示例 **1**： 使能后， FAL 指令将 *array_2* 中的每个元素复制到 *array_1* 中的相同位置。



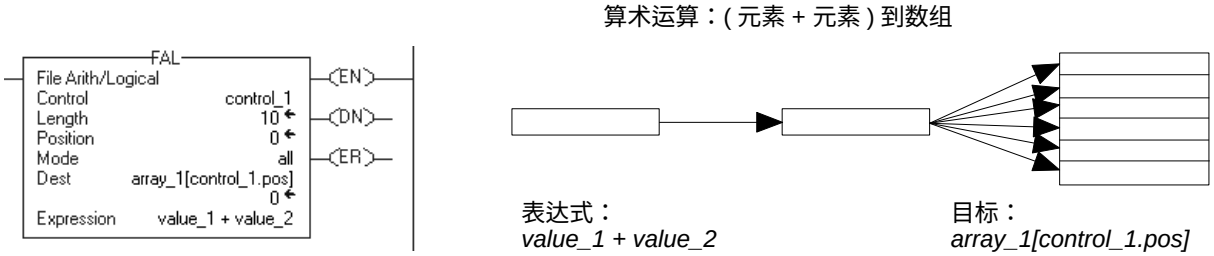
示例 **2**： 使能后， FAL 指令将 *value_1* 复制到 *array_2* 第二维的前 10 个位置中。



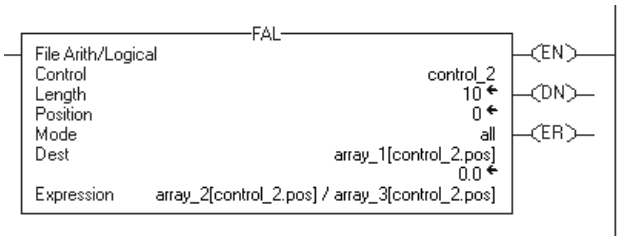
示例 **3**： 每当使能 FAL 指令时，它会 *array_1* 的当前值复制到 *value_1*。 FAL 指令使用增量模式，因此每次使能指令时仅复制一个数组值。下次使能指令时，指令将 *value_1* 覆盖为 *array_1* 中的下一个值。

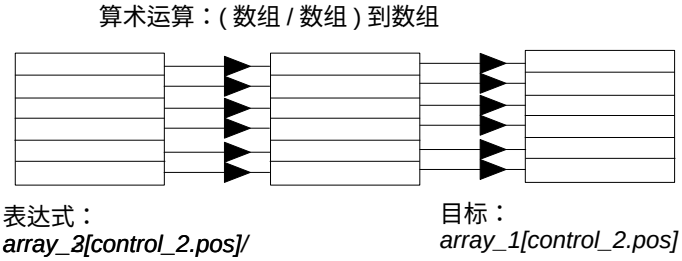


示例 **4**： 使能后， FAL 指令将 *value_1* 和 *value_2* 相加，并将结果存储到 *array_1* 的当前位置中。

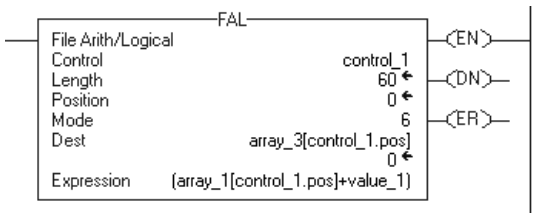


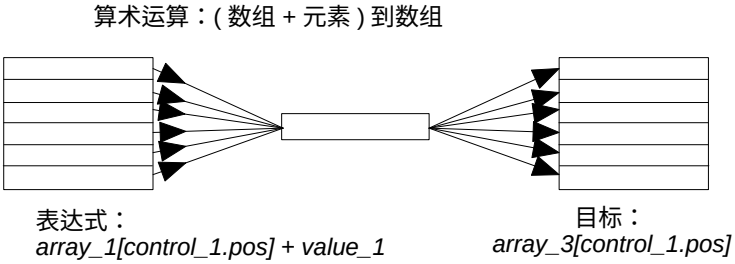
示例 **5**： 使能后， FAL 指令将 *array_2* 当前位置的值除以 *array_3* 当前位置的值，并将结果存储到 *array_1* 的当前位置中。



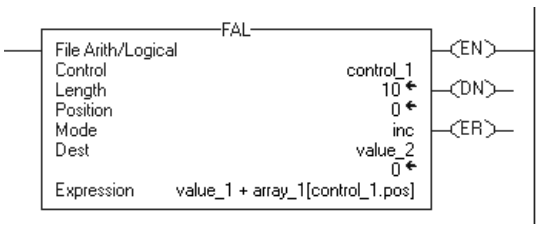


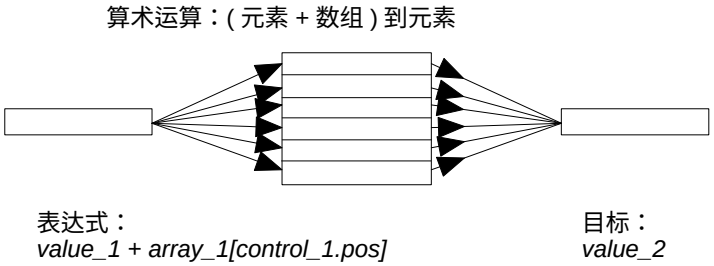
示例 **6**： 使能后， FAL 指令将 *array_1* 当前位置的值与 *value_1* 相加，并将结果存储到 *array_3* 的当前位置中。必须对要处理的 *array_1* 和 *array_3* 完全执行 10 次该指令。



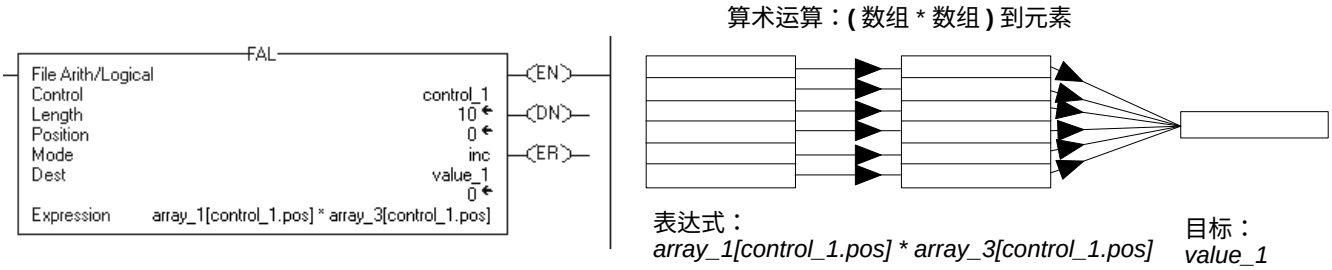


示例 **7**： 每次使能 FAL 指令时，它都会将 *value_1* 和 *array_1* 的当前值相加，并将结果存储到 *value_2* 中。 FAL 指令使用增量模式，因此每次使能指令时只有一个数组值与 *value_1* 相加。下次使能指令时，指令将覆盖 *value_2*。





示例 **8**： 使能后， FAL 指令将 *array_1* 的当前值和 *array_3* 的当前值相乘，并将结果存储到 *value_1* 中。 FAL 指令使用增量模式，因此每次使能指令时仅相乘一对数组值。下次使能指令时，指令将覆盖 *value_1*。



FAL 表达式

FAL 指令中表达式的编写方法与 CPT 指令中表达式的编写方法相同。请查看以下部分，了解以上两种指令通用的有效运算符、格式和运算顺序。

有效运算符

运算符	说明	最佳类型
+	加	DINT、 REAL
-	减 / 取反	DINT、 REAL
*	乘	DINT、 REAL
/	除	DINT、 REAL
**	指数 (x 的 y 次幂)	DINT、 REAL
ABS	绝对值	DINT、 REAL
ACS	反余弦	REAL
AND	按位与	DINT
ASN	反正弦	REAL
ATN	反正切	REAL
COS	余弦	REAL
DEG	弧度转角度	DINT、 REAL
FRD	BCD 转整数	DINT

运算符	说明	最佳类型
LN	自然对数	REAL
LOG	以 10 为底的对数	REAL
MOD	模数除法	DINT、 REAL
NOT	按位求补	DINT
OR	按位或	DINT
RAD	角度转弧度	DINT、 REAL
SIN	正弦	REAL
SQR	平方根	DINT、 REAL
TAN	正切	REAL
TOD	整数转 BCD	DINT
TRN	截断	DINT、 REAL
XOR	按位异或	DINT

格式表达式

对于表达式中使用的每一个运算符，都需要提供一个或两个操作数 (标签或立即值)。有关表达式中运算符和操作数的格式，请遵循下表中的规定。

运算符涉及的操作数的个数	使用的格式	示例
一个操作数	运算符 (操作数)	ABS(*tag_a*)
两个操作数	操作数 _a 运算符 操作数 _b	• *tag_b* + 5 • *tag_c* AND *tag_d* • (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

确定运算顺序

写入表达式中的运算将由指令按照规定的顺序执行，不一定按照写入的顺序执行。通过用圆括号括起一些内容，可以更改运算顺序，这样指令就会先执行括号内的运算，后执行其它运算。

相同顺序的运算从左到右执行。

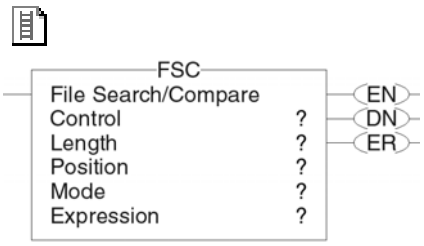
顺序	运算
1.	( )
2.	ABS、 ACS、 ASN、 ATN、 COS、 DEG、 FRD、 LN、 LOG、 RAD、 SIN、 SQR、 TAN、 TOD、 TRN
3.	**
4.	– (取反)， NOT
5.	*、 /、 MOD
6.	– (减)， +
7.	AND
8.	XOR
9.	OR

文件搜索和比较 **(FSC)**

FSC 指令对数组中的元素逐个进行比较。

操作数：

梯形图



操作数	类型	格式	说明
Control	CONTROL	标签	操作的控制结构
Length	DINT	立即数	数组中要处理的元素个数
Position	DINT	立即数	数组中的偏移量 初始值通常为 0

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 FSC 指令是否使能。
.DN	BOOL	指令对最后一个元素进行运算后 (.POS = .LEN)， 完成位被置位。
.ER	BOOL	不修改错误位。
.IN	BOOL	禁止位指示 FSC 指令检测到结果为真的比较。必须将此位清零才能继续搜索操作。
.FD	BOOL	发现位指示 FSC 指令检测到结果为真的比较。
.LEN	DINT	长度用于指定指令操作的数组元素数目。
.POS	DINT	位置包含指令正在访问的当前元素的位置。

说明：　当使能 FSC 指令且比较结果为真时，该指令将置位 .FD 位，.POS 位将反映该指令找到的结果为真的比较所在的数组位置。该指令将置位 .IN 位以阻止继续搜索。

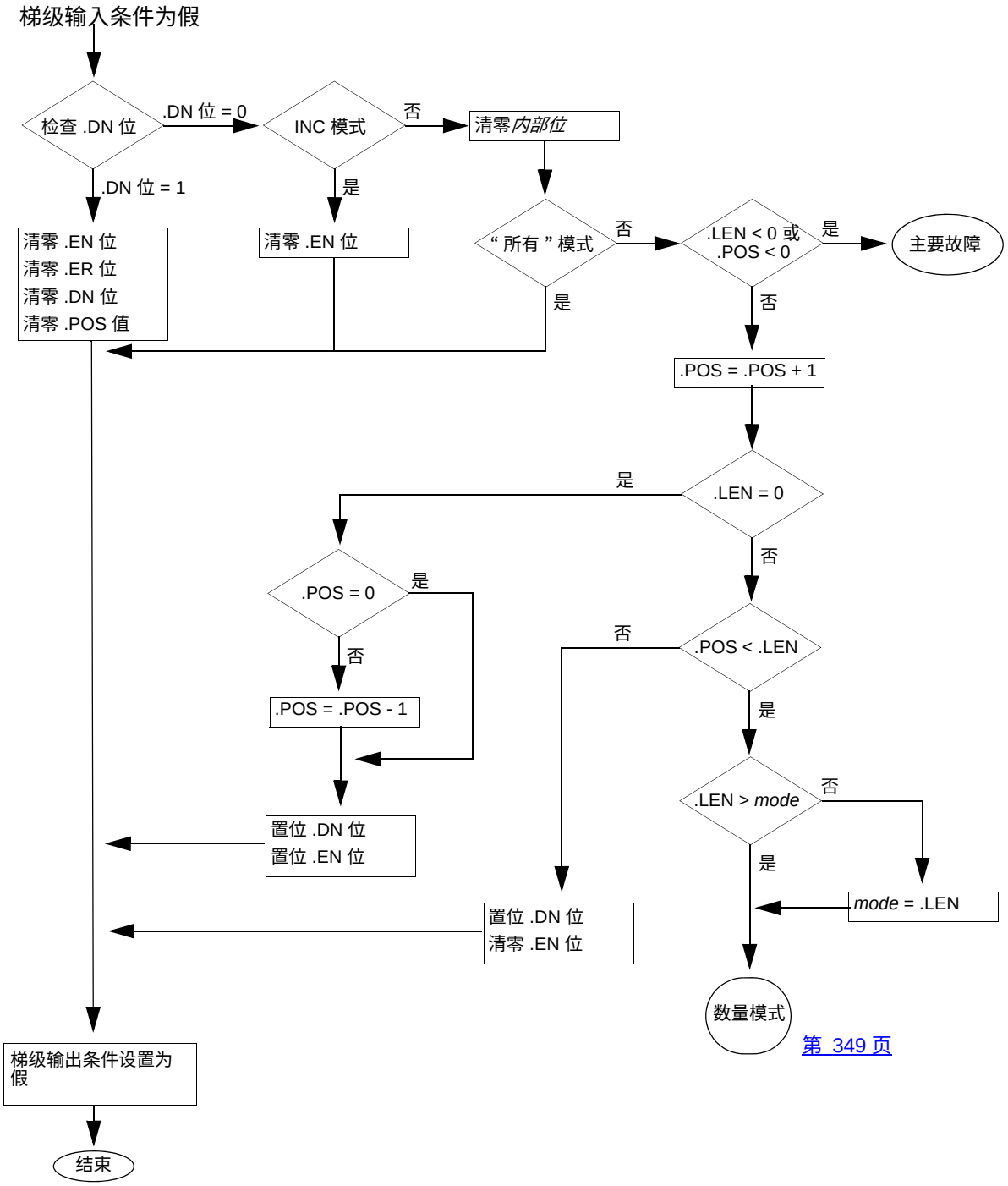
算术状态标志：　算术状态标志将受到影响。

故障条件：

出现主要故障的条件	故障类型	故障代码
.POS < 0 或 .LEN < 0	4	21

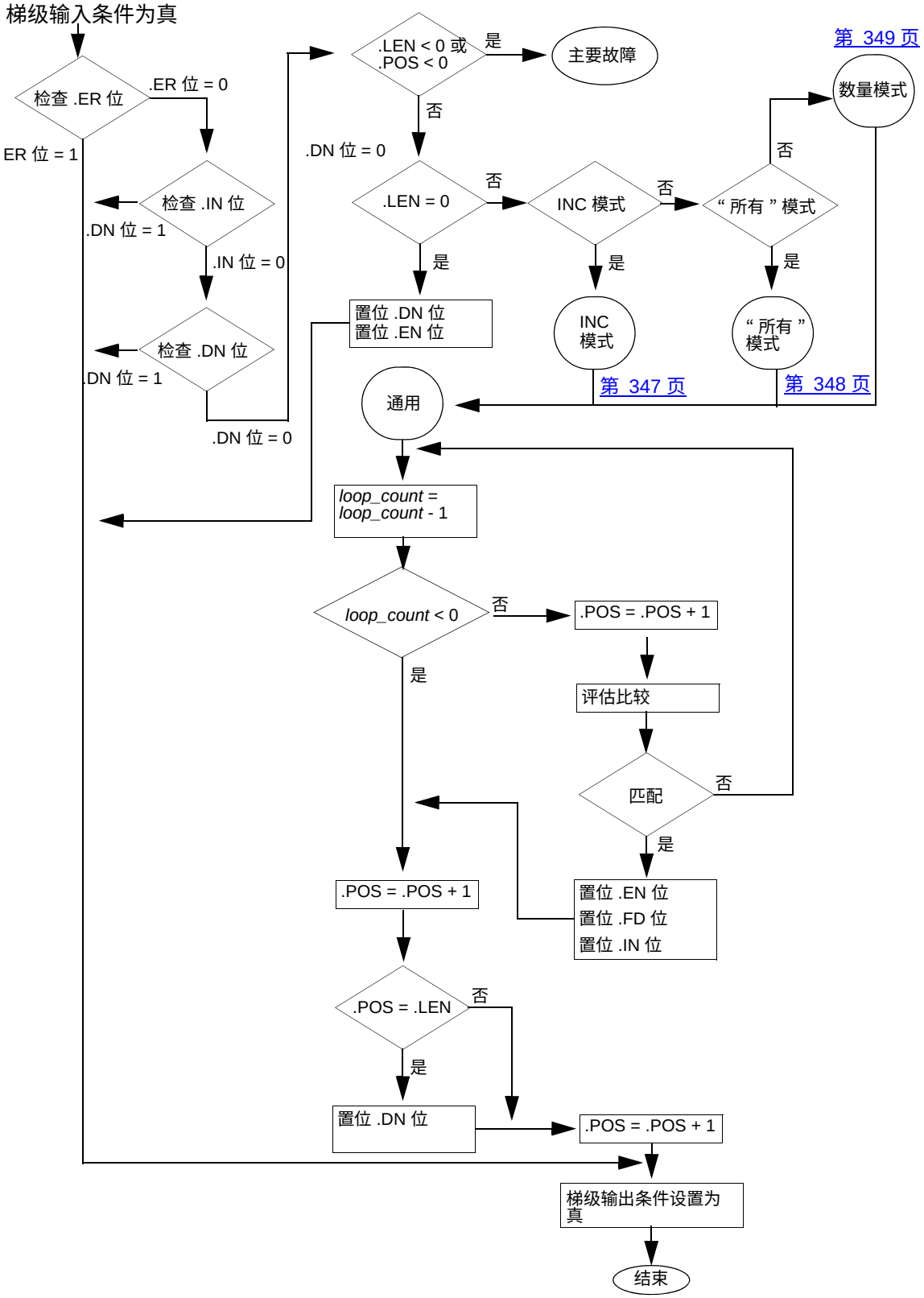
执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。



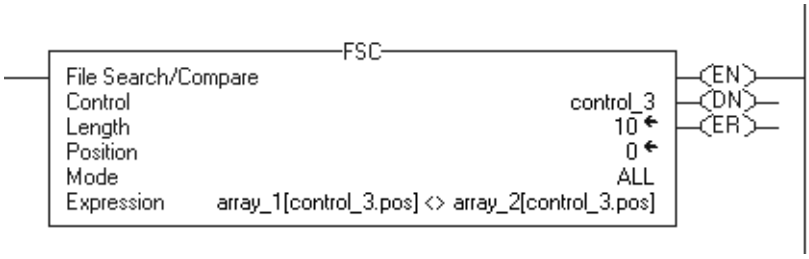
条件	梯形图操作

梯级输入条件为真



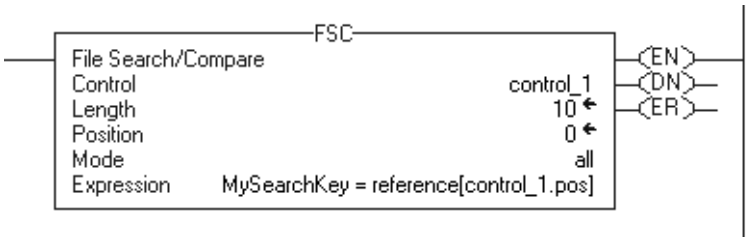
后扫描	梯级输出条件设置为假。

示例 **1**： 搜索两个数组之间的不匹配元素。使能后，FSC 指令将 *array_1* 中前 10 个元素的每一个与 *array_2* 中的对应元素进行比较。



array_1	*array_2*	*control_3.pos*	
00000000000000000000000000000000	00000000000000000000000000000000	0	
00000000000000000000000000000000	00000000000000000000000000000000	1	
00000000000000000000000000000000	00000000000000000000000000000000	2	
00000000000000000000000000000000	00000000000000000000000000000000	3	
00000000000000001111111111111111	11111111111111110000000000000000	4	→ FSC 指令发现这些元素不相等。该指令将置位 .FD 和 .IN 位。 .POS 值 (4) 指示不相等的元素的位置。要继续比较数组的剩余部分，请将 .IN 位清零。
11111111111111111111111111111111	11111111111111111111111111111111	5	
11111111111111111111111111111111	11111111111111111111111111111111	6	
11111111111111111111111111111111	11111111111111111111111111111111	7	
11111111111111111111111111111111	11111111111111111111111111111111	8	
11111111111111111111111111111111	11111111111111111111111111111111	9	

示例 **2**： 搜索一个数组中的匹配元素。使能后，FSC 指令将 *MySearchKey* 与 *array_1* 中的 10 个元素进行比较。



MySearchKey

11111111111111110000000000000000

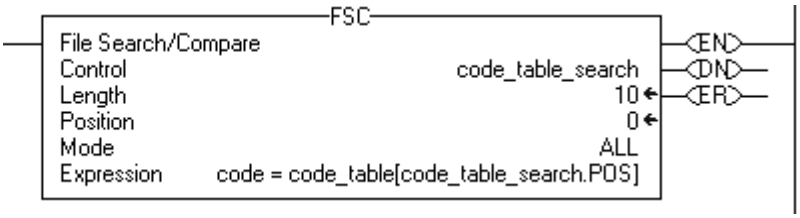
参考

00000000000000000000000000000000	0
00000000000000000000000000000000	1
00000000000000000000000000000000	2
00000000000000000000000000000000	3
11111111111111110000000000000000	4
11111111111111111111111111111111	5
11111111111111111111111111111111	6
11111111111111111111111111111111	7
11111111111111111111111111111111	8
11111111111111111111111111111111	9

control_3.pos

FSC 指令发现此数组元素等于 *MySearchKey*。该指令将置位 .FD 和 .IN 位。 .POS 值 (4) 指示相等元素的位置。要继续比较数组的剩余部分，请将 .IN 位清零。

示例 **3**： 在字符串数组中搜索字符串。使能后，FSC 指令将 *code* 中的字符与 *code_table* 中的 10 个元素进行比较。



Code
SAM

code_table	*code_table_search.POS*
AFG	0
BEH	1
HUO	2
SAK	3
SAM	4
FQG	5
CLE	6
CAK	7
DET	8
BWG	9

FSC 指令发现此数组元素等于 *code*。该指令将置位 .FD 和 .IN 位。.POS 值 (4) 指示相等元素的位置。要继续比较数组的剩余部分，请将 .IN 位清零。

FSC 表达式

FSC 指令中表达式的编写方法与 CMP 指令中表达式的编写方法相同。请查看以下部分，了解以上两种指令通用的有效运算符、格式和运算顺序。

有效运算符

运算符	说明	最佳类型
+	加	DINT、 REAL
-	减 / 取反	DINT、 REAL
*	乘	DINT、 REAL
/	除	DINT、 REAL
=	等于	DINT、 REAL
<	小于	DINT、 REAL
<=	小于等于	DINT、 REAL
>	大于	DINT、 REAL
>=	大于等于	DINT、 REAL
<>	不等于	DINT、 REAL
**	指数 (x 的 y 次幂)	DINT、 REAL
ABS	绝对值	DINT、 REAL
ACS	反余弦	REAL
AND	按位与	DINT
ASN	反正弦	REAL
ATN	反正切	REAL
COS	余弦	REAL

运算符	说明	最佳类型
DEG	弧度转角度	DINT、 REAL
FRD	BCD 转整数	DINT
LN	自然对数	REAL
LOG	以 10 为底的对数	REAL
MOD	模数除法	DINT、 REAL
NOT	按位求补	DINT
OR	按位或	DINT
RAD	角度转弧度	DINT、 REAL
SIN	正弦	REAL
SQR	平方根	DINT、 REAL
TAN	正切	REAL
TOD	整数转 BCD	DINT
TRN	截断	DINT、 REAL
XOR	按位异或	DINT

格式表达式

对于表达式中使用的每一个运算符，都需要提供一个或两个操作数 (标签或立即值)。有关表达式中运算符和操作数的格式，请遵循下表中的规定。

运算符涉及的操作数的个数	使用的格式	示例
一个操作数	运算符 (操作数)	ABS(*tag_a*)
两个操作数	操作数 _a 运算符 操作数 _b	• *tag_b* + 5 • *tag_c* AND *tag_d* • (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

确定运算顺序

写入表达式中的运算将由指令按照规定的顺序执行，不一定按照写入的顺序执行。通过用圆括号括起一些内容，可以更改运算顺序，这样指令就会先执行括号内的运算，后执行其它运算。

相同顺序的运算从左到右执行。

顺序	运算
1.	( )
2.	ABS、 ACS、 ASN、 ATN、 COS、 DEG、FRD、LN、LOG、RAD、SIN、 SQR、 TAN、 TOD、 TRN
3.	**
4.	– (取反)， NOT
5.	*、 /、 MOD
6.	<, <=, >, >=, =
7.	– (减)， +
8.	AND
9.	XOR
10.	OR

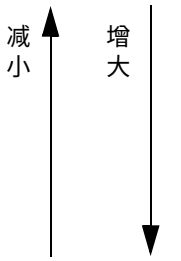
在表达式中使用字符串

要在表达式中使用 ASCII 字符串，请遵循以下原则：

- 通过表达式可比较两个字符串标签。
- 不能将 ASCII 字符直接输入到表达式中。
- 只允许使用下列运算符。

运算符	说明
=	等于
<	小于
<=	小于等于
>	大于
>=	大于等于
<>	不等于

- 如果字符串的字符匹配，则字符串相等。
- ASCII 字符区分大小写。大写字母 “ A ” ($41) *不* 等于小写字母 “ a ” ($61)。
- 字符的十六进制值决定了一个字符串小于还是大于另一个字符串。有关字符的十六进制代码，请参见本手册的封底。
- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。



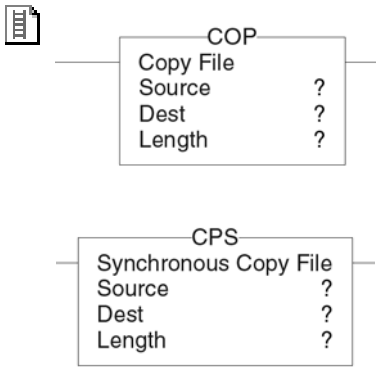
ASCII 字符	十六进制码	
1ab	$31$61$62	
1b	$31$62	
A	$41	
AB	$41$42	AB < B
B	$42	
a	$61	a > B
ab	$61$62	

复制文件 (COP) 同步复制文件 (CPS)

COP 和 CPS 指令可将 Source 中的值复制到 Destination 中。Source 保持不变。

操作数：

梯形图



操作数	类型	格式	说明
Source	**SINT** **INT** **DINT** **REAL** 字符串 结构	标签	要复制的起始元素 重要说明：Source 和 Destination 操作数的数据类型应该相同，否则可能产生意外结果
Destination	**SINT** **INT** **DINT** **REAL** 字符串 结构	标签	将被 Source 覆盖的初始元素 重要说明：Source 和 Destination 操作数的数据类型应该相同，否则可能产生意外结果
Length	DINT	立即数 标签	要复制的 Destination 元素的数目

结构化文本

```
COP(Source,Dest,Length);
CPS(Source,Dest,Length);
```

操作数与梯形图 COP 和 CPS 指令的操作数相同。

说明： 在 COP 和 CPS 指令的执行过程中，其它控制器操作可能尝试中断复制操作并更改源数据或目标数据。

如果源或目标为	并且您想要	则选择	备注
• 生产者标签 • 消费者标签 • I/O 数据 • 另一个任务可以覆盖的数据	防止数据在复制操作过程中被更改	CPS	• 试图中断 CPS 指令的任务将被延迟到指令完成。 • 要估计 CPS 指令的执行时间，请参见 《ControlLogix 系统用户手册》 (出版号 1756-UM001)。
	允许数据在复制操作过程中被更改	COP	
以上都不是	——→	COP	

复制的字节数为：

字节数 = Length * (Destination 数据类型的字节数)

注 意

如果字节数大于 Source 的长度，会将不可预知的数据复制到剩余的元素中。

重要事项

必须测试并确认指令不会更改您不希望更改的数据。

COP 和 CPS 指令对连续的存储器进行操作。它们进行直接的字节到字节的存储器复制。在某些情况下，指令会越过数组写入标签的其它子元素。如果 length 过大且标签是用户自定义的数据类型，就会发生这种情况。

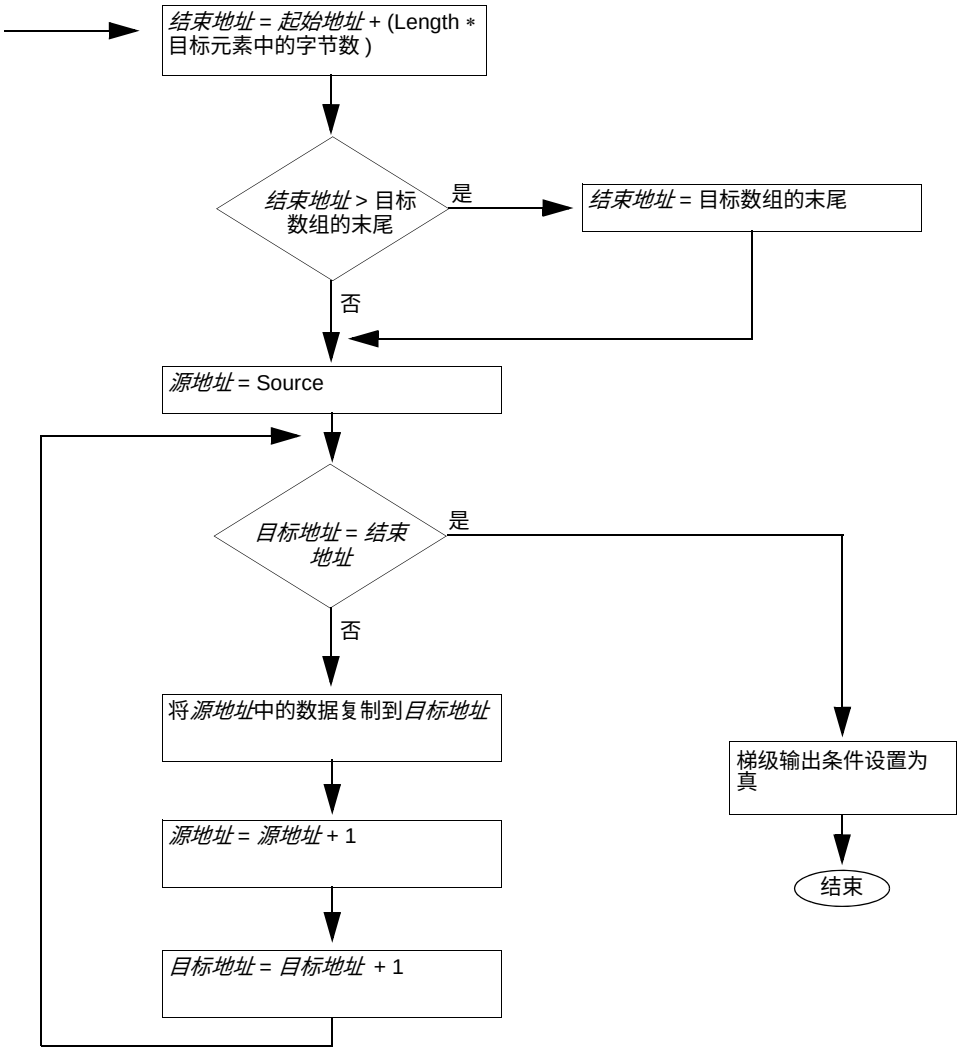
如果标签是	则
用户自定义的数据类型	如果 Length 过大，该指令将越过数组末尾写入标签的其它子元素。在标签的末尾将停止。不会产生主要故障。
非用户自定义的数据类型	如果 Length 过大，指令将在数组末尾停止。不会产生主要故障。

如果 Length 超过 Destination 数组中的元素总数，则说明 Length 过大。

算术状态标志： 不受影响

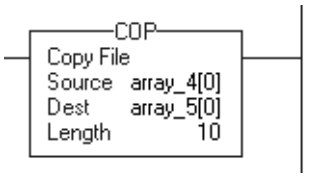
故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行		
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例 **1**： *array_4* 和 *array_5* 的数据类型相同。使能后， COP 指令将 *array_4* 的前 10 个元素复制到 *array_5* 的前 10 个元素。

梯形图

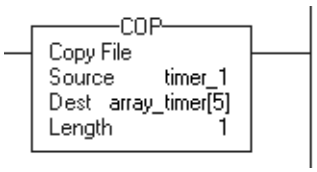


结构化文本

```
COP(array_4[0],array_5[0],10);
```

示例 **2**： 使能后， COP 指令将结构 *timer_1* 复制到 *array_timer* 的第 5 个元素中。该指令只将一个结构复制到一个数组元素中。

梯形图



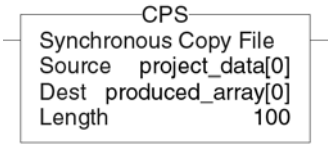
结构化文本

```
COP(timer_1,array_timer[5],1);
```

示例 **3**： *project_data* 数组 (100 个元素) 存储了各种会在应用项目的不同时间发生变化的值。为了将 *project_data* 在某一时刻的完整映像及时发送到另一个控制器， CPS 指令会将 *project_data* 复制到 *produced_array*。

- 在 CPS 指令复制数据的同时，任何 I/O 更新或其它任务都不能更改数据。
- *produced_array* 标签在 ControlNet 网络上生成供其它控制器使用的数据。
- 为了使用数据的相同映像 (即数据的同步副本)，消费者标签控制器将使用 CPS 指令将数据从消费者标签复制到另一个标签，以供应用项目使用。

梯形图



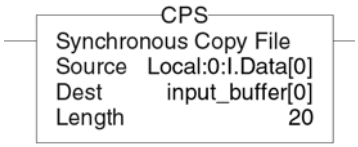
结构化文本

```
CPS(project_data[0],produced_array[0],100);
```

示例 **4**： *Local:0:I.Data* 存储了与插槽 0 中的 1756-DNB 模块相连接的 DeviceNet 网络的输入数据。为了使输入与应用项目同步， CPS 指令将输入数据复制到 *input_buffer*。

- 在 CPS 指令复制数据的同时， I/O 更新不能改变数据。
- 当应用项目执行时，它将 *input_buffer* 中的输入数据用于其输入。

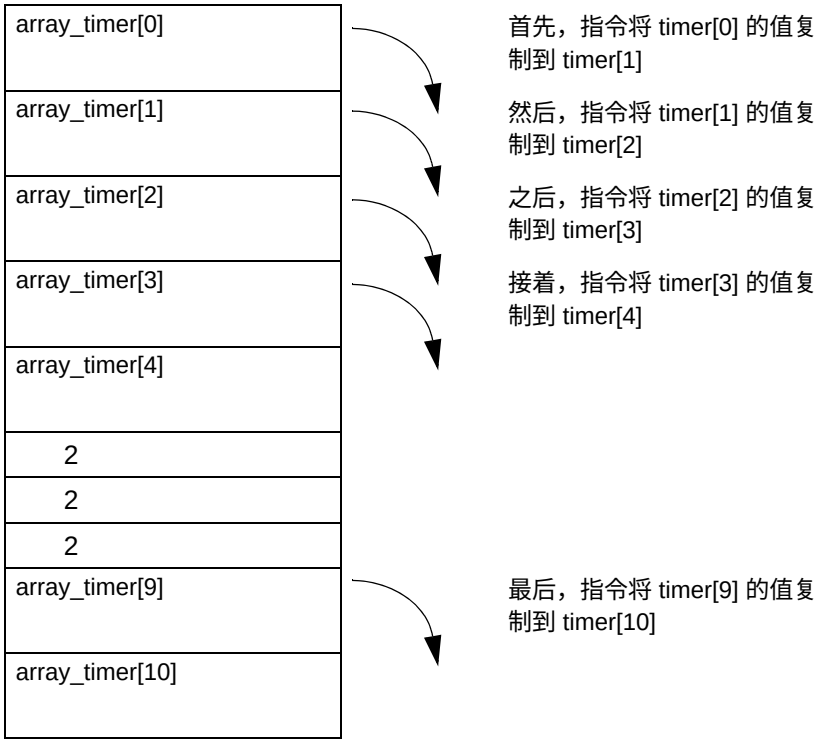
梯形图



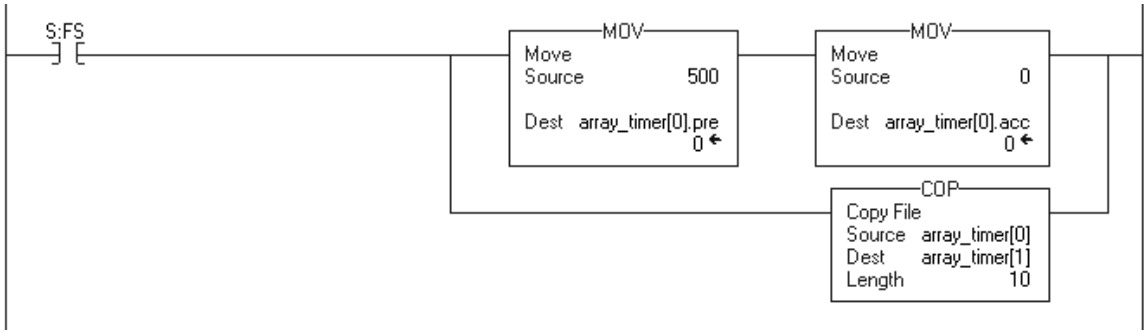
结构化文本

```
CPS(Local:0:I.Data[0],input_buffer[0],20);
```

示例 **5**： 此示例初始化一个计时器结构的数组。使能后， MOV 指令初始化第一个 *array_timer* 元素的 .PRE 和 .ACC 值。使能后， COP 指令将从 *array_timer[0]* 开始复制一个连续字节块。长度为九个计时器结构。



梯形图



结构化文本

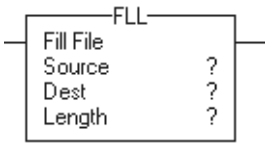
```
IF S:FS THEN
     array_timer[0].pre := 500;
     array_timer[0].acc := 0;
     COP(array_timer[0],array_timer[1],10);
END_IF;
```

文件填充 (FLL)

FLL 指令用 Source 值填充数组元素。 Source 保持不变。

操作数：

梯形图



操作数	类型	格式：	说明
Source	**SINT** **INT** **DINT** **REAL**	立即数 标签	要复制的元素 重要说明：Source 和 Destination 操作数的数据类型应该相同，否则可能产生意外结果
Destination	**SINT** **INT** **DINT** **REAL** 结构	标签	将被 Source 覆盖的初始元素 重要说明：Source 和 Destination 操作数的数据类型应该相同，否则可能产生意外结果 初始化结构的首选方法是使用 COP 指令。
Length	DINT	立即数	要填充的元素数目

结构化文本

结构化文本没有 FLL 指令，但可以通过 SIZE 指令和 FOR...DO 或其它循环结构来实现相同的结果。

```
SIZE(destination,0,length);
FOR position = 0 TO length-1 DO
    destination[position] := source;
END_FOR;
```

有关结构化文本中结构语法的信息，请参见结构化文本编程。

说明： 填充的字节数为：

字节数 = Length * (Destination 数据类型的字节数)

重要事项

必须测试并确认指令不会更改您不希望更改的数据。

FLL 指令对连续数据存储器进行操作。在某些情况下，该指令会越过数组写入标签的其它子元素。如果 length 过大且标签是用户自定义的数据类型，就会发生这种情况。

如果标签是	则
用户自定义的数据类型	如果 Length 过大，该指令将越过数组末尾写入标签的其它子元素。在标签的末尾将停止。不会产生主要故障。
非用户自定义的数据类型	如果 Length 过大，指令将在数组末尾停止。不会产生主要故障。

如果 Length 超过 Destination 数组中的元素总数，则说明 Length 过大。

为获得最好的结果， Source 和 Destination 的类型应该相同。如果要填充结构，请使用 COP 指令 (请参见第 359 页中的示例 3)。如果将 Source 和 Destination 的数据类型混用，则 Destination 元素将填充转换后的 Source 值。

Source 的数据类型	Destination 的数据类型	Source 转换后的数据类型
SINT、 INT、 DINT 或 REAL	SINT	SINT
SINT、 INT、 DINT 或 REAL	INT	INT
SINT、 INT、 DINT 或 REAL	DINT	DINT
SINT、 INT、 DINT 或 REAL	REAL	REAL
SINT	结构	SINT(不转换)
INT	结构	INT(不转换)
DINT	结构	DINT(不转换)
REAL	结构	REAL(不转换)

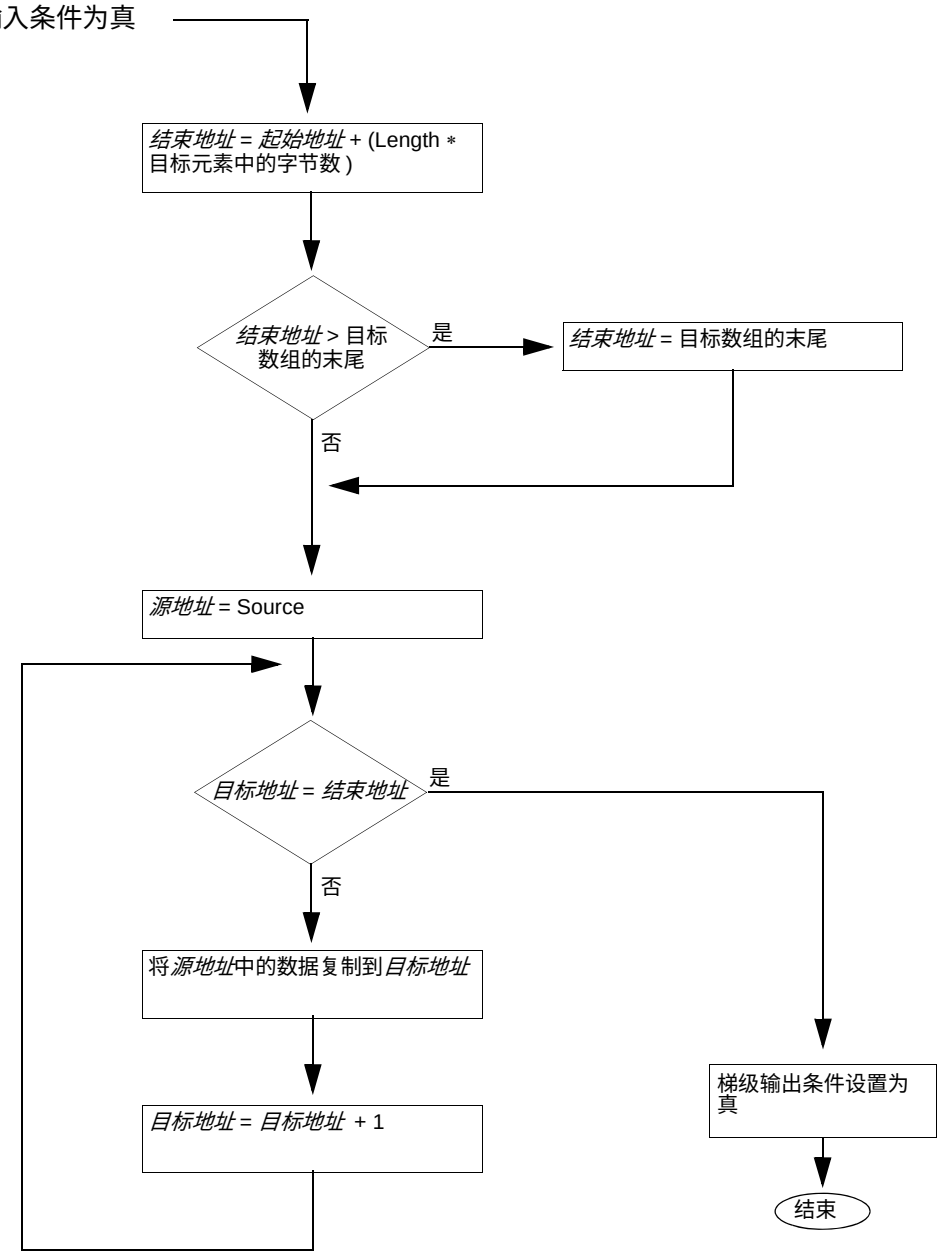
算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。

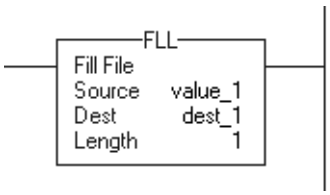
梯级输入条件为真



后扫描	梯级输出条件设置为假。

示例： FLL 指令将 *value_1* 中的值复制到 *dest_1*

梯形图



Source (*value_1*) 数据类型	Source (*value_1*) 值	Destination (*dest_1*) 数据类型	执行 FLL 指令后的 Destination (*dest_1*) 值
SINT	16#80 (-128)	DINT	16#FFFF FF80 (-128)
DINT	16#1234 5678	SINT	16#78
SINT	16#01	REAL	1.0
REAL	2.0	INT	16#0002
SINT	16#01	TIMER	16#0101 0101 16#0101 0101 16#0101 0101
INT	16#0001	TIMER	16#0001 0001 16#0001 0001 16#0001 0001
DINT	16#0000 0001	TIMER	16#0000 0001 16#0000 0001 16#0000 0001

结构化文本

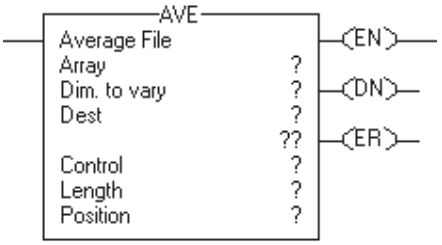
```
dest_1 := value_1;
```

文件平均值 (AVE)

AVE 指令计算一组值的平均值。

操作数：

梯形图



操作数	类型	格式	说明
Array	SINT INT DINT **REAL**	数组标签	计算此数组中值的平均值 指定要计算平均值的元素组的第一个元素 不要在下标中使用 CONTROL.POS
Dimension to vary	DINT	立即数 (0, 1, 2)	要使用的维度 取决于维数，顺序为： array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Destination	SINT INT DINT **REAL**	标签	运算结果
Control	CONTROL	标签	操作的控制结构
Length	DINT	立即数	要计算平均值的数组中的元素数目
Position	DINT	立即数	数组中的当前元素 初始值通常为 0

结构化文本

结构化文本没有 AVE 指令，但可以通过 SIZE 指令和 FOR...DO 或其它循环结构来实现相同的结果。

```
SIZE(array,0,length);
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + array[position];
END_FOR;
destination := sum / length;
```

有关结构化文本中结构语法的信息，请参见结构化文本编程。

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 AVE 指令是否使能。
.DN	BOOL	指令对数组中最后一个元素进行运算后 (.POS = .LEN)，完成位被置位。
.ER	BOOL	如果指令发生溢出，则错误位置位。指令停止执行直到程序清零 .ER 位。导致溢出的元素的位置存储在 .POS 值中。
.LEN	DINT	长度用于指定指令操作的数组元素数目。
.POS	DINT	位置包含指令正在访问的当前元素的位置。

说明： AVE 指令计算一组值的平均值。

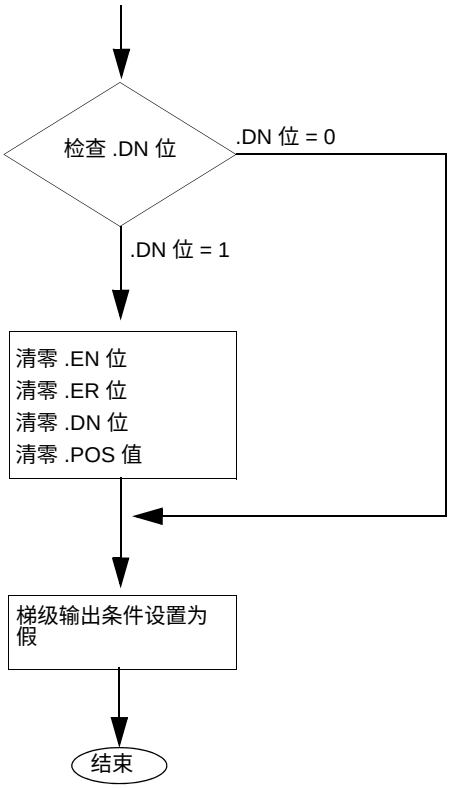
重要事项 确保 Length 不会使指令超出指定的 Dimension to vary。如果发生这种情况， Destination 将不正确。

算术状态标志： 算术状态标志将受到影响。

故障条件：

出现主要故障的条件	故障类型	故障代码
.POS < 0 或 .LEN < 0	4	21
指定数组的 Dimension to vary 不存在	4	20

执行：

条件	梯形图操作
预扫描	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。 梯级输出条件设置为假。
梯级输入条件为假	
梯级输入条件为真	AVE 指令通过将数组中所有指定的元素相加并除以元素数目来计算平均值。 该指令在内部使用 FAL 指令计算平均值： 表达式 = 平均值计算 模式 = 所有 有关 FAL 指令如何执行的详细信息，请参见第 345 页。
后扫描	梯级输出条件设置为假。

示例 **1**： 计算 *array_dint* 的平均值，其数据类型为 DINT[4,5]。

维度 1

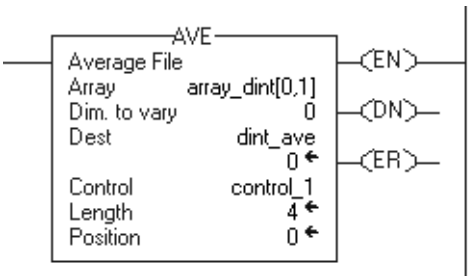
下标	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

维度 0

$$AVE = \frac{19 + 14 + 9 + 4}{4} = \frac{46}{4} = 11.5$$

dint_ave = 12

梯形图



结构化文本

```
SIZE(array_dint,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;
```

示例 **2**： 计算 *array_dint* 的平均值，其数据类型为 DINT[4,5]。

维度 1

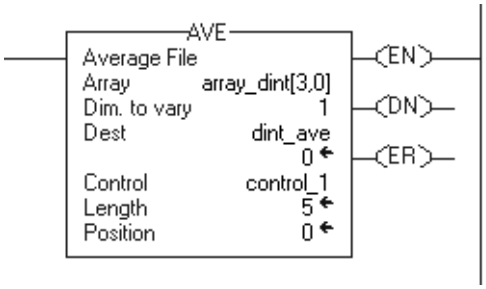
下标	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

维度 0

$$AVE = \frac{5+4+3+2+1}{5} = \frac{15}{5} = 3$$

dint_ave = 3

梯形图



结构化文本

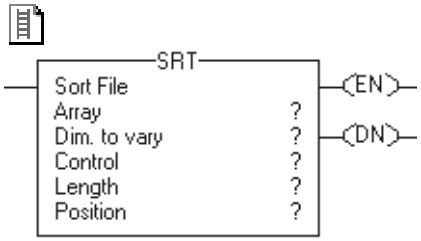
```
SIZE(array_dint,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;
```

文件排序 **(SRT)**

SRT 指令对 Array 的一个维度 (Dim to vary) 中的一组数值按升序进行排列。

操作数：

梯形图



操作数	类型	格式	说明
Array	**SINT** **INT** **DINT** **REAL**	数组标签	要排序的数组 指定要排序的元素组的第一个元素 不要在下标中使用 CONTROL.POS
Dimension to vary	DINT	立即数 (0, 1, 2)	要使用的维度 取决于维数，顺序为： array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Control	CONTROL	标签	操作的控制结构
Length	DINT	立即数	要排序的数组的元素数目
Position	DINT	立即数	数组中的当前元素 初始值通常为 0

结构化文本

```
SRT(Array,Dimtovary,
     Control);
```

操作数与梯形图 SRT 指令的操作数相同。但是，指定 Length 和 Position 值的方式是访问 CONTROL 结构的 .LEN 和 .POS 子元素，而不是将值包括在操作数列表中。

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 SRT 指令是否使能。
.DN	BOOL	当指定的元素经过排序后，完成位置位。
.ER	BOOL	如果 .LEN < 0 或 .POS < 0，错误位将置位。这两种情况中的任何一种都会产生主要故障。
.LEN	DINT	长度用于指定指令操作的数组元素数目。
.POS	DINT	位置包含指令正在访问的当前元素的位置。

说明： SRT 指令对 Array 的一个维度 (Dim to vary) 中的一组数值按升序进行排列 .

重要事项	必须测试并确认指令不会更改您不希望更改的数据。 SRT 指令对连续存储器进行操作。在某些情况下，该指令会更改标签的其它子元素中的数据。如果 length 过大且标签是用户自定义的数据类型，就会发生这种情况。

重要事项	确保 Length 不会使指令超出指定的 Dimension to vary。如果发生这种情况，将产生意外的结果。

这是一条触发执行指令。

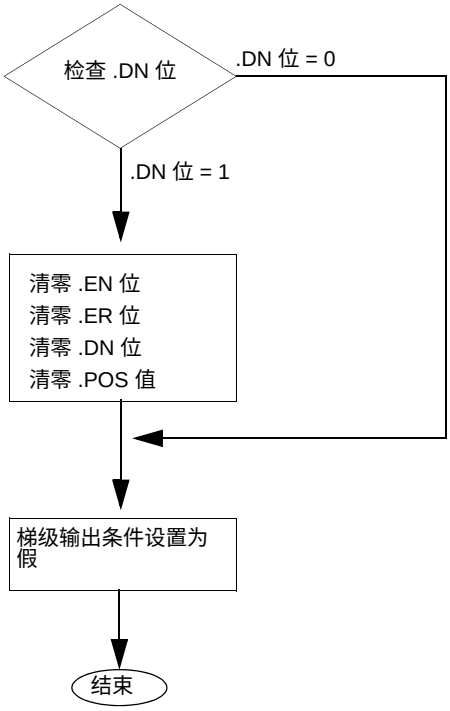
- 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
- 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。请参见结构化文本编程。

算术状态标志： 算术状态标志将受到影响。

故障条件：

出现主要故障的条件	故障类型	故障代码
.POS < 0 或 .LEN < 0	4	21
指定数组的 Dimension to vary 不存在	4	20
指令试图访问数组边界以外的数据	4	20

执行：

条件	梯形图操作	结构化文本操作
预扫描	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。 梯级输出条件设置为假。	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。
梯级输入条件为假		N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令按升序排列指定的数组元素。	该指令按升序排列指定的数组元素。
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例 **1**： 排序 *int* _array，其数据类型为 DINT[4,5]。

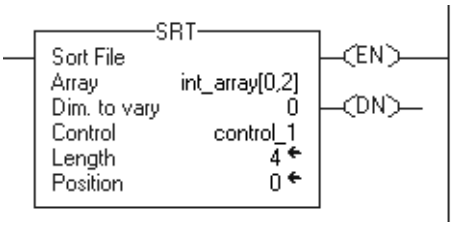
排序前

维度 0 \ 下标	维度 1 0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

排序后

维度 0 \ 下标	维度 1 0	1	2	3	4
0	20	19	3	17	16
1	15	14	8	12	11
2	10	9	13	7	6
3	5	4	18	2	1

梯形图



结构化文本

```
control_1.LEN := 4;

control_1.POS := 0;

SRT(int_array[0,2],0,control_1);
```

示例 **2**： 排序 *int* _array，其数据类型为 DINT[4,5]。

排序前

维度 0 / 维度 1

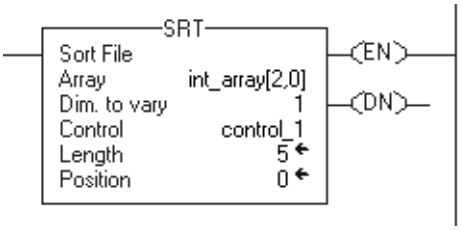
下标	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

排序后

维度 0 / 维度 1

下标	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	6	7	8	9	10
3	5	4	3	2	1

梯形图



结构化文本

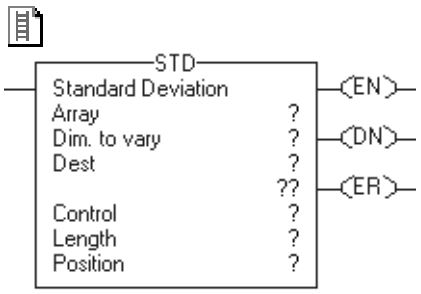
```
control_1.LEN := 5;

control_1.POS := 0;

SRT(int_array[2,0],1,control_1);
```

文件标准偏差 (STD)

STD 指令计算 Array 的某一维度中一组数值的标准偏差，并将结果存储在 Destination 中。

操作数：

梯形图



操作数	类型	格式	说明
Array	SINT INT **DINT** **REAL**	数组标签	计算此数组中值的标准偏差 指定要用于计算标准偏差的元素组的第一个元素 不要在下标中使用 CONTROL.POS
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
Dimension to vary	DINT	立即数 (0, 1, 2)	要使用的维度 取决于维数，顺序为： array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Destination	REAL	标签	运算结果
Control	CONTROL	标签	操作的控制结构
Length	DINT	立即数	用于计算标准偏差的数组的元素数目
Position	DINT	立即数	数组中的当前元素 初始值通常为 0

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 STD 指令是否使能。
.DN	BOOL	当计算完成时，完成位置位。
.ER	BOOL	如果指令发生溢出，则错误位置位。指令停止执行直到程序清零 .ER 位。导致溢出的元素的位置存储在 .POS 值中。
.LEN	DINT	长度用于指定指令操作的数组元素数目。
.POS	DINT	位置包含指令正在访问的当前元素的位置。

结构化文本

结构化文本没有 STD 指令，但可以通过 SIZE 指令和 FOR...DO 或其它循环结构来实现相同的结果。

```
SIZE(array,0,length);
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO length-1 DO
    sum := sum + ((array[position] - average)**2);
END_FOR;
destination := SQRT(sum /(length-1));
```

有关结构化文本中结构语法的信息，请参见结构化文本编程。

说明： 标准偏差的计算公式如下：

$$\text{标准偏差} = \sqrt{\frac{\left(\sum_{i=1}^{N}[\langle X_{(start+i)} - AVE\rangle^2]\right)}{(N-1)}}$$

其中：

- start = 数组操作数的 dimension-to-vary 下标
- x_i = 数组中的变量元素
- N = 数组中指定元素的数目

- AVE = $\dfrac{\left(\sum_{i=1}^{N} X(start+i)\right)}{N}$

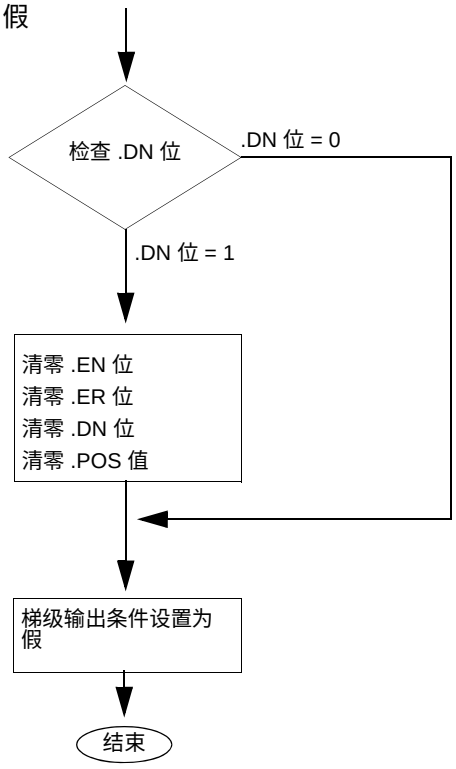
重要事项	确保 Length 不会使指令超出指定的 Dimension to vary。如果发生这种情况， Destination 将不正确。

算术状态标志： 算术状态标志将受到影响。

故障条件：

出现主要故障的条件	故障类型	故障代码
.POS < 0 或 .LEN < 0	4	21
指定数组的 Dimension to vary 不存在	4	20

执行：

条件	梯形图操作
预扫描	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。 梯级输出条件设置为假。
梯级输入条件为假	
梯级输入条件为真	STD 指令用于计算指定元素的标准偏差。 该指令在内部使用 FAL 指令计算平均值： 表达式 = 标准偏差计算 模式 = 所有 有关 FAL 指令如何执行的详细信息，请参见第 345 页。
后扫描	梯级输出条件设置为假。

示例 **1**： 计算 *dint_array* 的标准偏差，其数据类型为 DINT[4,5]。

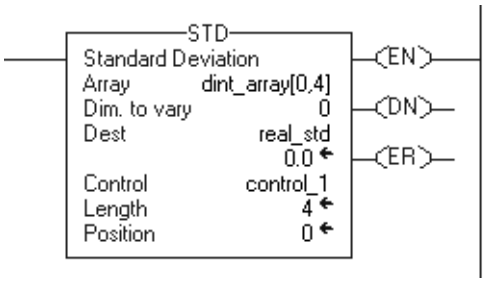
下标	维度 1				
维度 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$AVE = \frac{16 + 11 + 6 + 1}{4} = \frac{34}{4} = 8.5$$

$$STD = \sqrt{\frac{\langle 16 - 8.5 \rangle^2 + \langle 11 - 8.5 \rangle^2 + \langle 6 - 8.5 \rangle^2 + \langle 1 - 8.5 \rangle^2}{\langle 4 - 1 \rangle}} = 6.454972$$

real_std = 6.454972

梯形图



结构化文本

```
SIZE(dint_array,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
```

示例 **2**： 计算 d*int_array* 的标准偏差，其数据类型为 DINT[4,5]。

维度 1

下标	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

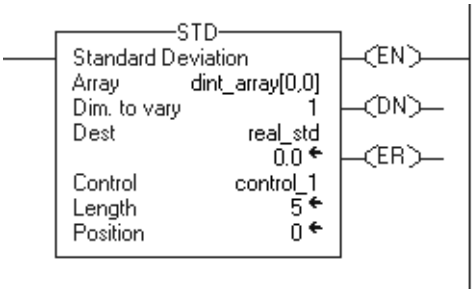
维度 0

$$AVE = \frac{20 + 19 + 18 + 17 + 16}{5} = \frac{90}{5} = 18$$

$$STD = \sqrt{\frac{\langle 20 - 18\rangle^2 + \langle 19 - 18\rangle^2 + \langle 18 - 18\rangle^2 + \langle 17 - 18\rangle^2 + \langle 16 - 18\rangle^2}{\langle 5 - 1\rangle}} = 1.581139$$

real_std = 1.581139

梯形图



结构化文本

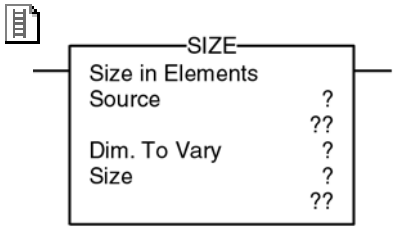
```
SIZE(dint_array,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
```

元素尺寸 (SIZE)

SIZE 指令用于找出数组某一维度的尺寸。

操作数：

梯形图



操作数	类型	格式	说明
Source	SINT INT DINT REAL 结构 字符串	数组标签	指令要操作的数组
Dimension to Vary	DINT	立即数 (0、 1、 2)	要使用的维度： 获取以下维度的大小 | 输入 第一维 | 0 第二维 | 1 第三维 | 2
Size	SINT INT DINT REAL	标签	用于存储数组的指定维度中的元素数目的标签

结构化文本

```
SIZE(Source,Dimtovary,Size);
```

操作数与梯形图 SIZE 指令的操作数相同。

说明： SIZE 指令用于找出 Source 数组的指定维度中的元素数目 (大小)，并将结果放到 Size 操作数中。

- 该指令可找出数组某一维度的尺寸。
- 该指令可操作：
  - 数组
  - 结构数组
  - 作为较大数组的一部分的数组

算术状态标志： 不受影响

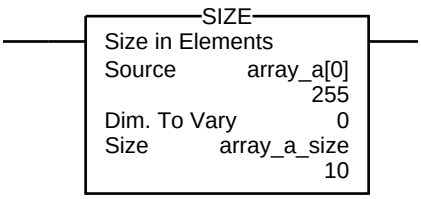
故障条件： 无。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令可计算某一维度的大小。	该指令可计算某一维度的大小。
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例 **1**： 找出 *array_a* 的维度 0(第一维) 的元素数目。将尺寸存储在 *array_a_size* 中。在此示例中， *array_a* 的维度 0 有 10 个元素。

梯形图

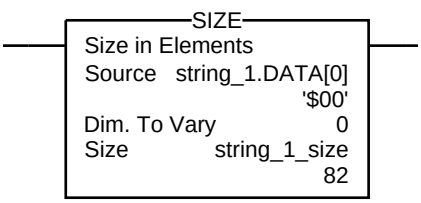


结构化文本

```
SIZE(array_a,0,array_a_size);
```

示例 **2**： 找出 *string_1*(一个字符串) 的 DATA 子元素中的元素个数。将尺寸存储在 *string_1_size* 中。在此示例中， *string_1* 的 DATA 子元素有 82 个元素。 (字符串使用默认的 STRING 数据类型。) 由于每个元素存储一个字符，因此 *string_1* 最多可包含 82 个字符。

梯形图

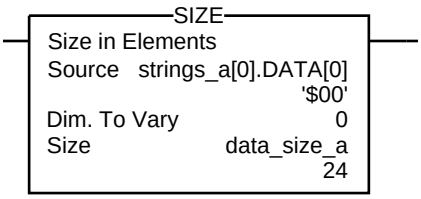


结构化文本

```
SIZE(string_1.DATA[0],0,string_1_size);
```

示例 **3**： *Strings_a* 是一个字符串结构数组。 SIZE 指令可找出字符串结构的 DATA 子元素中的元素个数，并将尺寸存储到 *data_size_a* 中。在此示例中，DATA 子元素有 24 个元素。(字符串结构的用户指定长度为 24。)

梯形图



结构化文本

```
SIZE(strings_a[0].DATA[0],0,data_size_a);
```

注：

数组 (文件)/ 移位指令
(BSL、 BSR、 FFL、 FFU、 LFL、 LFU)

简介

数组 (文件)/ 移位指令可用于修改数组中数据的位置。

如果要	使用以下指令	在以下语言中可用	页码
以一次一位的方式将位装载到位数组中、在位数组中移位或者从位数组中卸载位	BSL	梯形图	394
	BSR	梯形图	398
按照相同的顺序装载和卸载值	FFL	梯形图	402
	FFU	梯形图	408
按照相反的顺序装载和卸载值	LFL	梯形图	414
	LFU	梯形图	420

可以混合使用不同数据类型，但可能会造成精度降低和舍入错误。

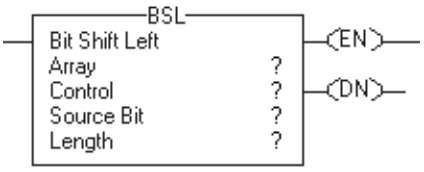
对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

位左移 (BSL)

BSL 指令用于将数组中指定的位向左移一个位置。

操作数：

梯形图



操作数	类型	格式	说明
Array	DINT	数组标签	要修改的数组 指定要开始移位的元素 不要在下标中使用 CONTROL.POS
Control	CONTROL	标签	操作的控制结构
Source bit	BOOL	标签	要装载的位
Length	DINT	立即数	要移位的数组中位的个数

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	该使能位指示 BSL 指令是否使能。
.DN	BOOL	该完成位置位时指示位左移了一个位置。
.UL	BOOL	该卸载位是指令的输出。 .UL 位用于存储移出位的状态。
.ER	BOOL	当 .LEN < 0 时，错误位将被置位。
.LEN	DINT	长度用于指定要移位的数组位的个数。

说明： 使能后，指令会将指定位的最高位卸载到 .UL 位，将其余的位向左移一个位置，然后将源位装载到数组的位 0 中。

重要事项

必须测试并确认指令没有更改不想更改的数据。

BSL 指令针对连续内存操作。如果数组是子元素数组 (例如，包含在结构中的数组)，指令可能越过数组的边界移位到数组后面的其它子元素中。因此，选择长度时必须注意，不要让这种情况发生。

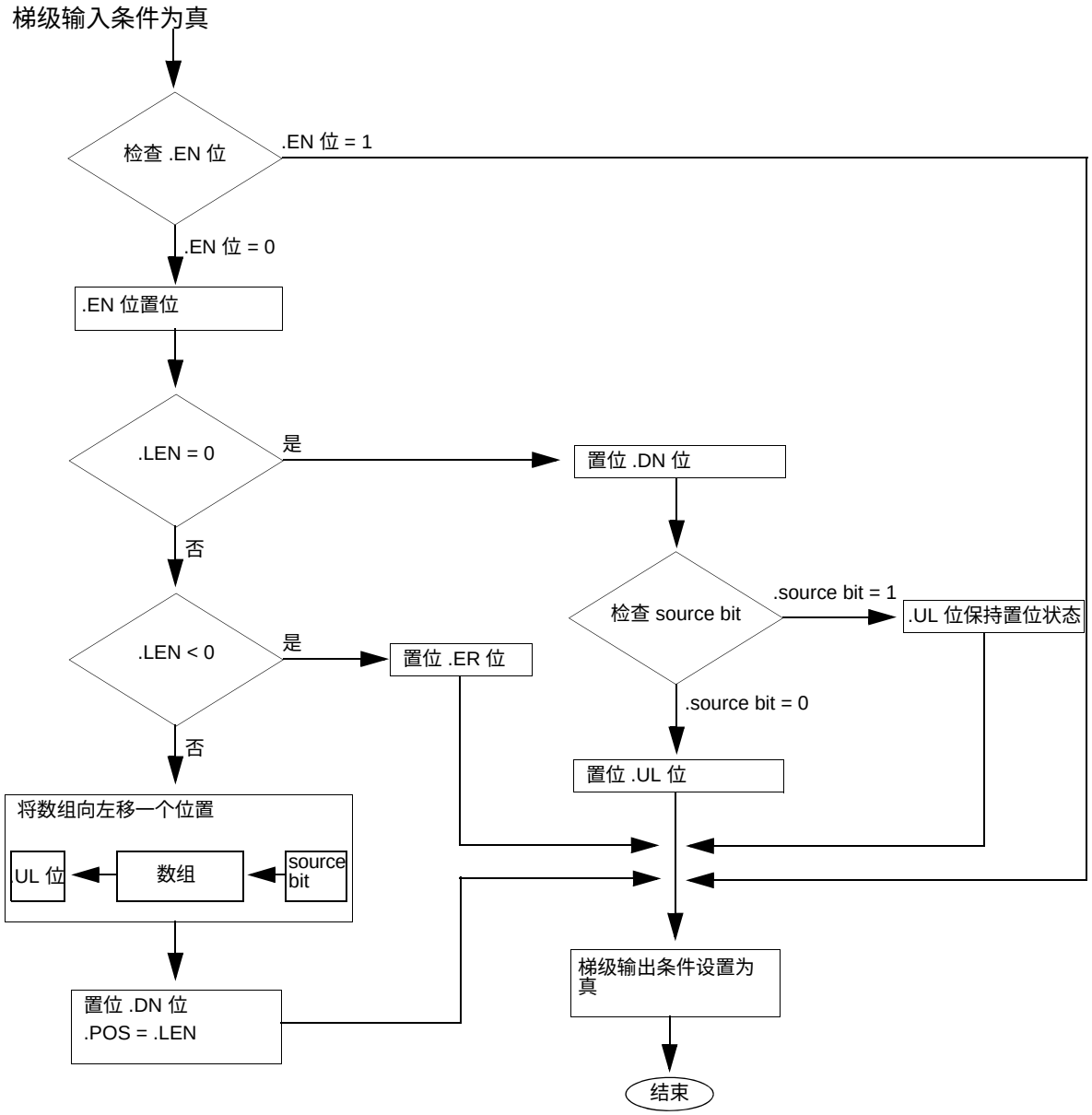
算术状态标志： 不受影响

错误条件：

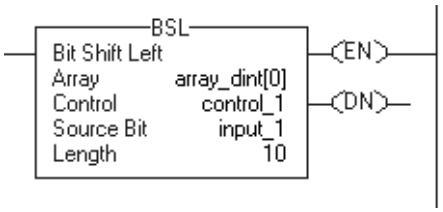
出现严重错误的条件	错误类型	错误代码
长度超过数组的存储区的尺寸。	4	20

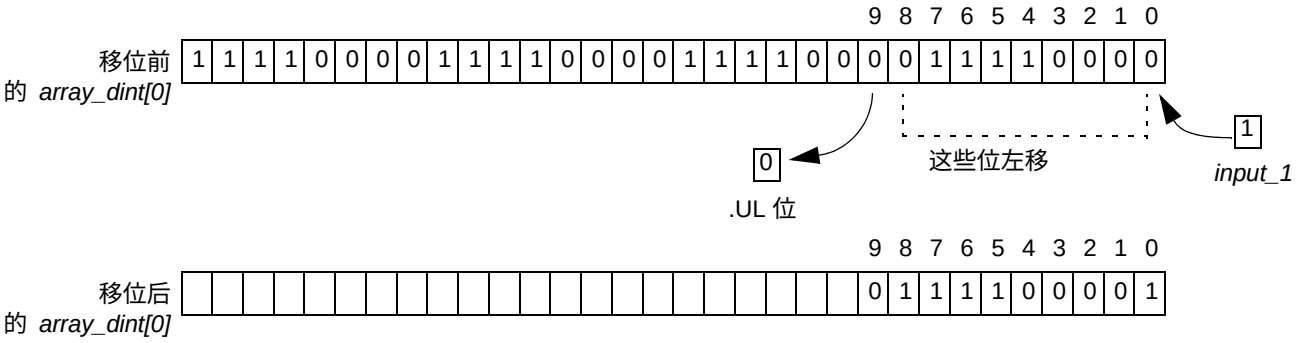
执行：

条件	梯形图操作
预扫描	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。 清零 .POS 值。 梯级输出条件设置为假。
梯级输入条件为假	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。 清零 .POS 值。 梯级输出条件设置为假。

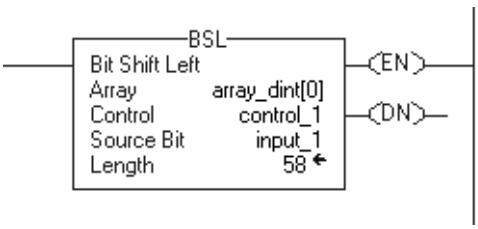
条件	梯形图操作
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

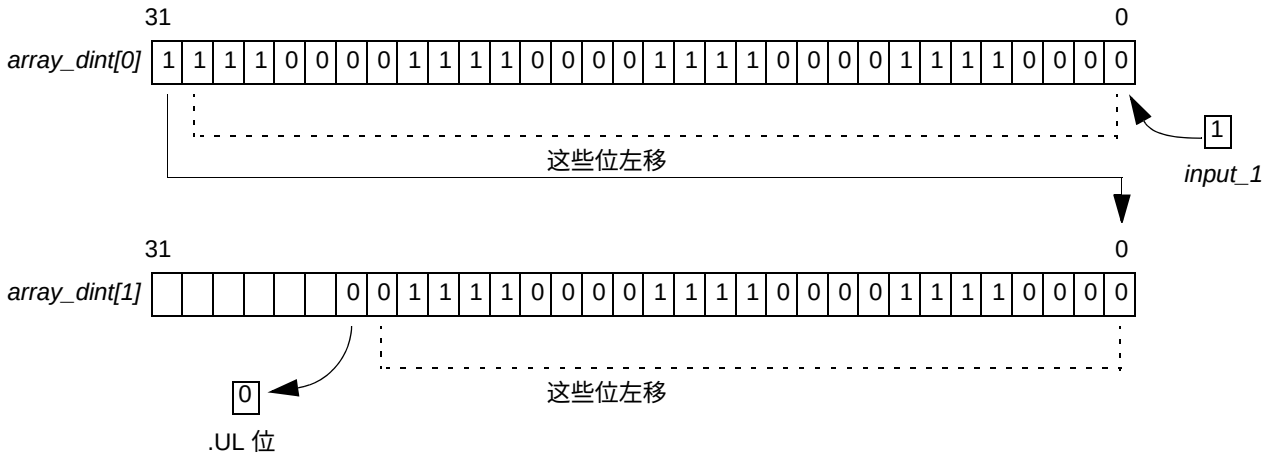
示例 **1**： 使能后， BSL 指令将从 *array_dint[0] 中的位 0 开始执行。*指令将 *array_dint[0].9* 卸载到 .UL 位，对其余的位移位，并将 *input_1* 装载到 *array_dint[0].0* 中。其余的位 (10…31) 中的值无效。





示例 **2**： 使能后， BSL 指令将从 *array_dint[0] 中的位 0 开始执行。*指令将 *array_dint[1].25* 卸载到 .UL 位，对其余的位移位，并将 *input_1* 装载到 *array_dint[0].0* 中。其余的位 (*array_dint[1]* 中的 31…26) 中的值无效。注意 *array_dint[0].31* 是如何经过字移位到 *array_dint[1].0* 的。



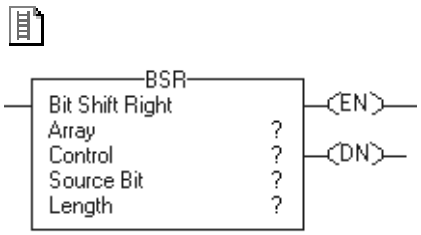


位右移 (BSR)

BSR 指令用于将数组中指定的位向右移一个位置。

操作数：

梯形图



操作数	类型	格式	说明
Array	DINT	数组标签	要修改的数组 指定要开始移位的元素 不要在下标中使用 CONTROL.POS
Control	CONTROL	标签	操作的控制结构
Source bit	BOOL	标签	要装载的位
Length	DINT	立即数	要移位的数组中位的个数

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	该使能位指示 BSR 指令是否使能。
.DN	BOOL	该完成位置位时指示位右移了一个位置。
.UL	BOOL	该卸载位是指令的输出。 .UL 位用于存储移出位的状态。
.ER	BOOL	当 .LEN < 0 时，错误位将被置位。
.LEN	DINT	长度用于指定要移位的数组位的个数。

说明： 使能后，指令会将数组的位 0 处的值卸载到 .UL 位，将其余的位向右移一个位置，然后将源位装载到指定位的最高位中。

重要事项

必须测试并确认指令没有更改不想更改的数据。

BSR 指令针对连续内存操作。如果数组是子元素数组 (例如，包含在结构中的数组)，指令可能越过数组的边界移位到数组后面的其它子元素中。因此，选择长度时必须注意，不要让这种情况发生。

算术状态标志： 不受影响

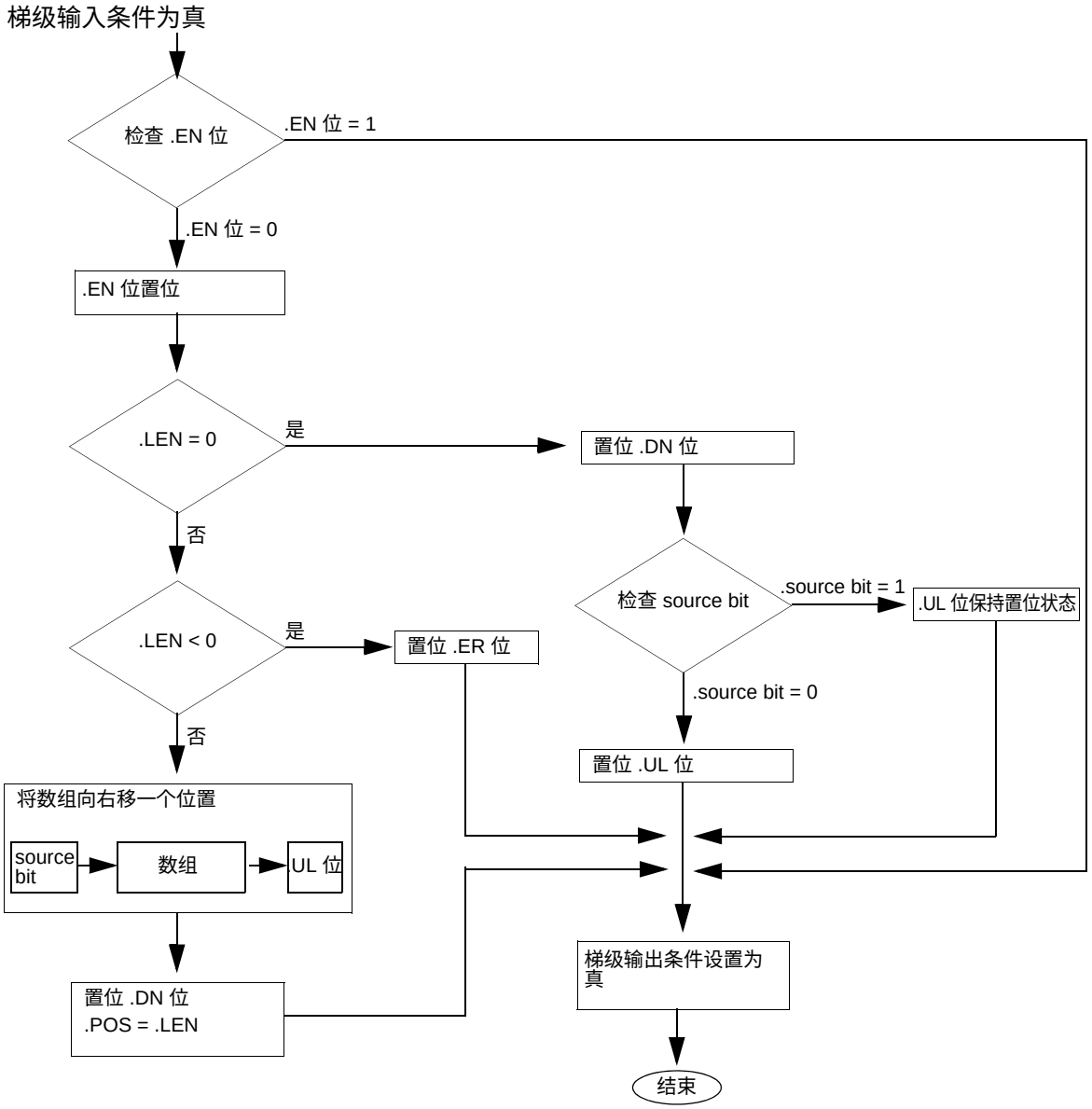
错误条件：

出现严重错误的条件	错误类型	错误代码
长度超过数组的存储区的尺寸。	4	20

执行：

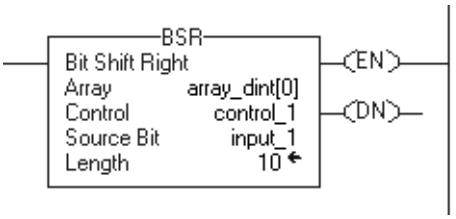
条件	梯形图操作
预扫描	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。 清零 .POS 值。 梯级输出条件设置为假。
梯级输入条件为假	清零 .EN 位。 清零 .DN 位。 清零 .ER 位。 清零 .POS 值。 梯级输出条件设置为假。

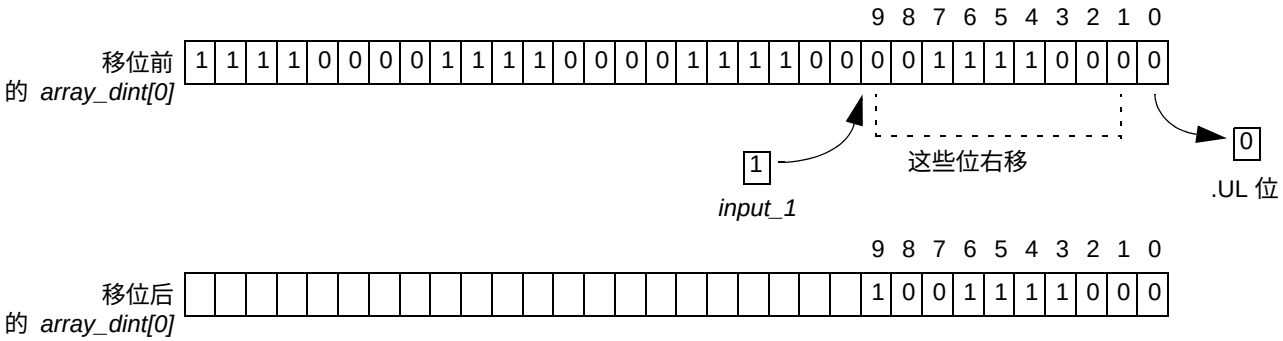
条件	梯形图操作



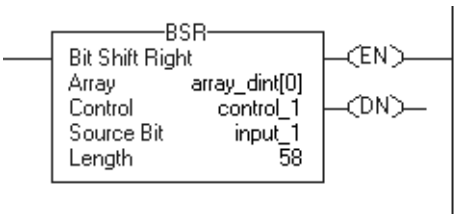
后扫描	梯级输出条件设置为假。

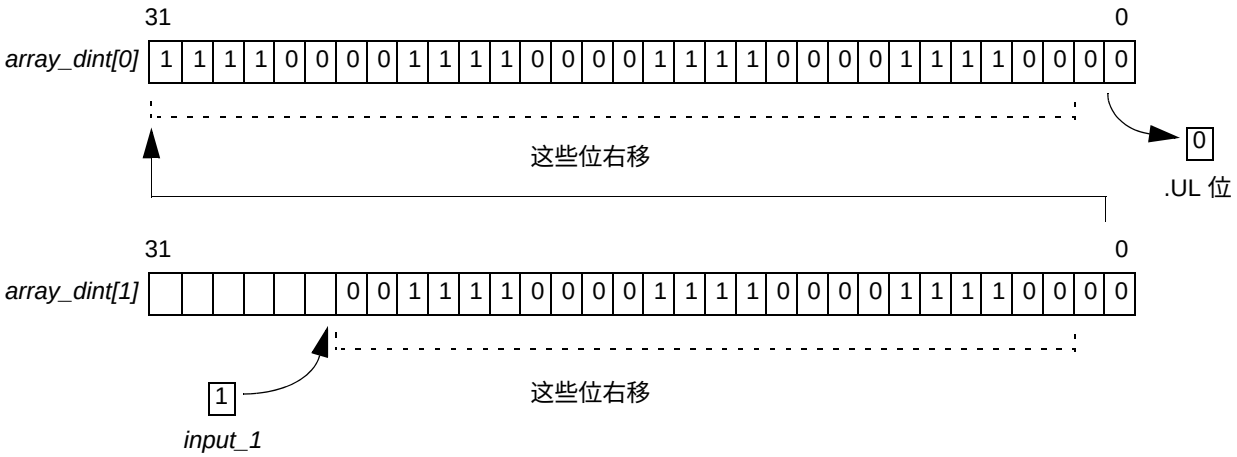
示例 **1**： 使能后， BSR 指令将从 *array_dint[0] 中的位 9 开始执行。*指令将 *array_dint[0].0* 卸载到 .UL 位，右移其余的位，然后将 *input_1* 装载到 *array_dint[0].9* 中。其余的位 (10…31) 中的值无效。





示例 **2**： 使能后，BSR 指令将从 *array_dint[1] 中的位 25 开始执行。*该指令将 *array_dint[0].0* 卸载到 .UL 位，右移其余的位，并将 *input_1* 装载到 *array_dint[1].25* 中。其余的位 (*dint_array[1]* 中的 31...26) 中的值无效。注意 *array_dint[1].0* 是如何经过字移位到 *array_dint[0].31* 的。



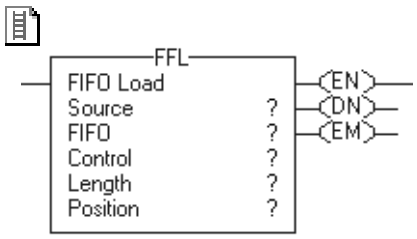


FIFO 装载 (FFL)

FFL 指令用于将源标签值复制到堆栈数组 FIFO。

操作数：

梯形图



操作数	类型	格式	说明
Source	SINT INT DINT REAL 字符串 结构	立即数 标签	要存储在 FIFO 内的数据
	Source 将转换为数组标签的数据类型。较小的整数将通过符号扩展转换为较大的整数。		
FIFO	SINT INT DINT REAL 字符串 结构	数组标签	要修改的 FIFO 指定 FIFO 的第一个元素 不要在下标中使用 CONTROL.POS
Control	CONTROL	标签	操作的控制结构 通常与关联的 FFU 使用相同的 CONTROL
Length	DINT	立即数	FIFO 可同时容纳元素的最多个数
Position	DINT	立即数	FIFO 中的下一个位置，指令将在其中装载数据 初始值通常为 0

如果使用用户自定义的结构作为 Source 或 FIFO 操作数的数据类型，这两个操作数请使用同一种结构。

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	该使能位指示 FFL 指令是否使能。
.DN	BOOL	该完成位置位时指示 FIFO 已满 (.POS = .LEN)。 .DN 位将禁止装载 FIFO，直到 .POS < .LEN 为止。
.EM	BOOL	空位指示 FIFO 为空。如果 .LEN ≤ 0 或 .POS < 0，将置位 .EM 位和 .DN 位。
.LEN	DINT	长度指定 FIFO 可同时容纳元素的最多个数。
.POS	DINT	位置标识 FIFO 中的位置，指令将在其中装载下一个值。

说明： FFL 指令可以和 FFU 指令配合使用，按照先入先出的顺序存储和检索数据。成对使用时，FFL 和 FFU 指令将建立一个异步移位寄存器。

通常， Source 和 FIFO 为同一种数据类型。

使能后， FFL 指令会将 Source 的值装载到 FIFO 中由 .POS 值标识的位置。指令每使能一次，便会装载一个值，直至 FIFO 已满。

重要事项

必须测试并确认指令没有更改不想更改的数据。

FFL 指令针对连续内存操作。有些情况下，该指令会跳过数组将数据装载到标签的其它子元素中。如果长度过大且标签是用户自定义的数据类型，就会发生这种情况。

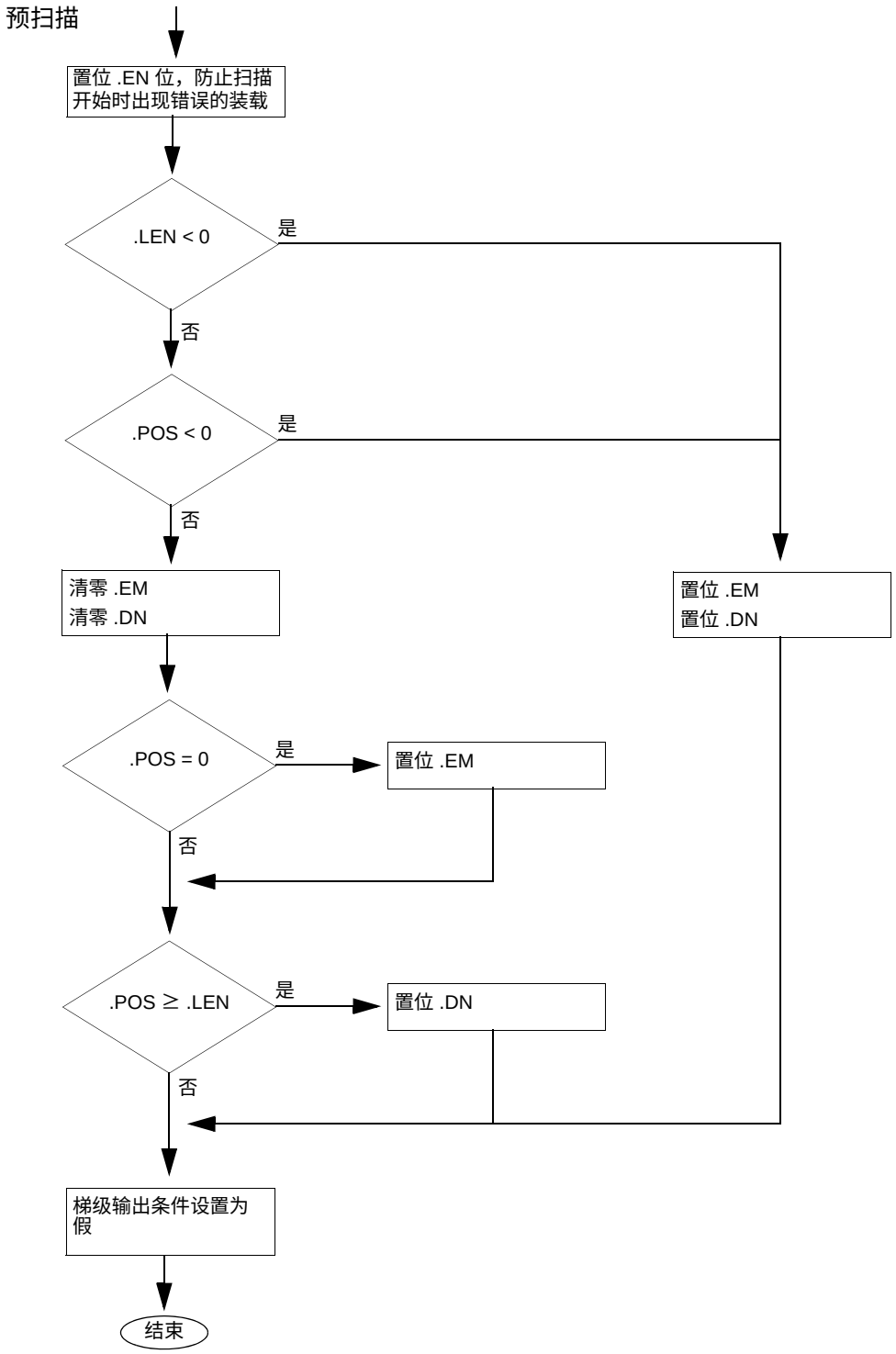
算术状态标志： 不受影响

错误条件：

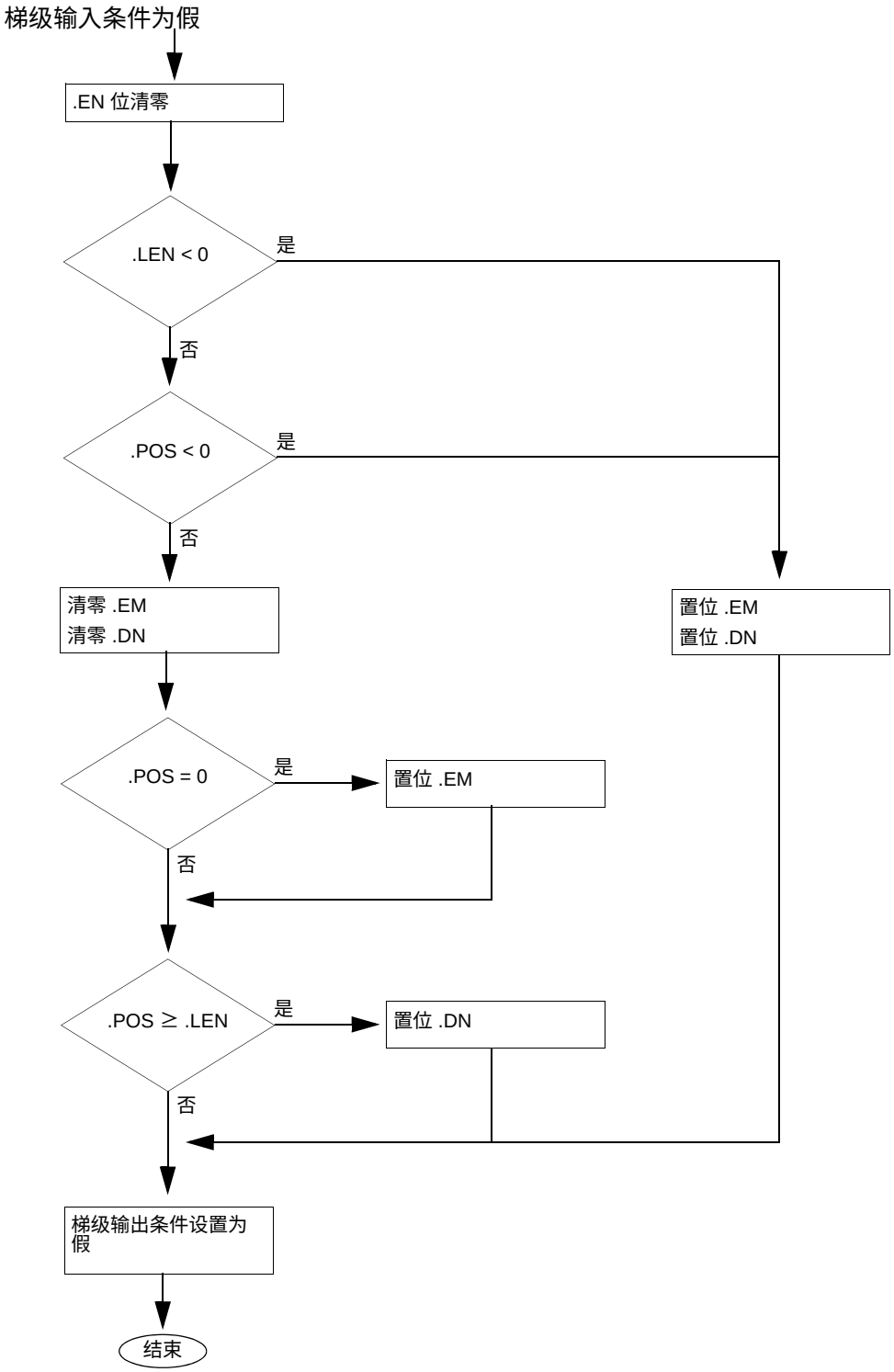
出现严重错误的条件	错误类型	错误代码
(起始元素 + .POS) > FIFO 数组尺寸	4	20

执行：

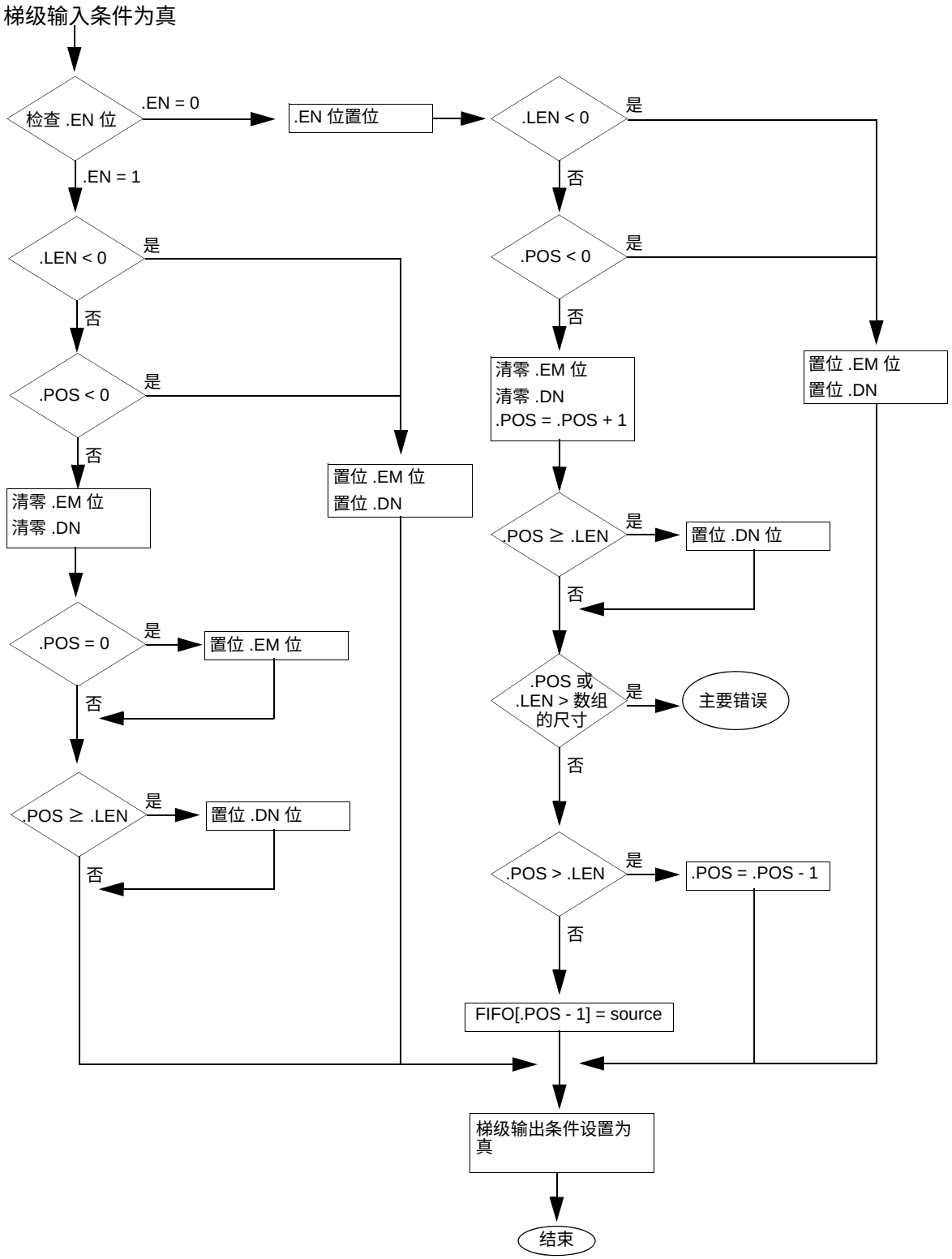
条件	梯形图操作


预扫描
置位 .EN 位，防止扫描开始时出现错误的装载
.LEN < 0
是
否
.POS < 0
是
否
清零 .EM
清零 .DN
置位 .EM
置位 .DN
.POS = 0
是
置位 .EM
否
.POS ≥ .LEN
是
置位 .DN
否
梯级输出条件设置为假
结束

条件	梯形图操作

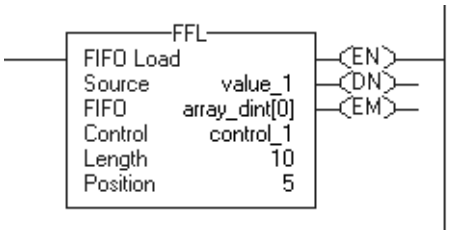

梯级输入条件为假
.EN 位清零
.LEN < 0
是
否
.POS < 0
是
否
清零 .EM
清零 .DN
置位 .EM
置位 .DN
.POS = 0
是
置位 .EM
否
.POS ≥ .LEN
是
置位 .DN
否
梯级输出条件设置为假
结束

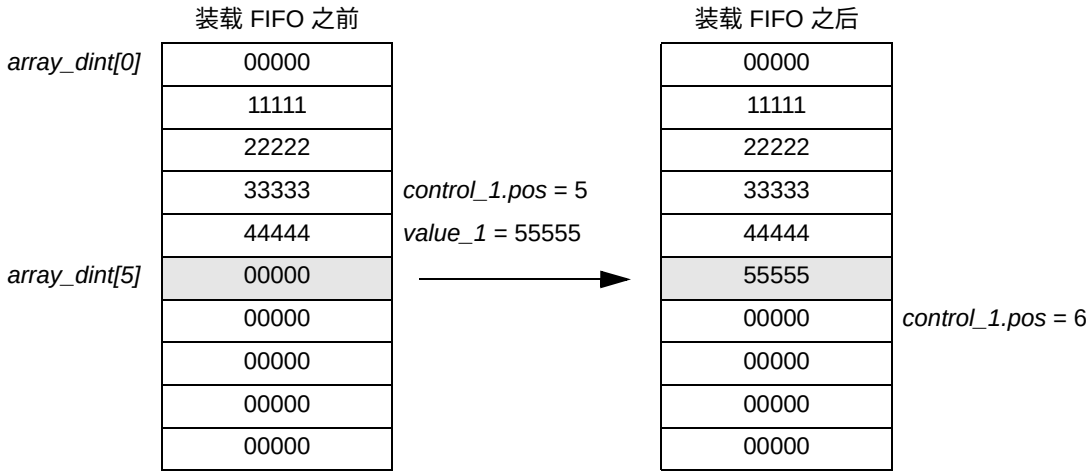
条件	梯形图操作



后扫描	梯级输出条件设置为假。

示例： 使能后， FFL 指令会将 *value_1* 装载到 FIFO 中的下一个位置 (在本示例中是 *array_dint[5]*)。



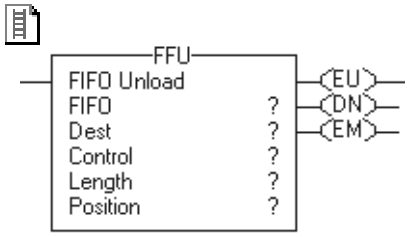


FIFO 卸载 (FFU)

FFU 指令用于卸载堆栈数组 FIFO 中位置 0(第一个位置) 的值，并将该值存储在目标标签中。 FIFO 中其余的数据将下移一个位置。

操作数：

梯形图



操作数	类型	格式	说明
FIFO	SINT INT DINT REAL 字符串 结构	数组标签	要修改的 FIFO 指定 FIFO 的第一个元素 不要在下标中使用 CONTROL.POS
Destination	SINT INT DINT REAL 字符串 结构	标签	从 FIFO 卸载的值
	Destination 的值将转换为目标标签的数据类型。较小的整数将通过符号扩展转换为较大的整数。		
Control	CONTROL	标签	操作的控制结构 通常与关联的 FFL 使用相同的 CONTROL
Length	DINT	立即数	FIFO 可同时容纳元素的最多个数
Position	DINT	立即数	FIFO 中的下一个位置，指令将从其中卸载数据 初始值通常为 0

如果使用用户自定义的结构作为 FIFO 或目标操作数的数据类型，这两个操作数请使用同一种结构。

CONTROL 结构

助记符	数据类型	说明
.EU	BOOL	该使能卸载位指示 FFU 指令是否使能。置位 .EU 位，防止程序扫描开始时出现错误卸载。
.DN	BOOL	该完成位置位时可指示 FIFO 已满 (.POS = .LEN)。
.EM	BOOL	空位指示 FIFO 为空。如果 .LEN ≤ 0 或 .POS < 0，将置位 .EM 位和 .DN 位。
.LEN	DINT	长度指定 FIFO 中元素的最大数量。
.POS	DINT	位置标识装载到 FIFO 中的数据的末尾。

说明： FFU 指令可以和 FFL 指令配合使用，按照先入先出的顺序存储和检索数据。

使能后， FFU 指令将从 FIFO 的第一个元素中卸载数据，并将该值放在 Destination 中。指令每使能一次，便会卸载一个值，直至 FIFO 清空为止。如果 FIFO 为空， FFU 将 0 返回到 Destination。

重要事项

必须测试并确认指令没有更改不想更改的数据。

FFU 指令针对连续内存操作。有些情况下，该指令会从标签的其它子元素中卸载数据。如果长度过大且标签是用户自定义的数据类型，就会发生这种情况。

算术状态标志： 不受影响

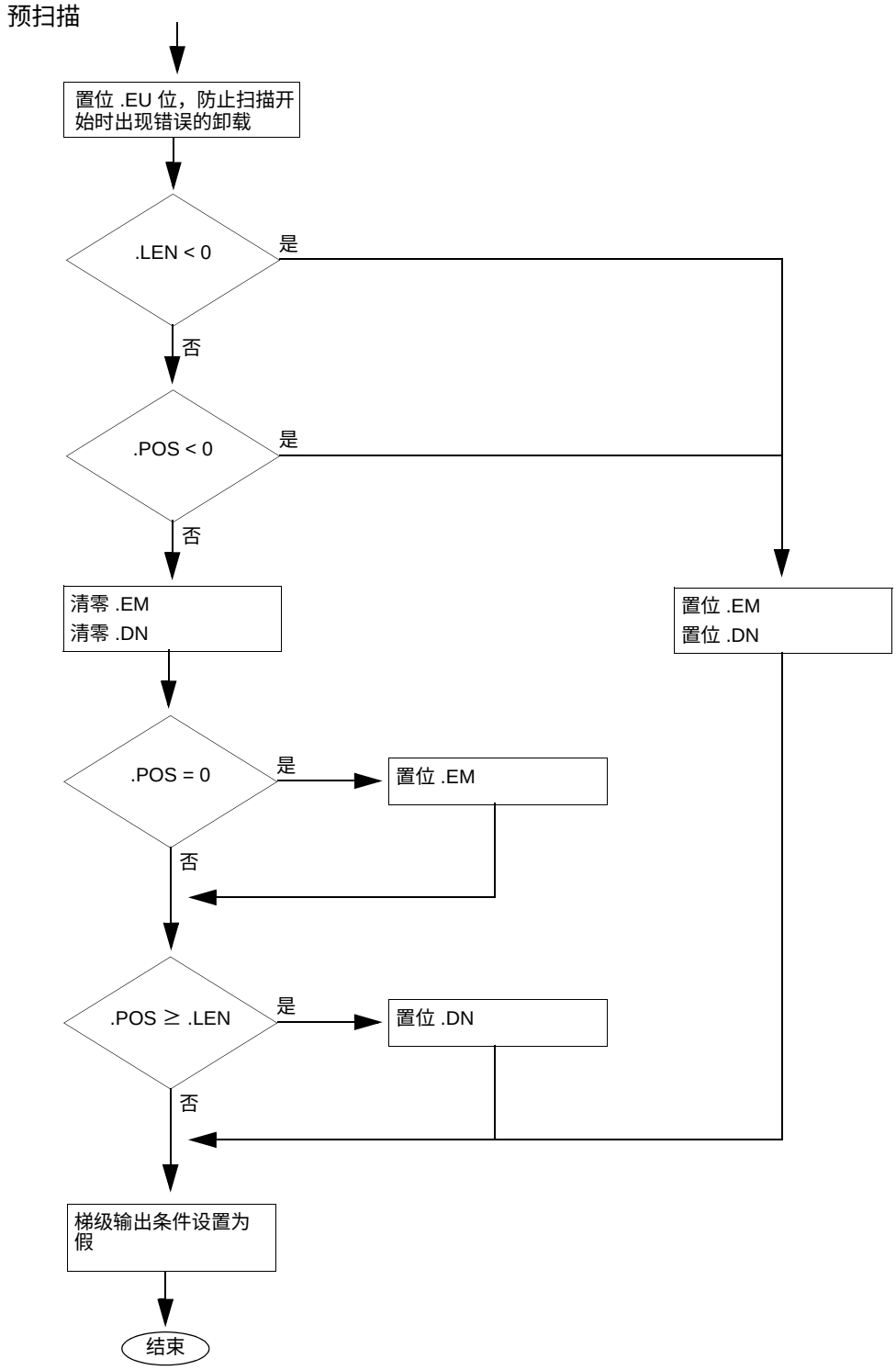
错误条件：

出现严重错误的条件	错误类型	错误代码
长度 > FIFO 数组尺寸	4	20

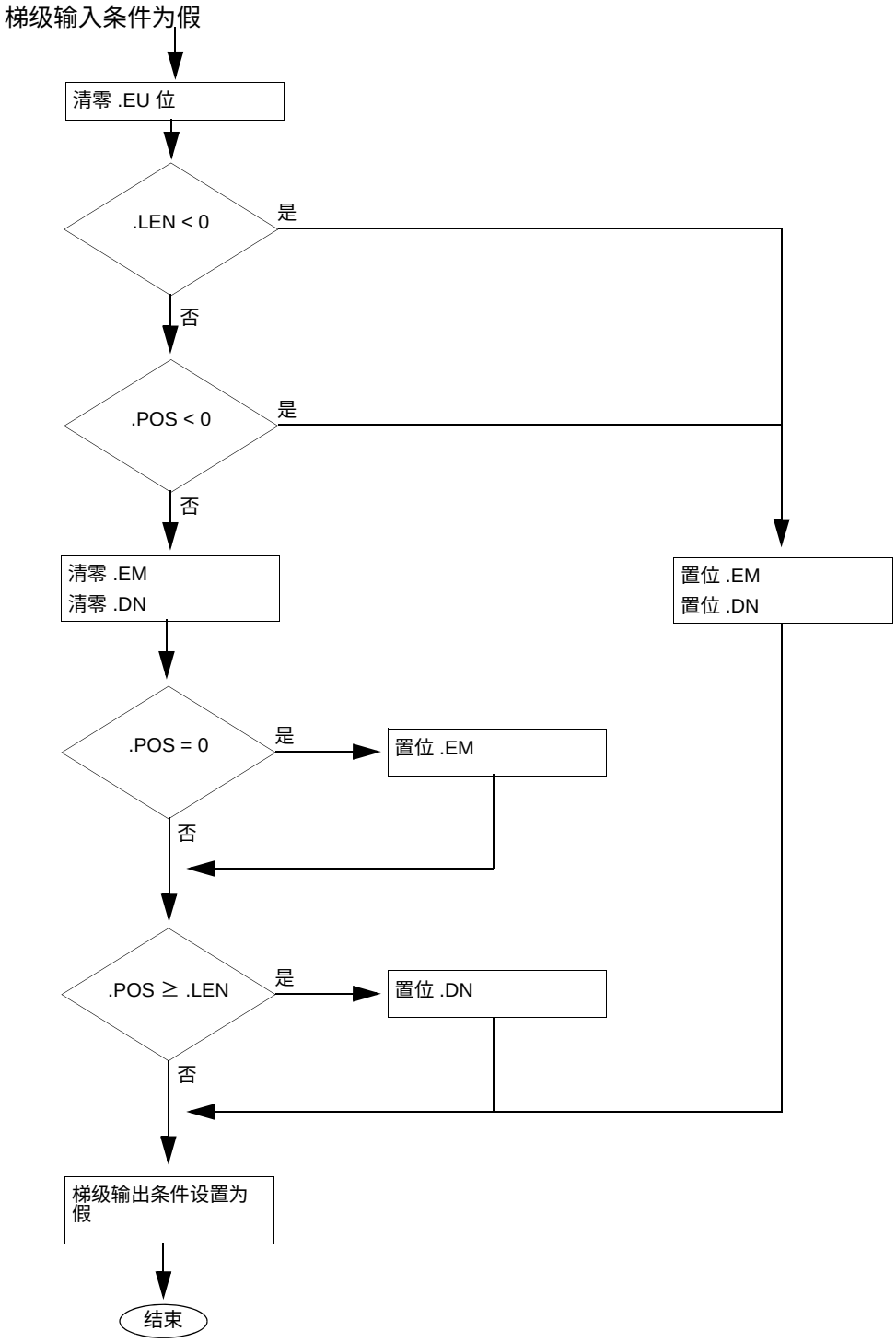
执行：

条件	梯形图操作

预扫描

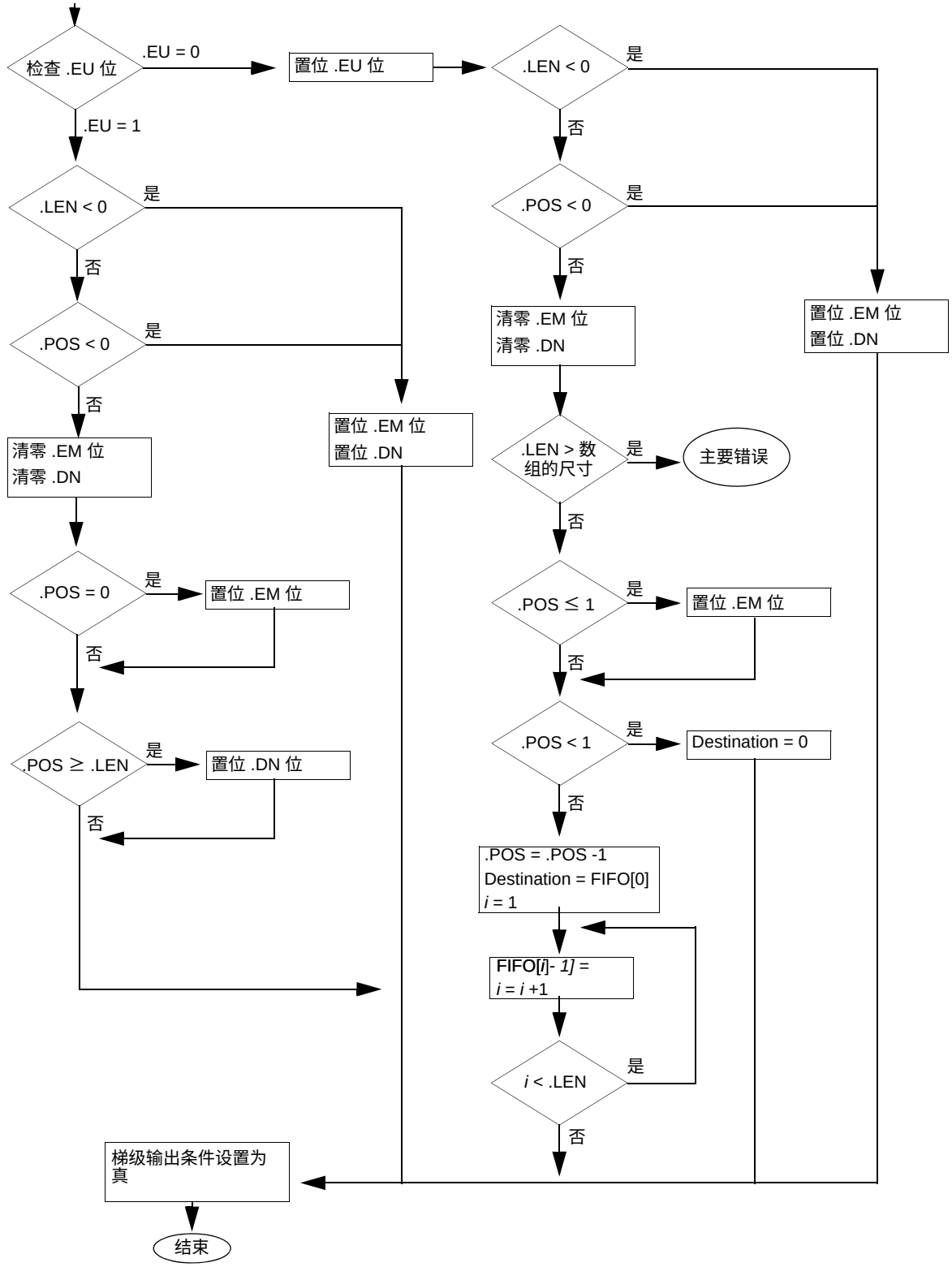


条件	梯形图操作


梯级输入条件为假
清零 .EU 位
.LEN < 0
是
否
.POS < 0
是
否
清零 .EM
清零 .DN
置位 .EM
置位 .DN
.POS = 0
是
置位 .EM
否
.POS ≥ .LEN
是
置位 .DN
否
梯级输出条件设置为假
结束

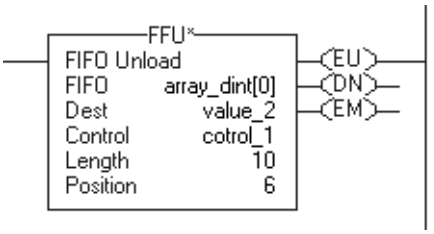
条件	梯形图操作

梯级输入条件为真



后扫描	梯级输出条件设置为假。

示例： 使能后， FFU 指令将 *array_dint[0]* 卸载到 *value_2* 中，并对 *array_dint* 中其余的元素进行移位。



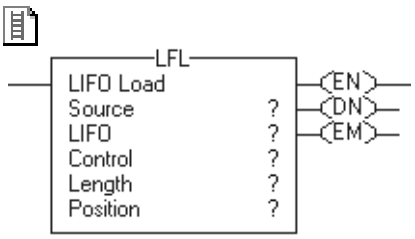
	卸载 FIFO 之前		卸载 FIFO 之后	
array_dint[0]	00000	→	11111	
	11111		22222	
	22222		33333	
	33333		44444	
	44444		55555	
array_dint[5]	55555		00000	*control_1.pos* = 5
	00000	*control_1.pos* = 6	00000	*value_2* = 00000
	00000		00000	
	00000		00000	
	00000		00000	

LIFO 装载 (LFL)

LFL 指令用于将源的值复制到堆栈数组 LIFO。

操作数：

梯形图



操作数	类型	格式	说明
Source	**SINT** **INT** **DINT** **REAL** 字符串 结构	立即数 标签	要存储到 LIFO 中的数据
	Source 将转换为数组标签的数据类型。较小的整数将通过符号扩展转换为较大的整数。		
LIFO	**SINT** **INT** **DINT** **REAL** 字符串 结构	数组标签	要修改的 LIFO 指定 LIFO 的第一个元素 不要在下标中使用 CONTROL.POS
Control	CONTROL	标签	操作的控制结构 通常与关联的 LFU 使用相同的 CONTROL
Length	DINT	立即数	LIFO 可同时容纳元素的最多个数
Position	DINT	立即数	LIFO 中的下一个位置，指令将在其中装载数据 初始值通常为 0

如果使用用户自定义的结构作为源或堆栈数组 LIFO 操作数的数据类型，这两个操作数请使用同一种结构。

CONTROL 结构

助记符	数据类型	说明：
.EN	BOOL	该使能位指示 LFL 指令是否使能。
.DN	BOOL	该完成位置位时指示 LFL 已满 (.POS = .LEN)。.DN 位将禁止装载 LIFO，直到 .POS < .LEN 为止。
.EM	BOOL	空位指示 LIFO 为空。如果 .LEN ≤ 0 或 .POS < 0，将置位 .EM 位和 .DN 位。
.LEN	DINT	长度指定 LIFO 可同时容纳元素的最多个数。
.POS	DINT	位置标识 LIFO 中的位置，指令将在其中装载下一个值。

说明： LFL 指令可以和 LFU 指令配合使用，按照后入先出的顺序存储和检索数据。成对使用时， LFL 和 LFU 指令将建立一个异步移位寄存器。

通常，源和堆栈数组 LIFO 为同一种数据类型。

使能后， LFL 指令会将源的值装载到堆栈数组 LIFO 中由 .POS 值标识的位置。指令每使能一次，便会装载一个值，直至 LIFO 已满。

重要事项

必须测试并确认指令没有更改不想更改的数据。

LFL 指令针对连续内存操作。有些情况下，该指令会跳过数组将数据装载到标签的其它子元素中。如果长度过大且标签是用户自定义的数据类型，就会发生这种情况。

算术状态标志： 不受影响

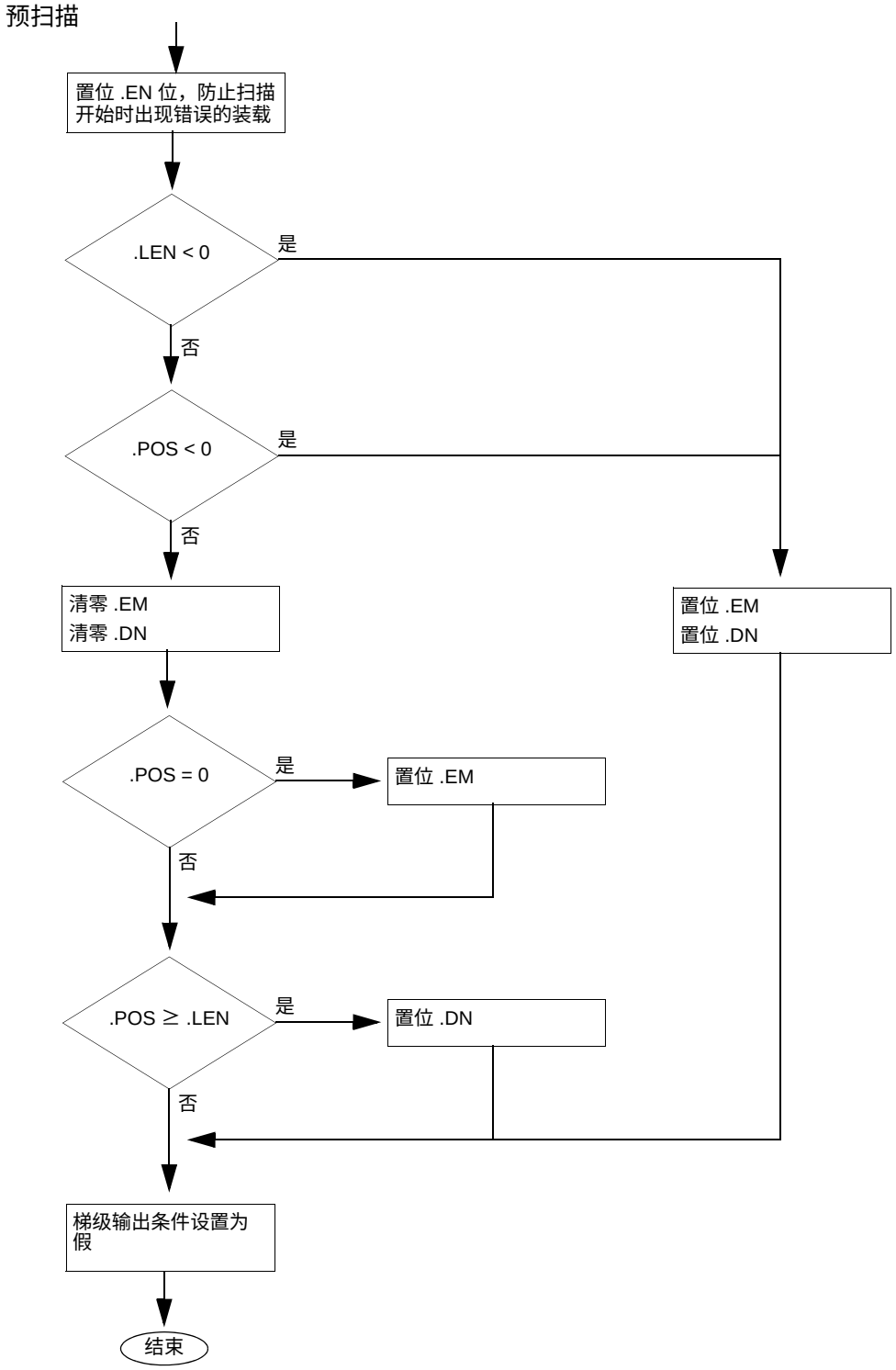
错误条件：

出现严重错误的条件	错误类型	错误代码
(起始元素 + .POS) > LIFO 数组尺寸	4	20

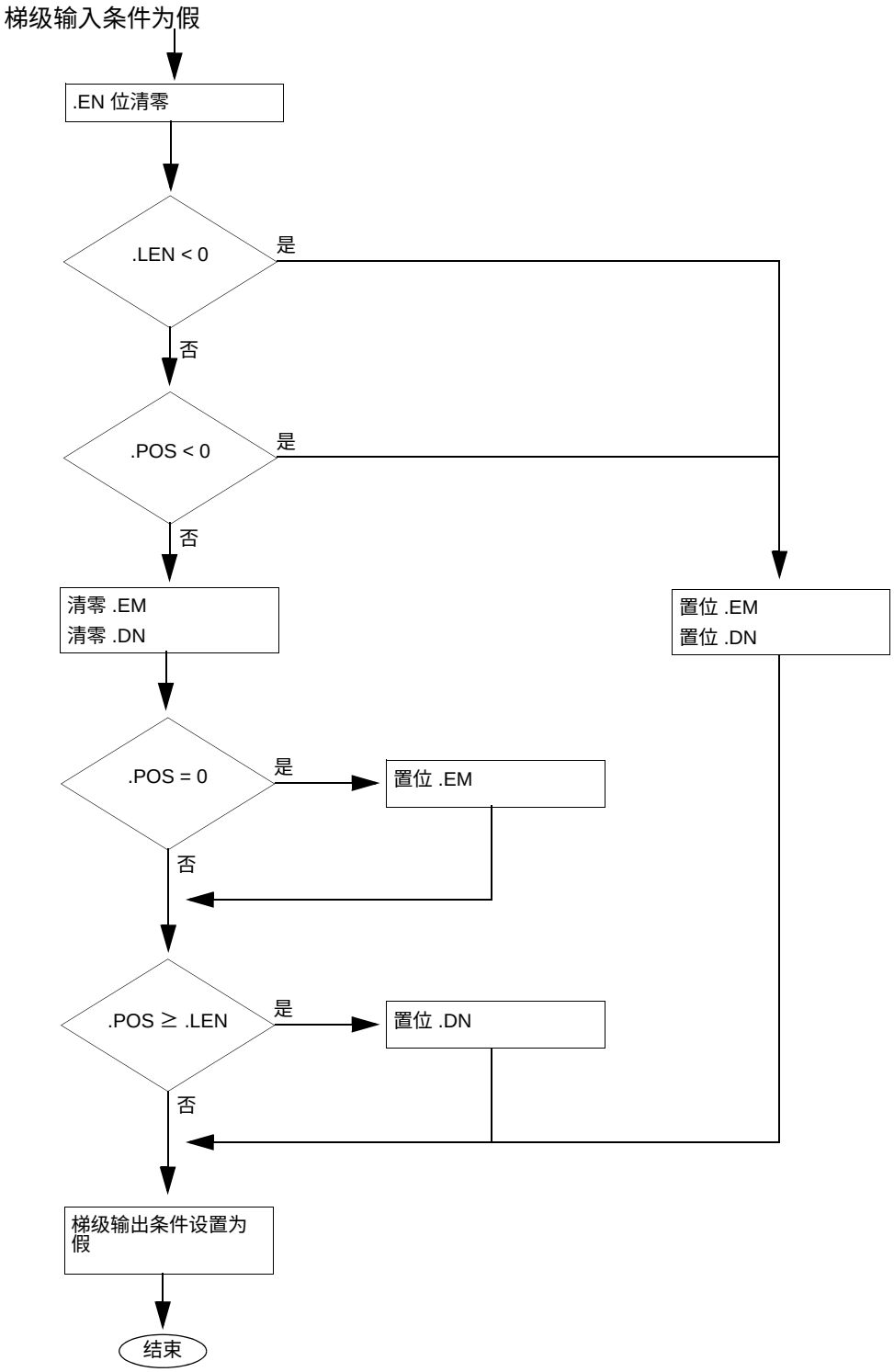
执行：

条件：	梯形图操作

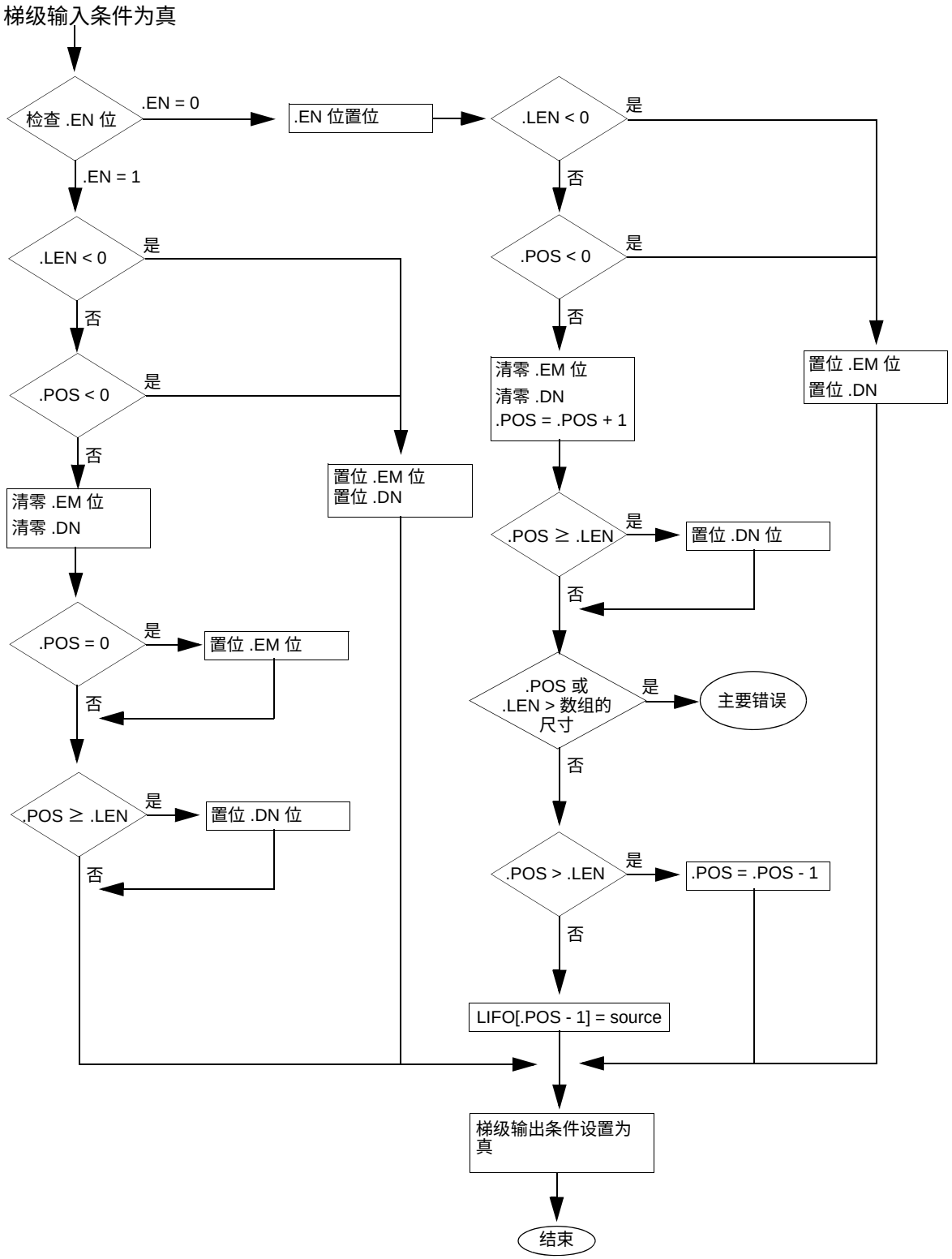
预扫描



条件：	梯形图操作

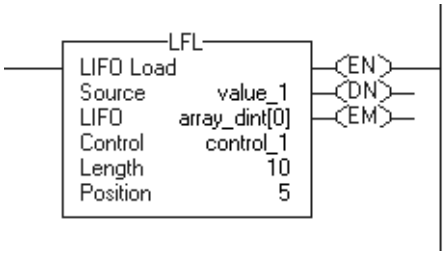

梯级输入条件为假
.EN 位清零
.LEN < 0
是
否
.POS < 0
是
否
清零 .EM
清零 .DN
置位 .EM
置位 .DN
.POS = 0
是
置位 .EM
否
.POS ≥ .LEN
是
置位 .DN
否
梯级输出条件设置为假
结束

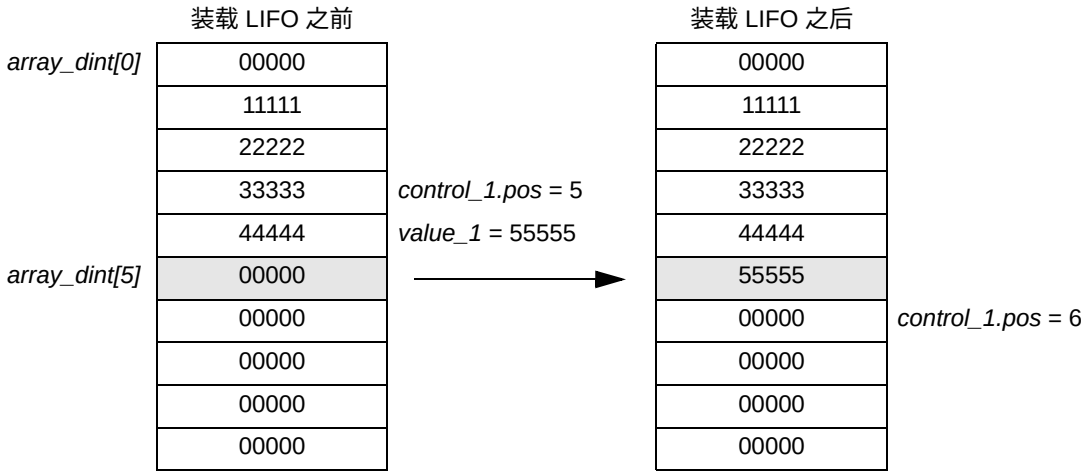
条件：	梯形图操作



后扫描	梯级输出条件设置为假。

示例： 使能后， LFL 指令会将 *value_1* 装载到 LIFO 中的下一个位置 (在本示例中是 *array_dint[5]*)。



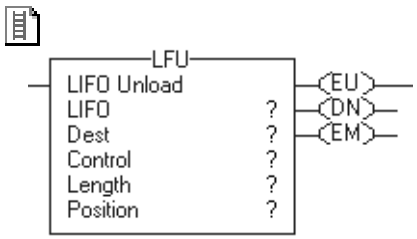


LIFO 卸载 (LFU)

LFU 指令用于卸载堆栈文件 LIFO 的 .POS 处的值并在该处置零。

操作数：

梯形图



操作数	类型	格式	说明
LIFO	SINT INT DINT REAL 字符串 结构	数组标签	要修改的 LIFO 指定 LIFO 的第一个元素 不要在下标中用 CONTROL.POS
Destination	SINT INT DINT REAL 字符串 结构	标签	从 LIFO 卸载的值
	Destination 的值将转换为目标标签的数据类型。较小的整数将通过符号扩展转换为较大的整数。		
Control	CONTROL	标签	操作的控制结构 通常与关联的 LFL 使用相同的 CONTROL
Length	DINT	立即数	LIFO 可同时容纳元素的最多个数
Position	DINT	立即数	LIFO 中的下一个位置，指令将从其中卸载数据 初始值通常为 0

如果使用用户自定义的结构作为堆栈数组 LIFO 或目标操作数的数据类型，这两个操作数请使用同一种结构。

CONTROL 结构

助记符	数据类型：	说明
.EU	BOOL	该使能卸载位指示 LFU 指令是否使能。置位 .EU 位，防止程序扫描开始时出现错误卸载。
.DN	BOOL	该完成位置位时可指示 LFL 已满 (.POS = .LEN)。
.EM	BOOL	空位指示 LIFO 为空。如果 .LEN ≤ 0 或 .POS < 0，将置位 .EM 位和 .DN 位。
.LEN	DINT	长度指定 LIFO 可同时容纳元素的最多个数。
.POS	DINT	位置标识装载到 LIFO 中的数据的末尾。

说明： LFU 指令可以和 LFL 指令配合使用，按照后入先出的顺序存储和检索数据。

使能后， LFU 指令将卸载堆栈数组 LIFO 的 .POS 处的值，并将该值放到目标标签中。每次使能指令时，指令都会卸载一个值，并用 0 替换这个值，直至 LIFO 清空为止。如果 LIFO 为空， LFU 将 0 返回到目标标签。

重要事项

必须测试并确认指令没有更改不想更改的数据。

LFU 指令针对连续内存操作。有些情况下，该指令会从标签的其它子元素中卸载数据。如果长度过大且标签是用户自定义的数据类型，就会发生这种情况。

算术状态标志： 不受影响

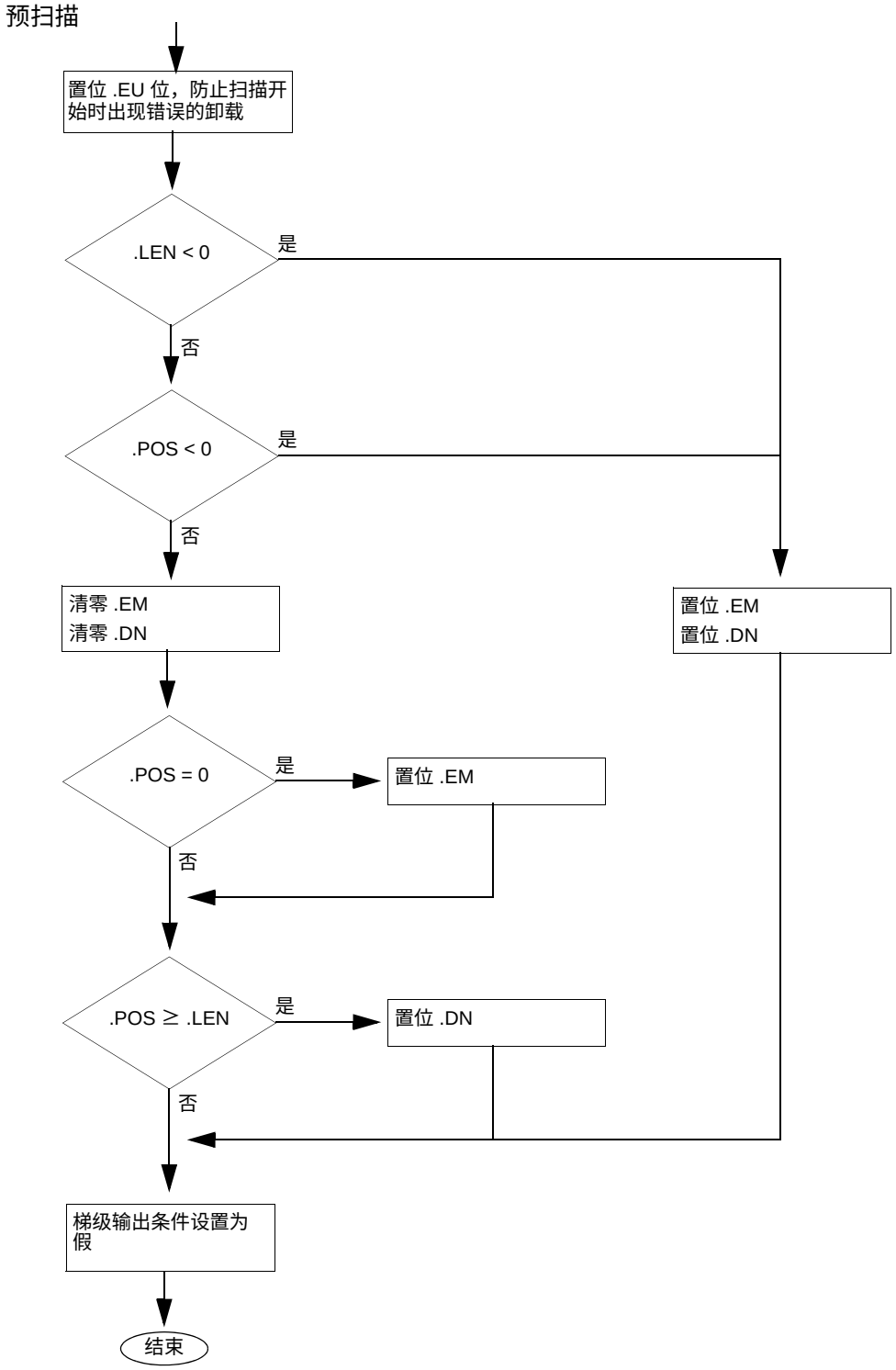
错误条件：

出现严重错误的条件	错误类型	错误代码
长度 > LIFO 数组尺寸	4	20

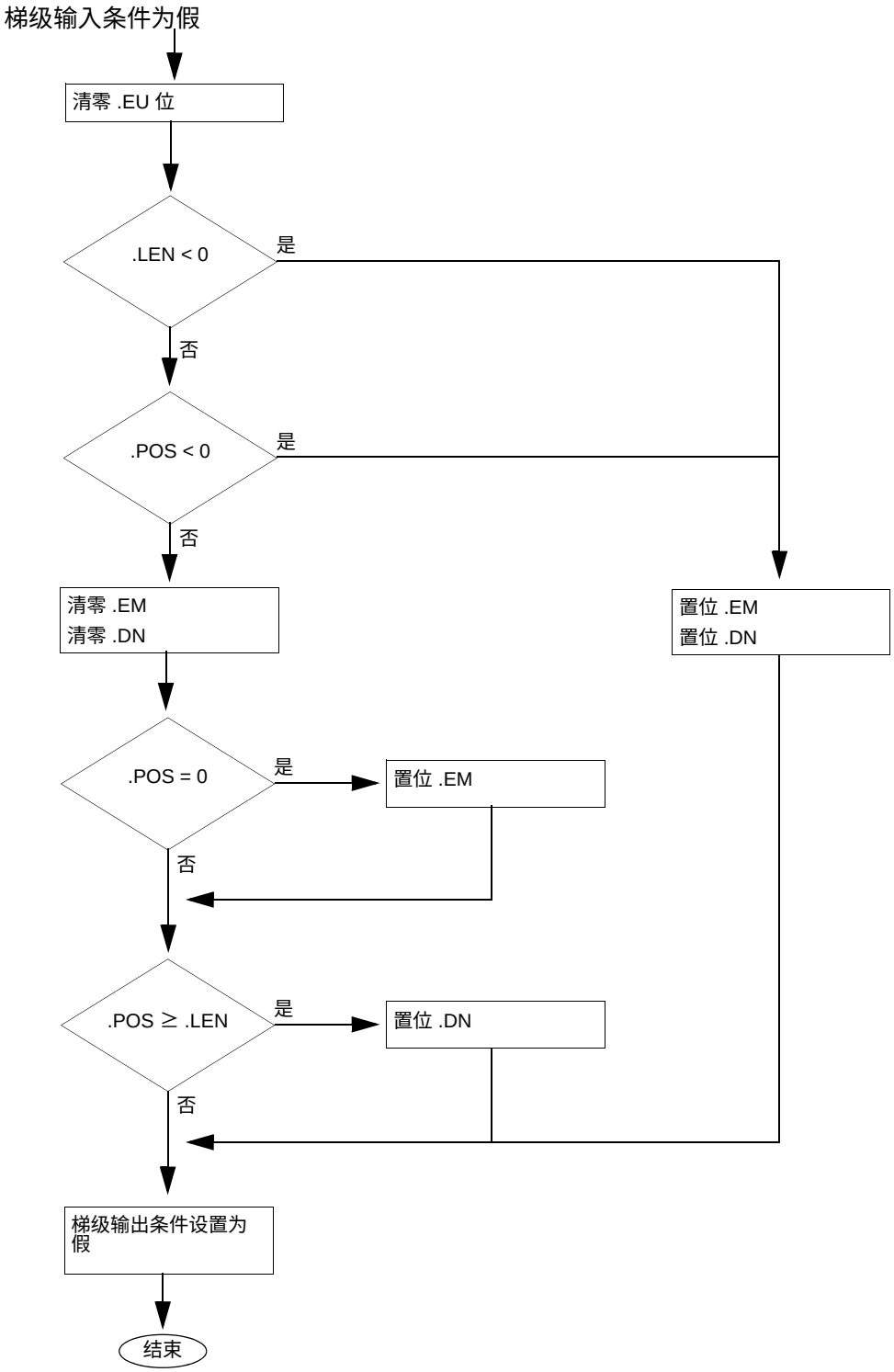
执行：

条件	梯形图操作

预扫描

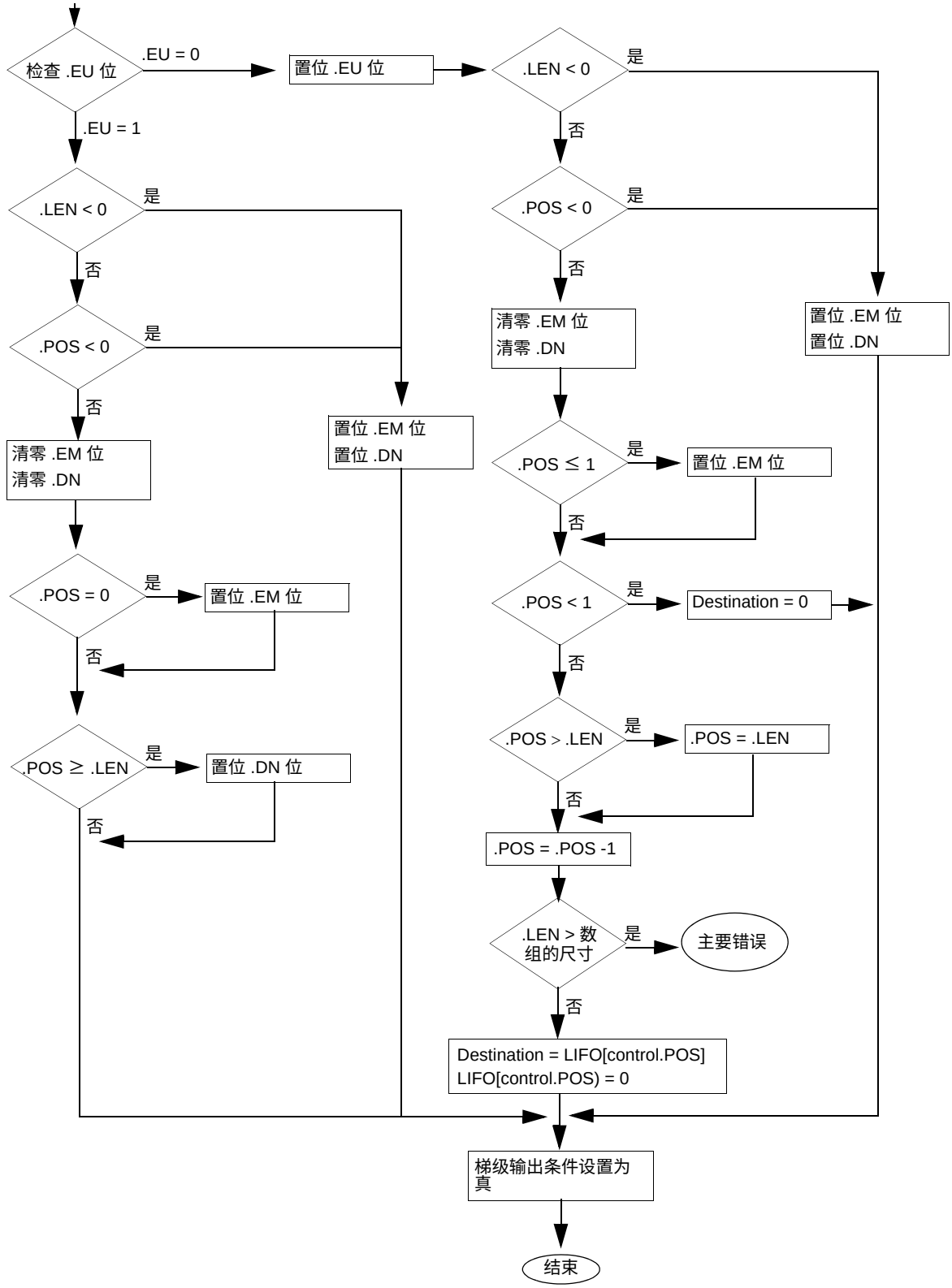


条件	梯形图操作


梯级输入条件为假
清零 .EU 位
.LEN < 0
是
否
.POS < 0
是
否
清零 .EM
清零 .DN
置位 .EM
置位 .DN
.POS = 0
是
置位 .EM
否
.POS ≥ .LEN
是
置位 .DN
否
梯级输出条件设置为假
结束

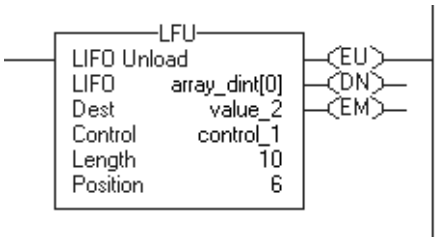
条件	梯形图操作

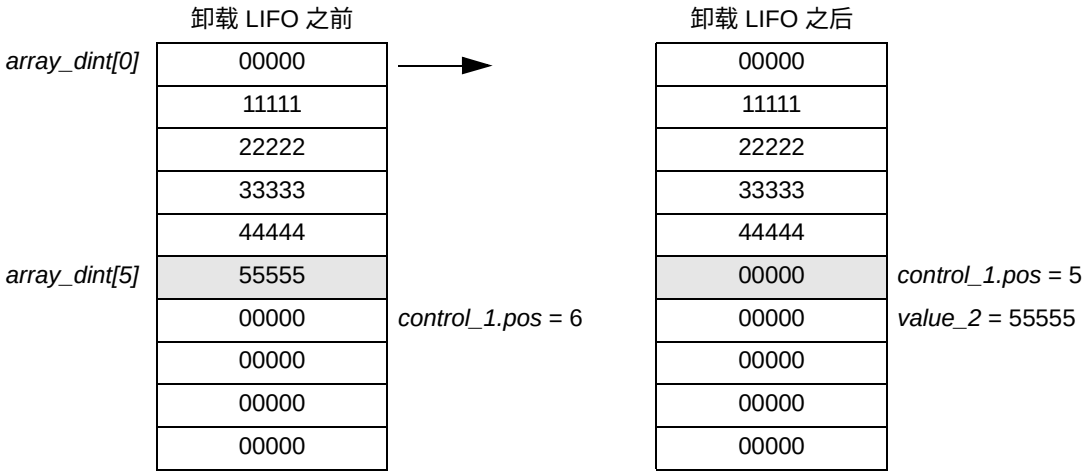
梯级输入条件为真



后扫描	梯级输出条件设置为假。

示例： 使能后， LFU 指令将 *array_dint[5]* 卸载到 *value_2* 中。





注：

顺序器指令
(SQI、 SQO、 SQL)

简介

不执行任何操作。顺序器指令监视规律性和重复性的操作。

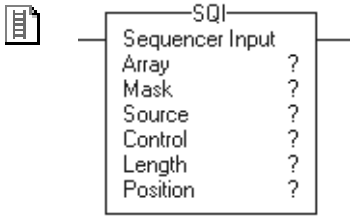
如果要	使用以下指令	在以下语言中可用	页码
检测步完成的时间	SQI	梯形图	428
为下一步设置输出条件	SQO	梯形图	432
装载参考条件到顺序器数组	SQL	梯形图	436

对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

顺序器输入 **(SQI)**

SQI 指令检测 SQO ／ SQI 指令顺序对中的步完成时间。

操作数：



梯形图

操作数	类型	格式	说明
数组 (Array)	DINT	数组标签	顺序器数组 指定顺序器数组的第一个元素 不要在下标中使用 CONTROL.POS
屏蔽码 (Mask)	SINT INT **DINT**	标签 立即数	要阻止或传递的位
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
源 (Source)	SINT INT DINT	标签	顺序器数组的输入数据
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
控制 (Control)	CONTROL	标签	操作的控制结构 通常使用与 SQO 和 SQL 指令相同的 CONTROL
长度 (Length)	DINT	立即数	数组 (顺序器表) 中比较的元素个数
位置 (Position)	DINT	立即数	数组中的当前位置 初始值通常为 0

CONTROL 结构

助记符	数据类型	说明
.ER	BOOL	当 .LEN ≤ 0、 .POS < 0 或 .POS > .LEN 时，错误位置位。
.LEN	DINT	长度指定顺序器数组中的步数。
.POS	DINT	位置标识指令当前比较的元素。

说明： 使能后， SQI 指令通过屏蔽比较源元素与数组元素是否相等。

通常使用与 SQO 和 SQL 指令相同的 CONTROL 结构。

SQI 指令对连续内存进行操作。

键入立即数屏蔽码值

键入屏蔽码时，编程软件默认为十进制值。如果想使用另一种格式键入屏蔽码，请在值的前面加上正确的前缀。

前缀	说明
16#	十六进制 例如， 16#0F0F
8#	八进制 例如， 8#16
2#	二进制 例如， 2#00110011

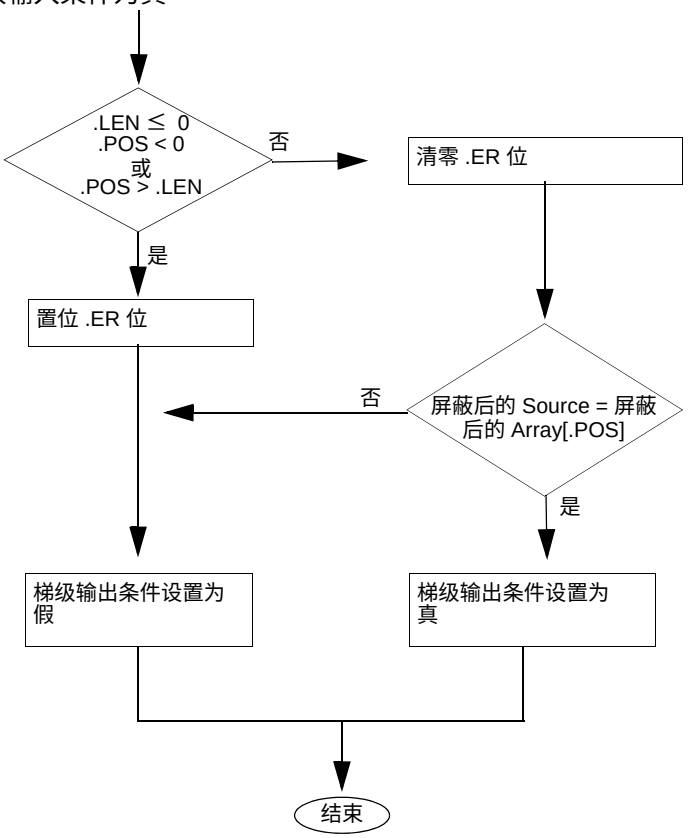
算术状态标志： 不受影响

错误条件： 无

执行：

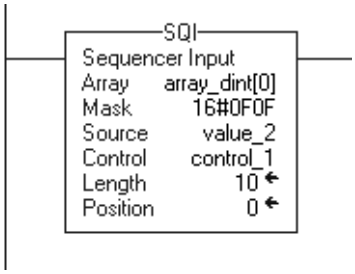
条件：	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。

梯级输入条件为真



后扫描	梯级输出条件设置为假。

示例： 使能后， SQI 指令通过屏蔽传递 *value_2*，并确定是否等于 *array_dint* 中的当前元素。通过屏蔽后的比较为真，因此梯级输出条件变成真。



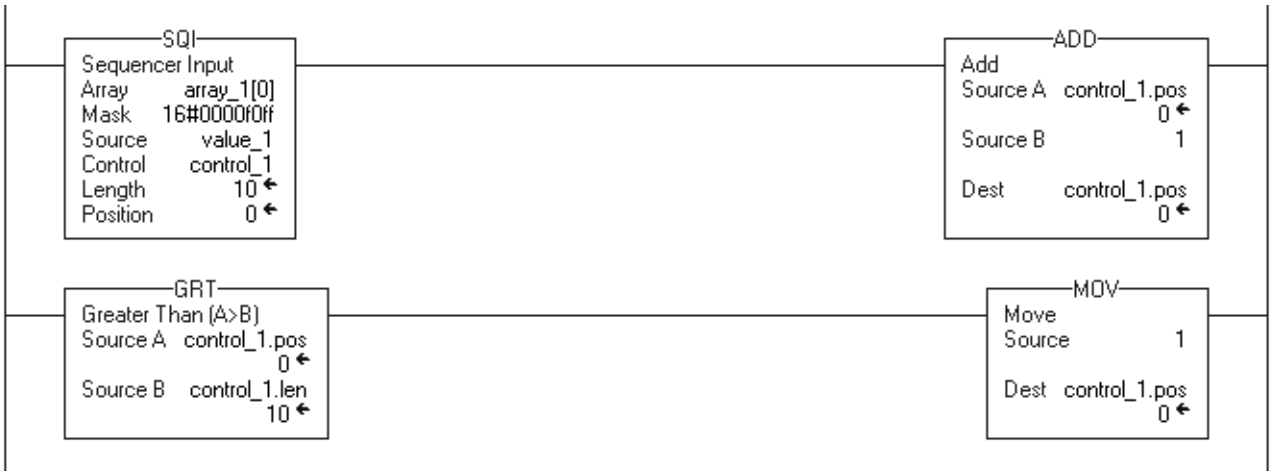
SQI 操作数	示例值 (以二进制显示的 DINT)
源 (Source)	**xxxxxxxx xxxxxxxx xxxx0101 xxxx1010**
屏蔽码 (Mask)	**00000000 00000000 00001111 00001111**
数组 (Array)	**xxxxxxxx xxxxxxxx xxxx0101 xxxx1010**

屏蔽码中的 0 表示不比较相关位 (本示例中标有 xxxx)。

只使用 SQI 而不使用 SQO

如果只使用 SQI 指令而不与 SQO 指令成对使用，则必须在外部增加顺序器数组的位置值。

SQI 指令比较源值。 ADD 指令增加数组位置值。 GRT 确定检查的顺序器数组是否在有效的位置值范围。从头到尾完成顺序器数组所有步一轮后， MOV 指令复位位置值。

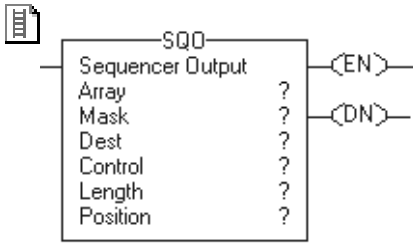


顺序器输出 (SQO)

SQO 指令为一对 SQO/SQI 指令的下一步设置输出条件。

操作数：

梯形图



操作数	类型	格式	说明
数组 (Array)	DINT	数组标签	顺序器数组 指定顺序器数组的第一个元素 不要在下标中使用 CONTROL.POS
屏蔽码 (Mask)	SINT INT DINT	标签 立即数	要阻止或传递的位
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
目标 (Destination)	DINT	标签	顺序器数组的输出数据
控制 (Control)	CONTROL	标签	操作的控制结构 通常使用与 SQI 和 SQL 指令相同的 CONTROL
长度 (Length)	DINT	立即数	数组 (顺序器表) 中要输出的元素个数
位置 (Position)	DINT	立即数	数组中的当前位置 初始值通常为 0

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 SQO 指令是否使能。
.DN	BOOL	当所有指定的元素都移动到 Destination 中后，完成位置位。
.ER	BOOL	当 .LEN ≤ 0、 .POS < 0 或 .POS > .LEN 时，错误位置位。
.LEN	DINT	长度指定顺序器数组的步数。
.POS	DINT	位置标识控制器当前处理的元素。

说明： 使能后， SQO 指令递增位置值，通过屏蔽移动该位置的数据，并将结果存储在目标标签中。如果 .POS > .LEN，该指令将返回到顺序器数组的开头，并从 .POS = 1 继续执行。

通常使用与 SQI 和 SQL 指令相同的 CONTROL 结构。

SQO 指令对连续内存进行操作。

键入立即数屏蔽码值

键入屏蔽码时，编程软件将默认为十进制值。如果想使用另一种格式键入屏蔽码，请在值的前面加上正确的前缀。

前缀	说明
16#	十六进制 例如， 16#0F0F
8#	八进制 例如， 8#16
2#	二进制 例如， 2#00110011

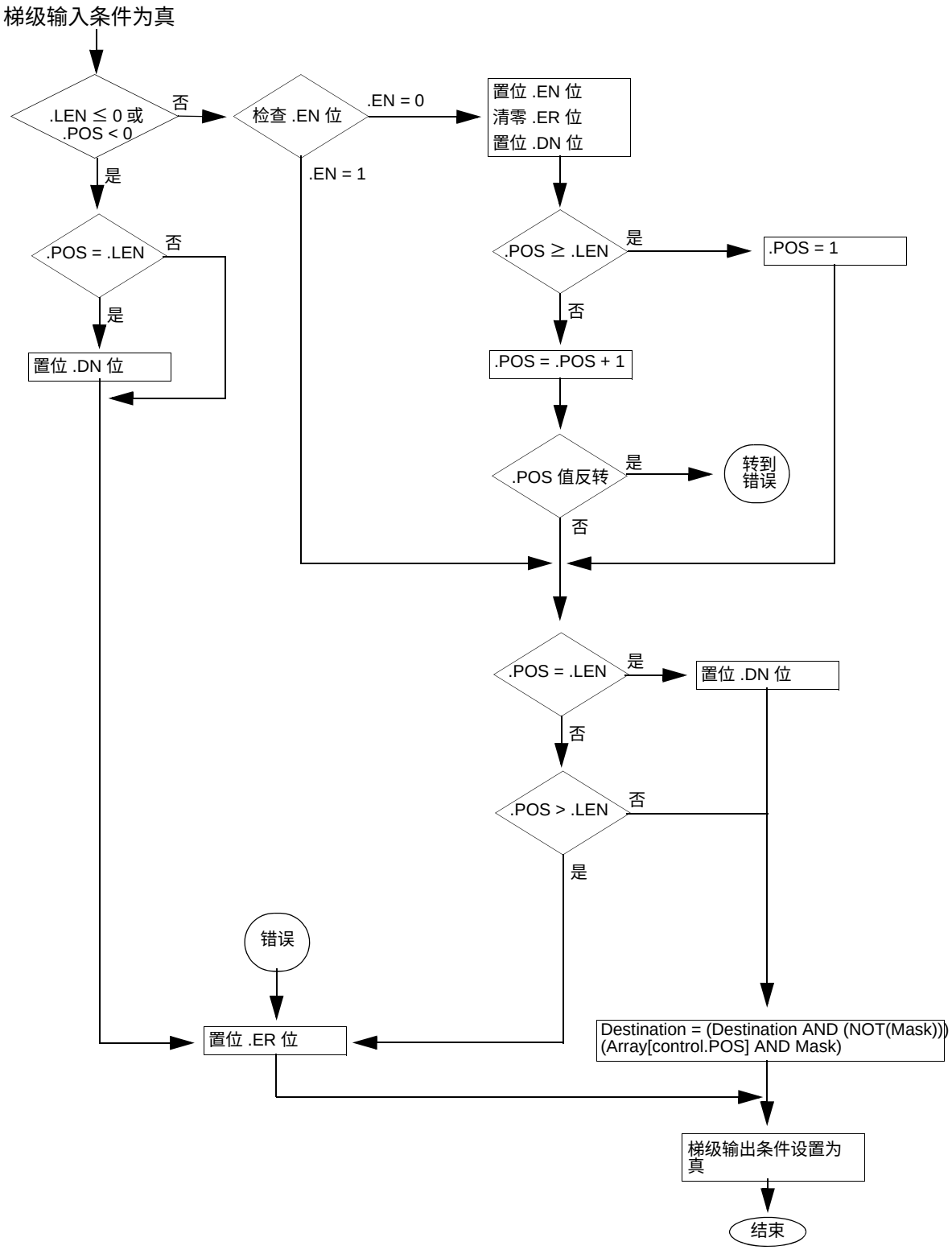
算术状态标志　不受影响

错误条件： 无

执行：

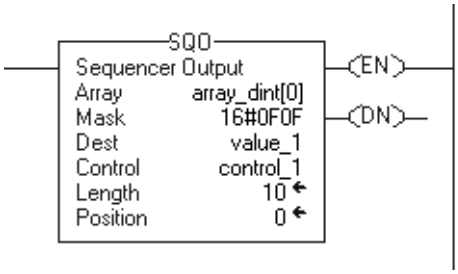
条件	梯形图操作
预扫描	置位 .EN 位，防止在程序扫描开始时发生错误装载。 梯级输出条件设置为假。
梯级输入条件为假	清零 .EN 位。 梯级输出条件设置为假。

条件	梯形图操作



后扫描	梯级输出条件设置为假。

示例： 使能后， SQO 指令递增位置值，通过屏蔽传递 *array_dint* 中该位置上的数据，并且将结果存储在 *value_1* 中。



SQO 操作数	示例值 (使用以二进制显示的 INT)
数组 (Array)	**xxxxxxxx xxxxxxxx xxxx0101 xxxx1010**
屏蔽码 (Mask)	**00000000 00000000 00001111 00001111**
目标 (Destination)	**xxxxxxxx xxxxxxxx xxxx0101 xxxx1010**

屏蔽码中的 0 表示不比较相关位 (本示例中标有 xxxx)。

配合使用 **SQI** 与 **SQO**

如果成对使用 SQI 指令与 SQO 指令，请确保这两条指令使用相同的 Control、 Length 和 Position 值。

复位 **SQO** 的 **Position**

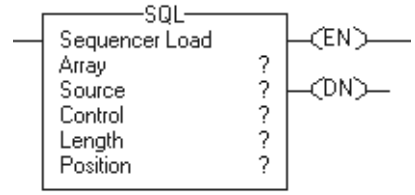
控制器每次从程序模式切换到运行模式时， SQO 指令都要清除 (初始化).POS 值。要将 .POS 复位为初始值 (.POS = 0)，请使用 RES 指令来清除位置值。下面的示例使用首次扫描位的状态来清除 .POS 值。

顺序器装载 (SQL)

SQL 指令将参考条件装载到顺序器数组。

操作数：

梯形图



操作数	类型	格式	说明
数组 (Array)	DINT	数组标签	顺序器数组 指定顺序器数组的第一个元素 不要在下标中使用 CONTROL.POS
源 (Source)	SINT INT **DINT**	标签 立即数	要装载到顺序器数组的输入数据
	SINT 或 INT 标签将通过符号扩展转换为 DINT 值。		
控制 (Control)	CONTROL	标签	操作的控制结构 通常使用与 SQI 和 SQO 指令相同的 CONTROL
长度 (Length)	DINT	立即数	要装载的数组 (顺序器表) 中的元素个数
位置 (Position)	DINT	立即数	数组中的当前位置 初始值通常为 0

CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 SQL 指令是否使能。
.DN	BOOL	当所有指定的元素都装载到数组中后，完成位置位。
.ER	BOOL	当 .LEN ≤ 0、 .POS < 0 或 .POS > .LEN 时，错误位置位。
.LEN	DINT	长度指定顺序器数组中的步数。
.POS	DINT	位置标识控制器当前处理的元素。

说明： 使能后，SQL 指令递增到顺序器数组中的下一位置，并将源值装载到该位置。如果 .DN 位置位或 .POS ≥ .LEN，指令将设置 .POS=1。

通常使用与 SQI 和 SQO 指令相同的 CONTROL 结构。

重要事项

必须测试并确认指令不会更改您不希望更改的数据。

SQL 指令对连续内存进行操作。在某些情况下，该指令会越过数组将数据装载到标签的其它子元素。如果 length 过大且标签是用户自定义的数据类型，就会发生这种情况。

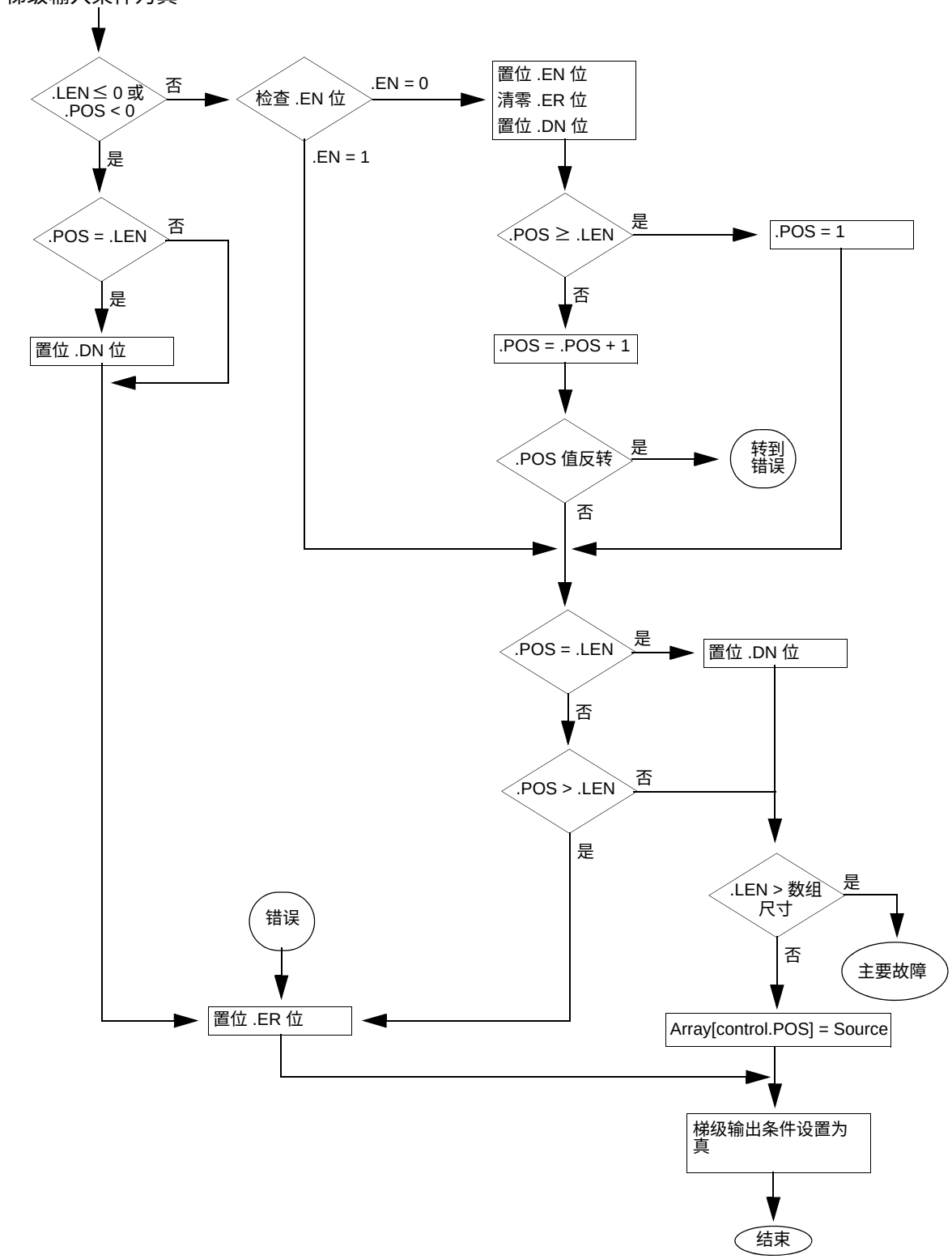
算术状态标志： 不受影响

错误条件：

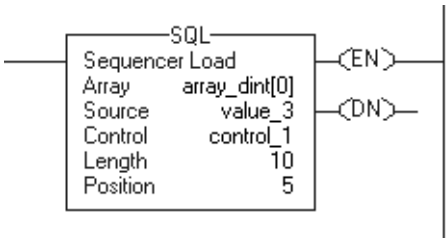
出现主要故障的条件	故障类型	故障代码
Length > 数组大小	4	20

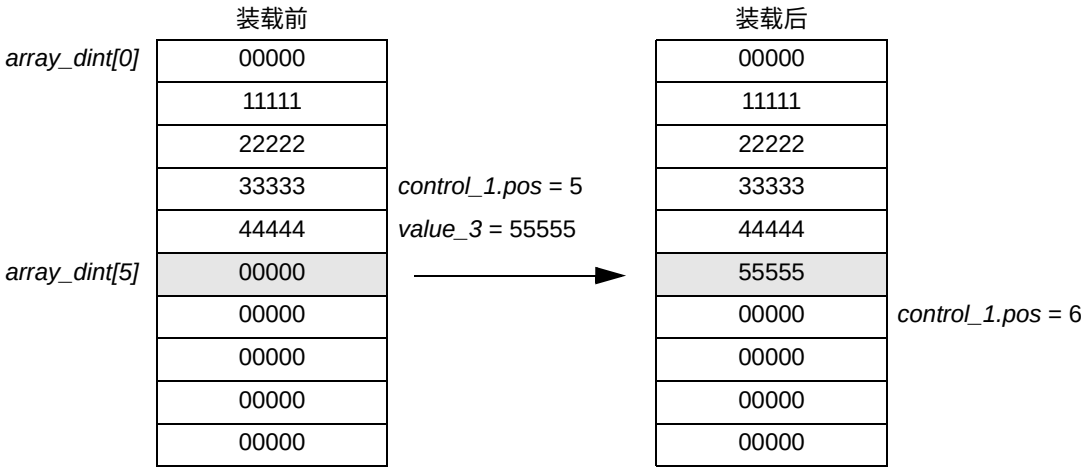
执行：

条件	梯形图操作
预扫描	置位 .EN 位，防止在程序扫描开始时发生错误装载。 梯级输出条件设置为假。
梯级输入条件为假	清零 .EN 位。 梯级输出条件设置为假。

条件	梯形图操作
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

示例： 使能后，SQL 指令将 *value_3* 装载到顺序器数组中的下一位置，在本示例中是 *array_dint[5]*。





注：

程序控制指令
(JMP、 LBL、 JSR、 RET、 SBR、 JXR、 TND、 MCR、 UID、 UIE、 AFI、 NOP、 EOT、 SFP、 SFR、 EVENT)

简介

程序控制指令可用于更改逻辑执行顺序。

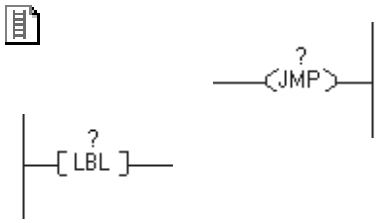
如果要	使用以下指令	在以下语言中可用	页码
跳过一段不需要始终执行的逻辑	JMP LBL	梯形图	442
跳转到单独的例程、将数据传递给这个例程、执行这个例程并返回结果	JSR SBR RET	梯形图 功能块 结构化文本	436
跳转到外部例程 (仅 SoftLogix5800 控制器)	JXR	梯形图	455
标记例程执行的临时结束	TND	梯形图 结构化文本	458
禁止一段逻辑中的所有梯级	MCR	梯形图	460
禁止用户任务	UID	梯形图 结构化文本	462
使能用户任务	UIE	梯形图 结构化文本	462
禁止梯级	AFI	梯形图	464
在逻辑中插入占位符	NOP	梯形图	465
顺序功能图的转换条件返回	EOT	梯形图 结构化文本	466
暂停顺序功能图	SFP	梯形图 结构化文本	468
复位顺序功能图	SFR	梯形图 结构化文本	470
触发事件任务的执行	EVENT	梯形图 结构化文本	472

跳转至标签 (JMP) 标签 (LBL)

JMP 和 LBL 指令用于跳过部分梯形图逻辑。

操作数：

梯形图



操作数	类型	格式	说明
JMP 指令			
标签名称 (Label name)		标签名称	输入相关 LBL 指令的名称
LBL 指令			
标签名称 (Label name)		标签名称	执行跳转到具有参考标签名称的 LBL 指令

说明：使能后， JMP 指令将跳转到参考 LBL 指令，控制器将从那里继续执行。禁止后， JMP 指令不会影响梯形图的执行。

JMP 指令可将梯形图执行位置向前或向后变动。向前跳转到标号可以省略掉一段逻辑并在需要时恢复，从而节省程序的扫描时间。向后跳转可以令控制器重复逻辑的执行。

注意向后跳转的次数不要过多。看门狗计时器可能由于控制器永远无法到达逻辑的末端而超时，从而令控制器出现故障。

注 意

不会扫描跳过的逻辑。请将重要逻辑放在跳过的区域之外。

LBL 指令是具有相同标签名称的 JMP 指令的目标。请确保 **LBL** 指令是梯级上的第一条指令。

标签名称在例程内必须惟一。名称可以：

- 最多 40 个字符。
- 含有字母、数字和下划线 (_)。

算术状态标志：不受影响

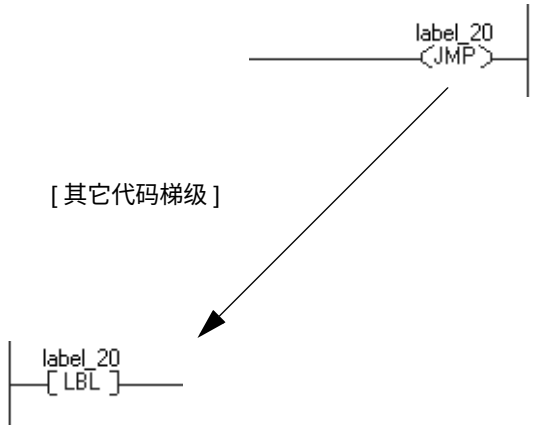
故障条件：

出现主要故障的条件	故障类型	故障代码
标号不存在	4	42

执行：

条件：	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	梯级输出条件设置为真。 执行跳转到梯级，该梯级包含具有参考标签名称的 LBL 指令。
后扫描	梯级输出条件设置为假。

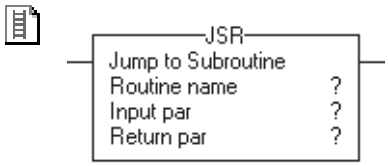
示例： 使能 JMP 指令后，执行将跳过连续的逻辑梯级，直至目标梯级，该梯级包含具有 *label_20* 的 LBL 指令。



跳转至子例程 (JSR) 子例程 (SBR) 返回 (RET)

JSR 指令可以使执行跳转到其它例程。 SBR 和 RET 指令是可选的指令，它们可以和 JSR 指令交换数据。

JSR 操作数：



梯形图

操作数	类型	格式	说明
例程名称 (Routine name)	ROUTINE	名称	要执行的例程 (即子例程)
输入参数 (Input parameter)	BOOL SINT INT DINT REAL 结构	立即数 标签 数组标签	该例程中要复制到子例程中的标签的数据 • 输入参数是可选的。 • 必要时可输入多个输入参数。
返回参数 (Return parameter)	BOOL SINT INT DINT REAL 结构	标签 数组标签	该例程中要将子例程的结果复制到其上的标签 • 返回参数是可选的。 • 必要时可输入多个返回参数。

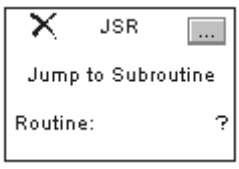
JSR 操作数 (续)：

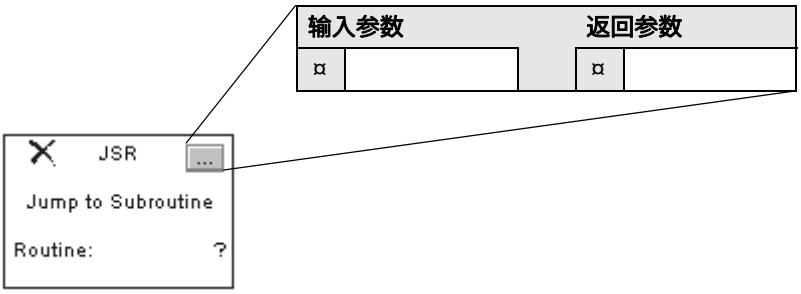
结构化文本

```
JSR(RoutineName,InputCount,
InputPar,ReturnPar);
```

操作数	类型	格式	说明
例程名称 (Routine name)	ROUTINE	名称	要执行的例程 (即子例程)
输入计数 (Input count)	SINT INT DINT REAL	立即数	输入参数的数量
输入参数 (Input parameter)	BOOL SINT INT DINT REAL 结构	立即数 标签 数组标签	该例程中要复制到子例程中的标签的数据 • 输入参数是可选的。 • 必要时可输入多个输入参数。
返回参数 (Return parameter)	BOOL SINT INT DINT REAL 结构	标签 数组标签	该例程中要将子例程的结果复制到其上的标签 • 返回参数是可选的。 • 必要时可输入多个返回参数。

功能块





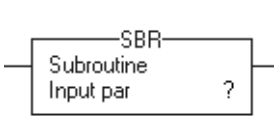
操作数与梯形图 JSR 指令的操作数相同。

注 意

对于 SBR 或 RET 指令中的每个参数，请使用与 JSR 指令中对应的参数相同的数据类型 (包括数组维数)。使用不同的数据类型可能导致出现意外的结果。

SBR 操作数： SBR 指令必须是梯形图或结构化文本例程中的第一条指令。

梯形图



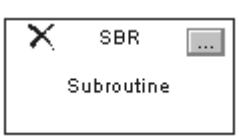
操作数	类型	格式	说明
输入参数 (Input parameter)	BOOL SINT INT DINT REAL 结构	标签 数组标签	该例程中要将对应的输入参数从 JSR 指令复制到其中的标签

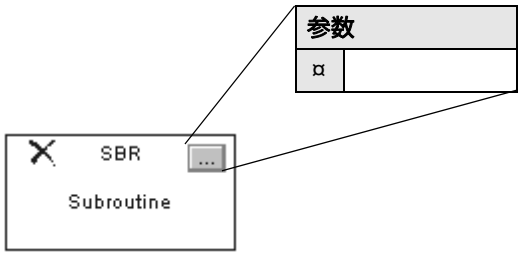
结构化文本

```
SBR(InputPar);
```

操作数与梯形图 SBR 指令的操作数相同。

功能块

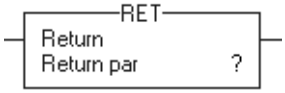




操作数与梯形图 SBR 指令的操作数相同。

RET 操作数：

梯形图



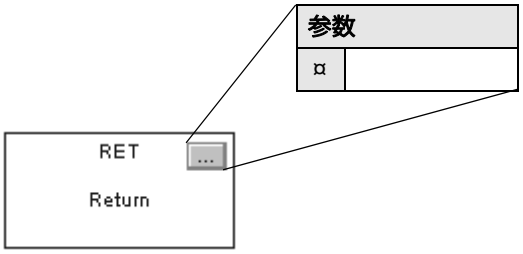
操作数	类型	格式	说明
返回参数 (Return parameter)	BOOL SINT INT DINT REAL 结构	立即数 标签 数组标签	该例程中的数据，要将其复制到 JSR 指令中的对应返回参数

结构化文本

```
RET(ReturnPar);
```

操作数与梯形图 RET 指令的操作数相同。

功能块



操作数与梯形图 RET 指令的操作数相同。

说明： JSR 指令可启动指定例程 (称为子例程) 的执行。

- 子例程将执行一次。
- 子例程执行后，逻辑执行将返回到含有 JSR 指令的例程。

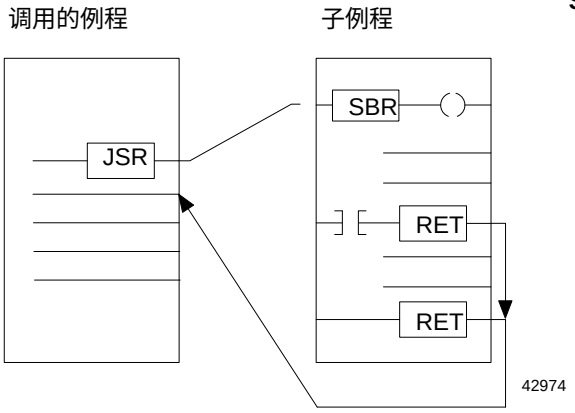
对跳转至子例程编程时，请遵循以下原则。

重要事项

不要使用 JSR 指令调用 (执行) 主例程。

- 可以将 JSR 指令置于主例程或其它任何例程中。
- 如果使用 JSR 指令调用主例程并随后将 RET 指令放入主例程，将出现主要故障 （类型 4，代码 31）。

下图显示指令的运行方式。



JSR

1. 如果要将数据复制到子例程中的标签内，则输入输入参数。
2. 如果要将子例程的结果复制到该例程中的标签内，则输入返回参数。
3. 根据需要输入多个输入参数和返回参数。

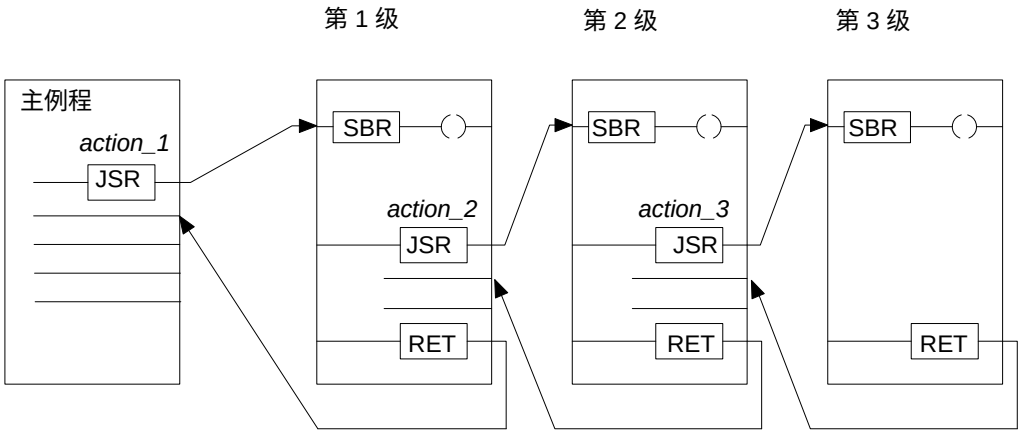
SBR

1. 如果 JSR 指令具有输入参数，则输入 SBR 指令。
2. 将 SBR 指令作为第一条指令放入例程中。
3. 对于 JSR 指令中的每个输入参数，输入要将数据复制到其中的标签。

RET

1. 如果 JSR 指令具有返回参数，则输入 RET 指令。
2. 将 RET 指令作为最后一条指令放入例程中。
3. 对于 JSR 指令中的每个返回参数，输入一个要发送到 JSR 指令的返回参数。
4. 在梯形图例程中，必要时可放置其它 RET 指令，以便根据不同的梯级条件来退出子例程。(功能块例程只允许使用一个 RET 指令。)

除控制器的内存外，嵌套例程的数量或者所传递或返回参数的数量不作任何限制。



算术状态标志： 算术状态标志将受到影响。

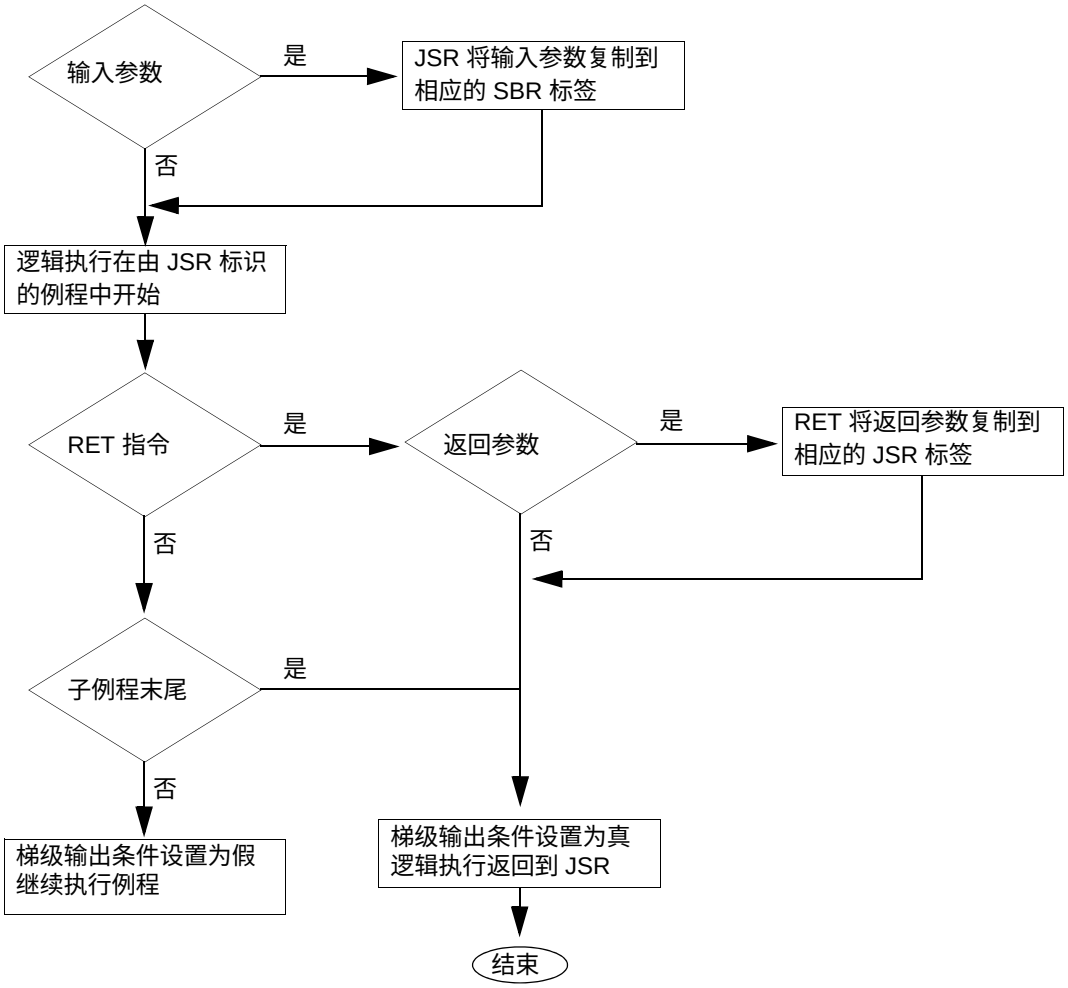
故障条件：

出现主要故障的条件	故障类型	故障代码
JSR 指令的输入参数比 SBR 指令的输入参数少	4	31
JSR 指令跳转到故障例程	4 或用户提供	0 或用户提供
RET 指令的返回参数比 JSR 指令的返回参数少	4	31
主例程内含有 RET 指令	4	31

执行：

梯形图 和结构化文本


条件	梯形图操作	结构化文本操作
预扫描	无论在任何梯级条件下，控制器都将执行所有子例程。为了确保子例程内的所有梯级都经过预扫描，控制器将忽略 RET 指令。 (即， RET 指令不会使子例程退出。) • 在版本 6.*x* 及更早的版本中，传递输入和返回参数。 • 在版本 7.*x* 及以后版本中，不传递输入和返回参数。 如果同一子例程存在递归调用，仅在首次执行时预扫描该子例程。如果同一子例程存在多次调用 (非递归)，每次执行时都预扫描该子例程。 梯级输出条件设置为假 (仅梯形图)。	
梯级输入条件对于 JSR 指令为假	子例程不执行。 子例程中的输出仍保持最后状态。 梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。

条件	梯形图操作	结构化文本操作
指令执行		
后扫描	动作与上述预扫描相同。	动作与上述预扫描相同。

功能块

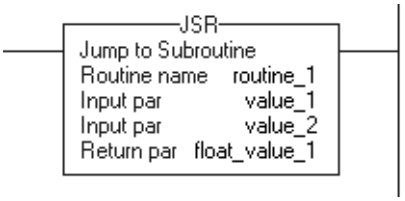
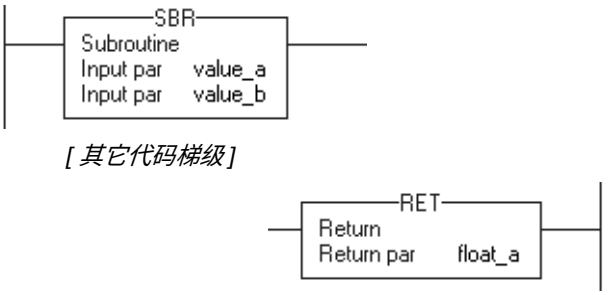
条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
正常执行	1. 如果例程中含有 SBR 指令，控制器将首先执行 SBR 指令。 2. 控制器锁存 IREF 中的所有数据值。 3. 控制器按照由连接确定的顺序执行其它功能块。这包括其它 JSR 指令。 4. 控制器将输出写入到 OREF 中。 5. 如果例程中含有 RET 指令，控制器将最后执行 RET 指令。
后扫描	调用子例程。 如果例程是 SFC 例程，例程的初始化方式将和预扫描期间的相同。

示例 **1**： JSR 指令将 *value_1* 和 *value_2* 传递到 *routine_1*。

SBR 指令从 JSR 指令接收 *value_1* 和 *value_2*，并将这些值分别复制到 *value_a* 和 *value_b*。逻辑执行在此例程中继续进行。

RET 指令将 *float_a* 发送到 JSR 指令。 JSR 指令接收到 *float_a* 后，将该值复制到 *float_value_1*。逻辑执行在 JSR 指令后的下一指令处继续。

梯形图

例程	编程
主例程	
子例程	

结构化文本

例程	编程
主例程	`JSR(routine_1,2,value_1,value_2,float_value_1);`

子例程	`SBR(value_a,value_b);`
	
	`    <statements>;`
	
	`RET(float_a);`

示例 2：

梯形图

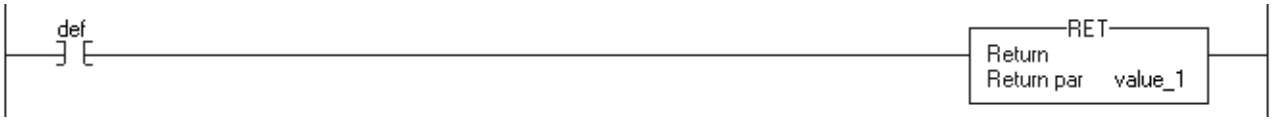
主例程

abc 使能时，执行 *subroutine_1*，计算 cookie 数，并将值放在 *cookies_1* 中。

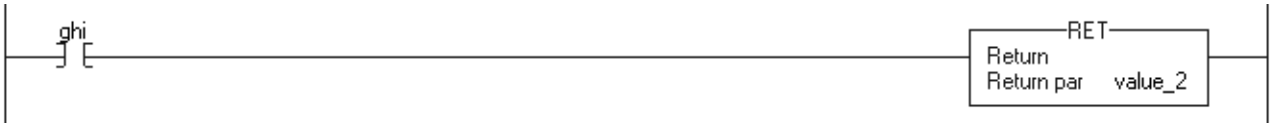
将 *cookies_1* 中的值加到 *cookies_2* 上，并将结果存储在 *total_cookies* 中。

Subroutine_1

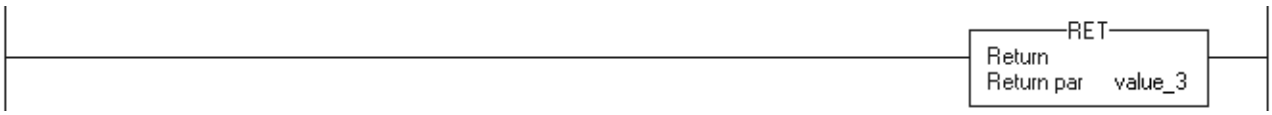
def 使能时， RET 指令将 *value_1* 返回到 JSR 的参数 *cookies_1*，并且不会扫描子例程的其余部分。



在 *def* 关闭 (前一梯级) 并且 *ghi* 使能时， RET 指令将 *value_2* 返回到 JSR 的参数 *cookies_1*，并且不会扫描子例程的其余部分。



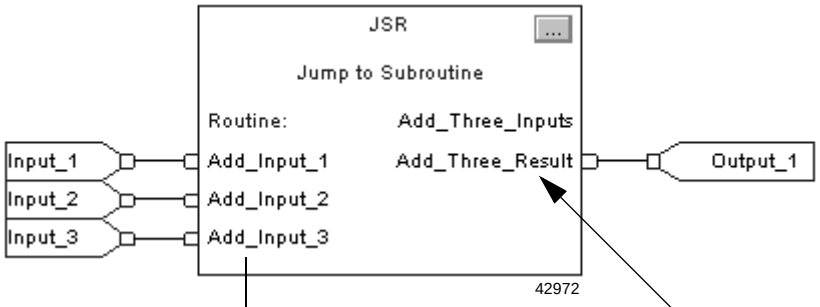
在 *def* 和 *ghi* 全都关闭 (前面的梯级) 时， RET 指令将 *value_3* 返回到 JSR 的参数 *cookies_1*。



示例 3：

功能块

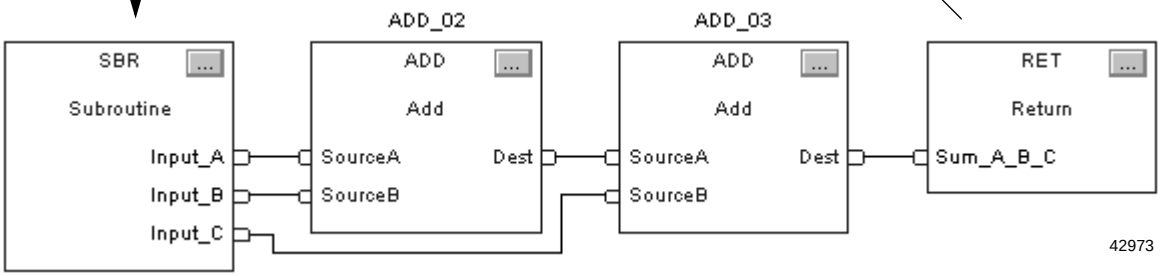
Routine_A 中的 JSR 指令



1. *Add_Input_1*、*Add_Input_2* 和 *Add_Input_3* 中的值被分别复制到 *Input_A*、 *Input_B* 和 *Input_C*。

3. *Sum_A_B_C* 的值被复制到 *Add_Three_Result*。

Add_Three_Inputs 例程的功能块



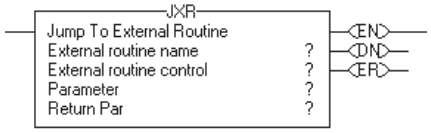
2. ADD 指令将 *Input_A*、 *Input_B* 和 *Input_C* 相加，并将结果放在 *Sum_A_B_C* 中。

跳转至外部例程 (JXR)

JXR 指令用于执行外部例程。仅 SoftLogix5800 控制器支持该指令。

操作数：

梯形图



.

操作数	类型	格式	说明
外部例程名称 (External routine name)	ROUTINE	名称	要执行的外部例程
外部例程控制 (External routine control)	EXT_ROUTINE_CONTROL	标签	控制结构
参数 (Parameter)	BOOL SINT INT DINT REAL 结构	立即数 标签 数组标签	该例程中要复制到外部例程中的变量的数据 • 参数是可选的。 • 需要时可输入多个参数。 • 最多可输入 10 个参数。
返回参数 (Return parameter)	BOOL SINT INT DINT REAL	标签	该例程中要将外部例程的结果复制到其中的标签 • 返回参数是可选的。 • 只能有一个返回参数

EXT_ROUTINE_CONTROL 结构

助记符	数据类型	说明	实施
ErrorCode	SINT	若发生错误，该值标识错误。有效值的范围是 0-255。	没有预定义的错误代码。外部例程的开发商必须提供错误代码。
NumParams	SINT	该值指示与该指令有关的参数的数量。	仅显示 - 此信息来自指令条目。
ParameterDefs	EXT_ROUTINE_PARAMETERS[10]	此数组含有要传递到外部例程的参数的定义。该指令最多可传递 10 个参数。	仅显示 - 此信息来自指令条目。
ReturnParamDef	EXT_ROUTIN_PARAMETERS	该值含有外部例程中的返回参数的定义。只有一个返回参数。	仅显示 - 此信息来自指令条目。
EN	BOOL	置位后，使能位指示 JXR 指令已使能。	外部例程对该位进行置位。
ReturnsValue	BOOL	如果已置位，则该位指示已经为指令输入了返回参数。如果已清零，则该位指示没有为指令输入返回参数。	仅显示 - 此信息来自指令条目。
DN	BOOL	当外部例程执行完一次时，将置位完成位。	外部例程对该位进行置位。
ER	BOOL	出现错误时，将置位错误位。该指令停止执行，直到程序清零错误位。	外部例程对该位进行置位。
FirstScan	BOOL	该位可表明这是否是控制器切换到运行模式后的第一次扫描。需要时可使用 FirstScan 初始化外部例程。	控制器将该位置位以反映扫描状态。
EnableOut	BOOL	使能输出。	外部例程对该位进行置位。
EnableIn	BOOL	使能输入。	控制器将该位置位以反映梯级输入条件。无论在什么梯级条件下，都将执行指令。外部例程的开发商应该监视此状态，并相应地作出回应。
User1	BOOL	这些位对于用户可用。控制器不初始化这些位。	外部例程或用户程序可置位这些位。
User0	BOOL		
ScanType1	BOOL	这些位标识当前扫描类型： 位值：　扫描类型： 00　正常 01　预扫描 10　后扫描 (不适用于梯形图程序)	控制器将这些位置位以反映扫描状态。
ScanType0	BOOL		

说明： “ 跳转至外部例程 ”(JXR) 指令用于从项目的梯形图例程中调用外部例程。 JXR 指令支持多个参数，因此可以在梯形图例程和外部例程之间传递值。

JXR 指令类似于 “ 跳转至子例程 ”(JSR) 指令。 JXR 指令可启动指定外部例程的执行：

- 外部例程将执行一次。
- 外部例程执行完毕后，逻辑执行将返回到含有 JXR 指令的例程。

算术状态标志： 算术状态标志不会受到影响。

故障条件：

出现主要故障的条件	故障类型	故障代码：
2外部例程 DLL 中出现异常。 2无法装载 DLL。 2DLL 中未找到入口点。	4	88

执行： JXR 可以是同步的，也可以是异步的，这取决于 DLL 的实施。 DLL 中的代码也可以确定如何响应扫描状态、梯级输入条件状态和梯级输出条件状态。

有关使用 JXR 指令和创建外部例程的详细信息，
请参见 《SoftLogix5800 系统用户手册》 (出版号 1789-UM002)。

临时结束 (TND)

TND 指令起分界线的作用。

操作数：

梯形图操作数

无

```
TND();
```

结构化文本

无

必须在指令助记符后面输入圆括号 ()，即使没有操作数也要输入。

说明： 使能后， TND 指令将使得控制器只能将逻辑执行到该指令。

使能后， TND 指令将充当例程的末尾。当控制器扫描 TND 指令时，控制器将移动到当前例程的末尾。如果 TND 指令位于子例程内，则控制将返回到调用例程。如果 TND 指令位于主例程内，则控制将返回到当前任务内的下一个程序。

算术状态标志： 不受影响

故障条件： 无

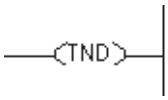
执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	当前例程终止。	当前例程终止。
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 调试或排除故障时，可使用 TND 指令将逻辑执行到某一点。随着调试每个新部分，逐渐将 TND 指令在逻辑中移动。

当 TND 指令使能后，控制器将停止扫描当前例程。

梯形图



结构化文本

```
TND();
```

主控复位 (MCR)

MCR 指令成对使用时，可创建一个程序区域，该区域可禁止 MCR 指令内的所有梯级。

操作数：

梯形图

无

说明：使能 MCR 区域后，将针对常规的真或假条件对 MCR 区域内的梯级进行扫描。禁止后，控制器将仍然对 MCR 区域内的梯级进行扫描，但扫描时间将缩短，因为区域内的非保持型输出已被禁止。对于禁止的 MCR 区域内的所有指令，梯级输入条件都为假。

对 MCR 区域编程时，请注意：

- 必须使用无条件 MCR 指令结束区域。
- 不能将一个 MCR 区域嵌套在另一个 MCR 区域中。
- 不要跳转到 MCR 区域中。区域为假时，如果跳转到该区域，将激活从跳入点到区域末尾的那一段梯级逻辑。
- 如果 MCR 区域持续到例程的末尾，则不必用 MCR 指令结束区域。

MCR 指令不能代替具有急停功能的硬接线主控继电器。应该仍然安装硬接线主控继电器以提供应急 I/O 电源切断功能。

注 意

不要重叠或嵌套 MCR 区域。每个 MCR 区域都必须是独立而完整的。如果 MCR 区域重叠或嵌套，机器的运转情况将无法预测，并可能造成设备损坏或人员伤害。

请将重要的操作放在 MCR 区域以外。如果在 MCR 区域中启动计时器等指令，当区域被禁止且计时器清零时，指令将停止执行。

算术状态标志：不受影响

故障条件：无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。 扫描区域内的指令，但梯级输入条件为假，且区域中的非保持型输出被禁止。
梯级输入条件为真	梯级输出条件设置为真。 正常扫描区域内的指令。
后扫描	梯级输出条件设置为假。

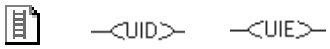
示例： 使能第一条 MCR 指令 (*input_1*、 *input_2* 和 *input_3* 置位) 时，控制器将执行 MCR 区域 (两条 MCR 指令之间) 中的梯级并将输出置位或清零，这取决于输入条件。

禁止第一条 MCR 指令 (*input_1*、 *input_2* 和 *input_3* 没有全部置位) 时，控制器将执行 MCR 区域 (两条 MCR 指令之间) 中的梯级，并且无论输入条件为何， MCR 区域中的所有梯级的梯级输入条件都将为假。

禁止用户中断 (UID) 允许用户中断 (UIE)

UID 指令和 UIE 指令可配合使用，以防止少数的重要梯级被其它任务中断。

操作数：



梯形图

无

```
UID();
UIE();
```

结构化文本

无

必须在指令助记符后面输入圆括号 ()，即使没有操作数也要输入。

说明：当梯级输入条件为真时：

- UID 指令将阻止优先权较高的任务中断当前的任务，但不会禁止故障例程或控制器故障处理器的执行。
- UIE 指令可允许其它任务中断当前任务。

请执行以下步骤，以防止一系列梯级被中断。

1. 尽可能将不想被中断的梯级的数量降到最低。

   如果长时间禁止中断，可能造成通信丢失。

2. 在*不*想被中断的第一个梯级上面，输入一个梯级和一条 UID 指令。

3. 在不想被中断的梯级系列的最后一个梯级的后面，输入一个梯级和一条 UIE 指令。

4. 必要时，可以嵌套 UID/UIE 指令对。

算术状态标志：不受影响

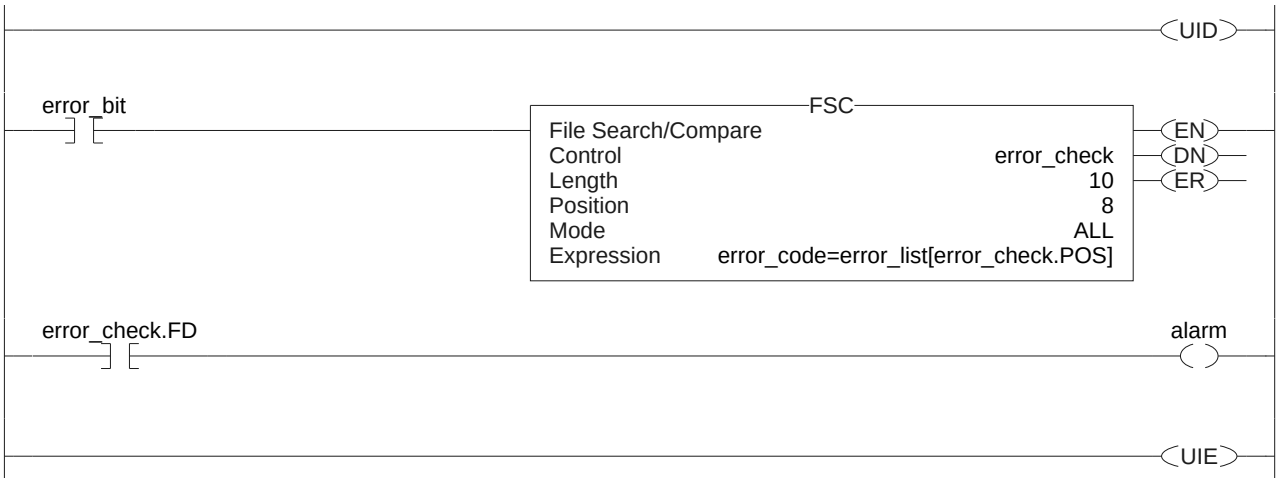
故障条件：无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	UID 指令防止优先级较高的任务进行中断。 UIE 指令允许优先级较高的任务进行中断。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 出现错误 (*error_bit* 开启) 时， FSC 指令根据严重错误列表检查错误代码。如果 FSC 指令发现该错误是严重错误 (error_check.FD 开启)，将发出警报。 UID 和 UIE 指令阻止其它任务中断错误检查和警报。

梯形图



结构化文本

```
UID();

    <statements>

UIE();
```

恒假指令 (AFI)

AFI 指令可将其梯级输出条件设置为假。

操作数：

梯形图

无

说明： AFI 指令可将其梯级输出条件设置为假。

算术状态标志： 不受影响

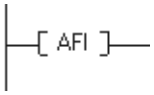
故障条件： 无

执行：

条件	梯形图操作：
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	梯级输出条件设置为假。
后扫描	梯级输出条件设置为假。

示例： 调试程序时，可使用 AFI 指令临时禁止梯级。

使能后， AFI 将禁止该梯级上的所有指令。



空操作 (NOP)

NOP 指令起占位符的作用。

操作数：

梯形图

-[NOP]-

无

说明： 可以将 NOP 指令放在梯级上的任何位置。使能后， NOP 指令不会执行任何操作。禁止后， NOP 指令不会执行任何操作。

算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

示例 将 NOP 指令放在分支上后，该指令可用于定位无条件分支。

NOP 指令可绕过 XIC 指令来使能输出。

返回转换 (EOT)

EOT 指令将布尔状态返回到 SFC 转换条件。

操作数：

梯形图

操作数	类型	格式	说明
数据位 (Data bit)	BOOL	标签	转换的状态 (0 = 正在执行， 1 = 已完成)

结构化文本

```
EOT(data_bit);
```

操作数与梯形图 EOT 指令的操作数相同。

说明：因为 EOT 指令返回布尔状态，所以多个 SFC 例程可共享含有 EOT 指令的同一个例程。如果调用例程不是转换条件，则 EOT 指令将充当 TND 指令 (请参见第 458 页)。

EOT 指令的 Logix 实施与 PLC-5 控制器中的情况不同。在 PLC-5 控制器中， EOT 指令没有参数。 PLC-5 EOT 指令返回梯级条件作为其状态。而在 Logix 控制器中，返回参数作为返回转换状态，因为所有 Logix 编程语言中梯级条件只决定是否返回参数。

算术状态标志：不受影响

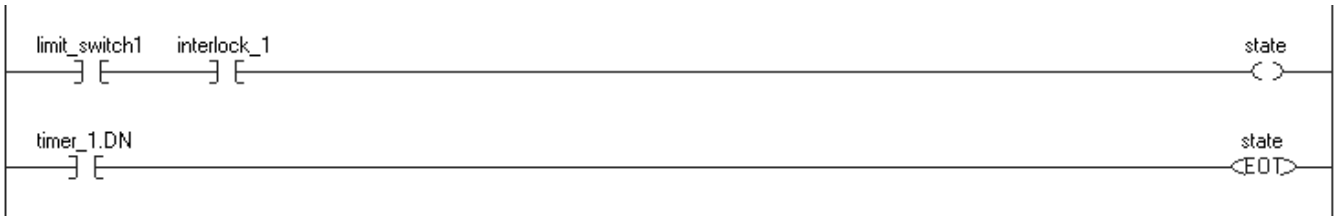
故障条件：无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令将数据位值返回到调用例程。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 当 *limit_switch1* 和 *interlock_1* 均置位时，将置位状态。*timer_1* 完成后， EOT 将 *state* 的值返回到调用例程。

梯形图



结构化文本

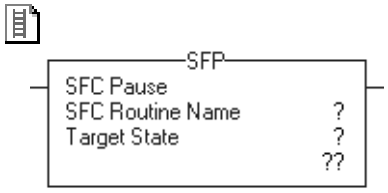
```
state := limit_switch1 AND interlock_1;


IF timer_1.DN THEN
    EOT(state);
END_IF;
```

SFC 暂停 (SFP)

SFP 指令用于暂停 SFC 例程。

操作数：

梯形图



操作数	类型	格式	说明
SFC 例程名称 (SFCRoutine Name)	ROUTINE	名称	要暂停的 SFC 例程
目标状态 (TargetState)	DINT	立即数 标签	选择一种状态： 正在执行 (或输入 0) 已暂停 (或输入 1)

结构化文本

```
SFP(SFCRoutineName,
    TargetState);
```

操作数与梯形图 SFP 指令的操作数相同。

说明：SFP 指令可暂停一个正在执行的 SFC 例程。如果 SFC 例程处于已暂停状态，再次使用 SFP 指令可更改状态并恢复例程的执行。

另外，在使用 SFR 指令 (请参见第 470 页) 复位 SFC 例程后，可使用 SFP 指令恢复 SFC 的执行。

算术状态标志：不受影响

故障条件：

出现主要故障的条件	故障类型	故障代码
例程类型不是 SFC 例程	4	85

执行：

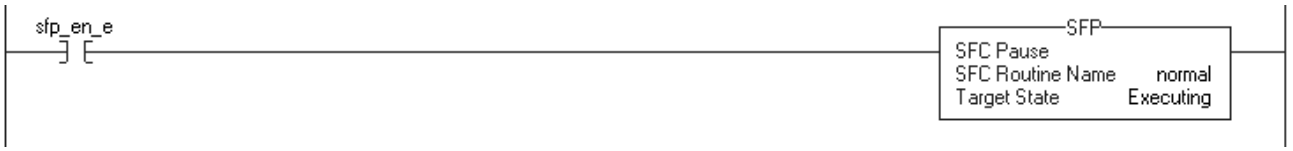
条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	指令暂停或恢复指定 SFC 例程的执行。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 如果 *sfc_en_p* 置位，则暂停名为 *normal* 的 SFC 例程。在 *sfc_en_e* 置位后，重新启动 SFC。

梯形图

暂停 SFC 例程。

恢复执行 SFC 例程。



结构化文本

暂停 **SFC** 例程：

```
IF (sfp_en_p) THEN
     SFP(normal,paused);
     sfp_en_p := 0;
END_IF;
```

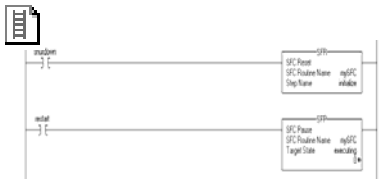
恢复执行 **SFC** 例程：

```
IF (sfp_en_e) THEN
     SFP(normal,executing);
     sfp_en_e := 0;
END_IF;
```

SFC 复位 (SFR)

SFR 指令可以在指定的步骤复位 SFC 例程的执行。

操作数：



梯形图操作数

操作数	类型	格式	说明
SFC 例程名称 (SFCRoutine Name)	ROUTINE	名称	要复位的 SFC 例程
步名称 (Step Name)	SFC_STEP	标签	目标步，在此处恢复执行

结构化文本

```
SFR(SFCRoutineName,StepName);
```

操作数与梯形图 SFR 指令的操作数相同。

说明：使能 SFR 指令时：

- 在指定的 SFC 例程内，存储的所有操作都将停止执行 (复位)。
- SFC 在指定步开始执行。

如果目标步为 0，顺序功能图将复位到初始步

SFR 指令的 Logix 实施与 PLC-5 控制器中的情况不同。在 PLC-5 控制器中， SFR 在梯级条件为真时执行。复位后， SFC 将一直处于暂停状态，直至含有 SFR 的梯级为假为止。这样可延迟执行复位后的操作。PLC-5 SFR 指令的这种暂停 / 取消暂停功能已经和梯级条件分离，移动到 SFP 指令中。

算术状态标志：不受影响

故障条件：

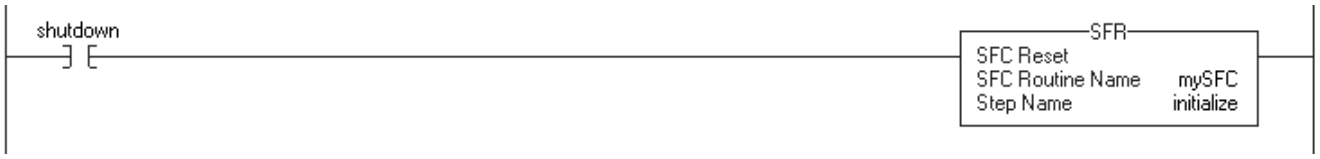
出现主要故障的条件	故障类型	故障代码
例程类型不是 SFC 例程	4	85
SFC 例程中不存在指定的目标步	4	89

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	指令复位指定的 SFC 例程。	指令复位指定的 SFC 例程。
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 如果满足特定的条件 (*shutdown* 置位)，在步骤 *initialize* 处重新启动 SFC。

梯形图



结构化文本

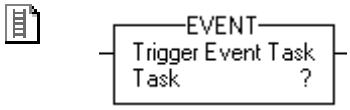
```
IF shutdown THEN

     SFR(mySFC,initialize);

END_IF;
```

触发事件任务 (EVENT)

EVENT 指令用于触发事件任务的执行。

操作数：

梯形图



操作数	类型	格式	说明
任务 (Task)	TASK	名称	要执行的事件任务
			该指令允许您选择其它类型的任务，但不会执行这些任务。

结构化文本

```
EVENT(task_name);
```

操作数与梯形图 EVENT 指令的操作数相同。

说明： EVENT 指令可用于通过编程执行事件任务：

- 每次执行指令时，指令都会触发指定的事件任务。
- 确保在再次触发事件任务之前给予该任务充分的时间完成执行。否则，将发生交迭。
- 如果在事件任务正在执行时执行 EVENT 指令，控制器将增加交迭计数器的计数，但不会触发事件任务。

编程确定 EVENT 指令是否触发了任务

要确定 EVENT 指令是否触发了事件任务，使用获取系统值 (GSV) 指令来监视任务的“ 状态 ”(Status) 属性。

TASK 对象的状态属性

属性	数据类型	指令	说明	
状态 (Status)	DINT	GSV SSV	提供任务的状态信息。一旦控制器置位一个位，则必须手动清零该位以确定是否发生了另一个该类型故障。	
			要确定	检查以下位
			EVENT 指令是否触发了任务 (仅事件任务)。	0
			超时是否触发了任务 (仅事件任务)。	1
			该任务是否出现了交迭。	2

“ 状态 ” (Status) 属性的位置位后，控制器不会清零这些位。

- 要将一个位用于新的状态信息，则必须手动清零该位。
- 使用设置系统值 (SSV) 指令将属性设置为其它值。

算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。	N/A
	梯级输出条件设置为真。	
EnableIn 处于置位状态	N/A	EnableIn 始终置位。
		执行该指令。
指令执行	指令触发指定事件任务的一次执行	
后扫描	梯级输出条件设置为假。	不执行任何操作。

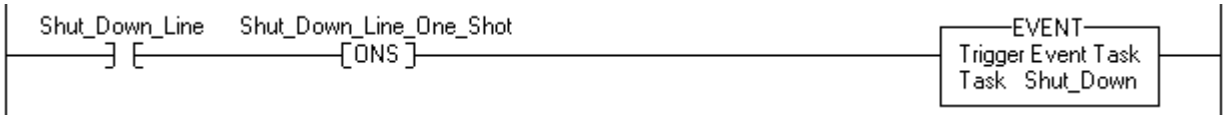
示例 **1**： 控制器使用多个程序，但使用一个公用的关闭程序。每个程序都使用名为 *Shut_Down_Line* 的程序作用域标签，该标签会在程序检测到要求关闭的条件时开启。每个程序中的逻辑都按照如下方式来执行：

如果 *Shut_Down_Line* = 开启 (条件要求关闭)，则

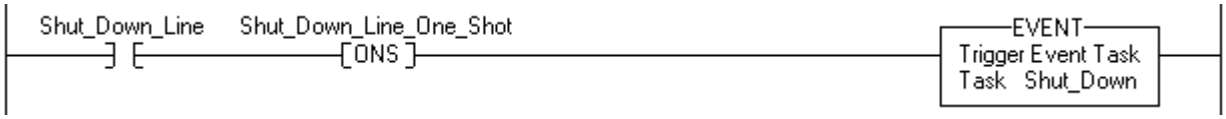
执行一次 *Shut_Down* 任务

梯形图

程序 ***A***



程序 ***B***



结构化文本

程序 ***A***

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
  EVENT (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot := Shut_Down_Line;
```

程序 **B**

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
  EVENT (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot := Shut_Down_Line;
```

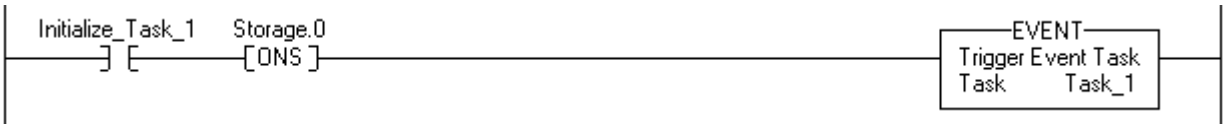
示例 **2**： 在下面的示例中，EVENT 指令用于初始化一个事件任务。(另一类型的事件通常会触发事件任务。)

连续任务

如果 *Initialize_Task_1* = 1，则

ONS 指令将 EVENT 指令的执行限制为一次扫描。

EVENT 指令触发 *Task_1*(事件任务）的执行。

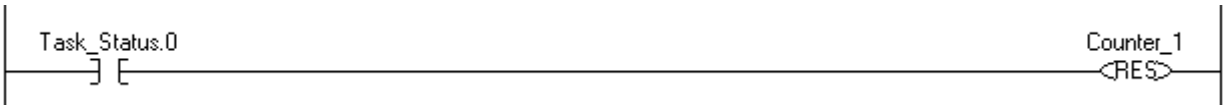


Task_1(事件任务)

GSV 指令设置 *Task_Status*(DINT 标签)= 事件任务的 “ 状态 ” (Status) 属性。在 “ 实例名称 ” (Instance Name) 属性中， THIS 表示指令所在任务的 TASK 对象 (即 *Task_1*)。

如果 *Task_Status*.0 = 1，则说明 EVENT 指令触发了事件任务 (即，当连续任务执行 EVENT 指令初始化事件任务时)。

RES 指令复位事件任务所使用的计数器。

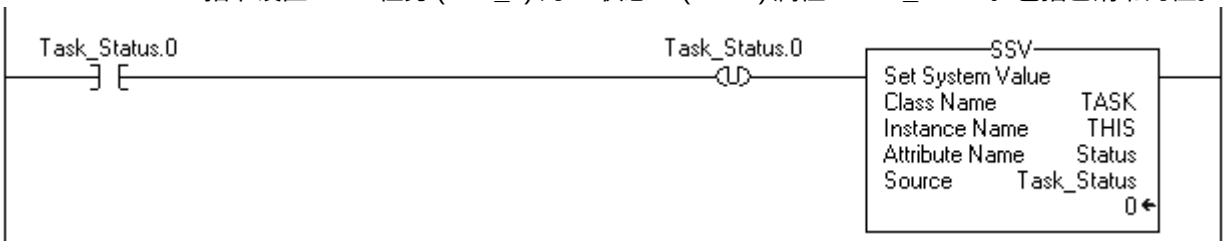


“ 状态 ” (Status) 属性的位置位后，控制器不会清零这些位。要将一个位用于新的状态信息，则必须手动清零该位。

如果 *Task_Status*.0 = 1，则清零该位。

OTU 指令设置 *Task_Status*.0 = 0。

SSV 指令设置 THIS 任务 (*Task_1*) 的 “ 状态 ” (Status) 属性 = *Task_Status*。包括已清零的位。



注：

循环 / 中断指令
(FOR、 FOR...DO、 BRK、 EXIT、 RET)

简介

使用 FOR 指令可重复调用子例程。使用 BRK 指令可中断子例程的执行。

如果要	使用以下指令	可用编程语言	页码
重复执行例程	FOR FOR...DO(1)	梯形图 结构化文本	478
终止例程的重复执行	BRK EXIT(1)	梯形图 结构化文本	481
返回到 FOR 指令	RET	梯形图	482

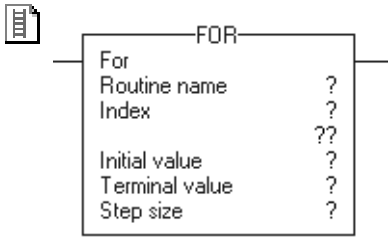
(1) 仅限结构化文本。

循环 (FOR)

FOR 指令用于重复执行某一例程。

操作数：

梯形图



操作数	类型	格式	说明
例程名称 (Routine name)	ROUTINE	例程名称	要执行的例程
索引 (Index)	DINT	标签	计数例程的执行次数
初始值 (Initial value)	SINT INT DINT	立即数 标签	索引起始值
终止值 (Terminal value)	SINT INT DINT	立即数 标签	达到该值时例程停止执行
步长 (Step size)	SINT INT DINT	立即数 标签	FOR 指令每执行一次例程，索引值的增量

结构化文本

```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

使用 FOR...DO 结构。有关结构化文本结构的信息，请参见结构化文本编程。

说明：

重要事项

请勿使用 FOR 指令调用 (执行) 主例程。

- 可将 FOR 指令放入主例程或其它任何例程中。
- 如果使用 FOR 指令调用主例程并随后将 RET 指令放入主例程，将出现主要故障 (类型 4，代码 31)。

使能后， FOR 指令重复执行例程，直到索引值超出终止值。该指令不将参数传递到例程。

FOR 指令每次执行例程时，都将步长添加到索引中。

请注意，不要在一次扫描中循环太多次。重复次数太多可导致控制器的看门狗超时，进而引起主要故障。

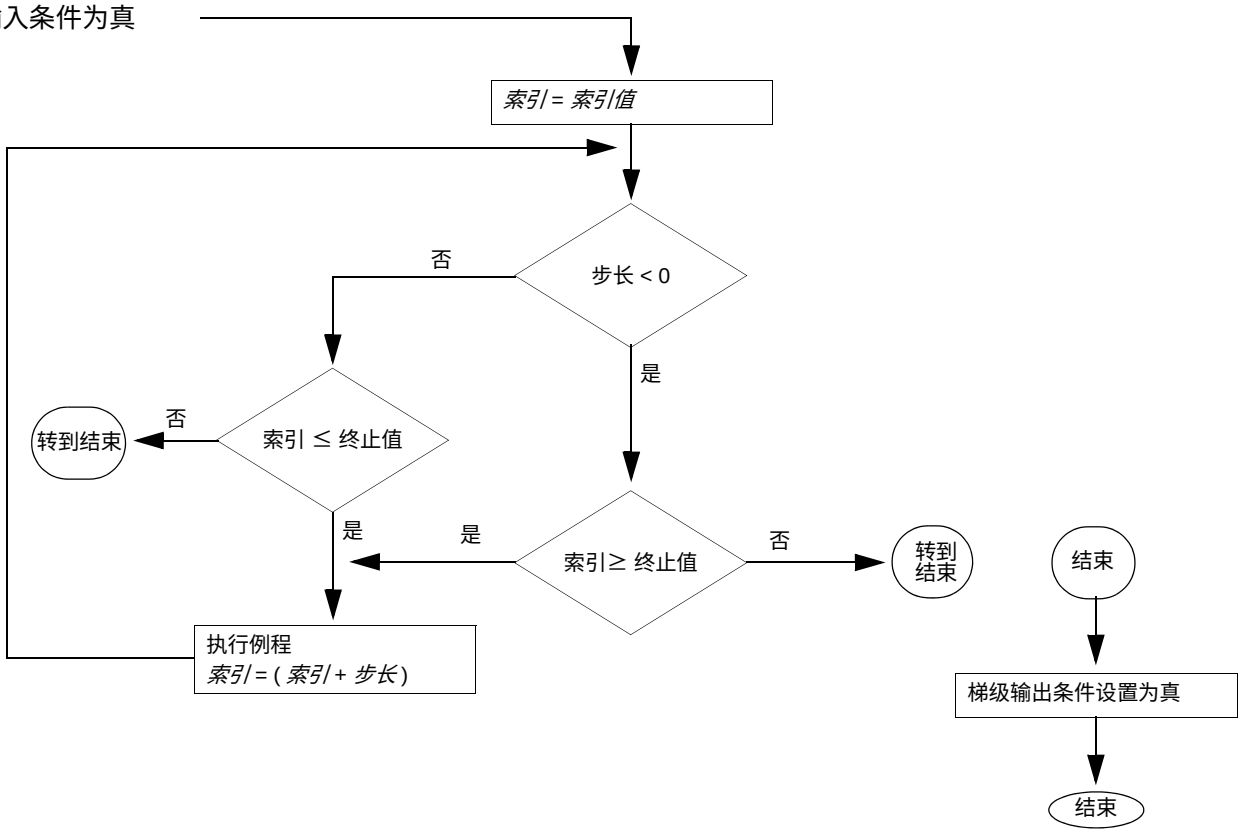
算术状态标志： 不受影响

故障条件：

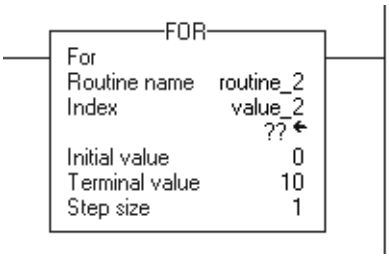
出现主要故障的条件	故障类型	故障代码
主例程内含有 RET 指令	4	31

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。 控制器执行一次子例程。 如果同一子例程存在递归 FOR 指令，仅在首次执行时预扫描该子例程。如果同一子例程存在多个 FOR 指令 (非递归)，每次执行时都预扫描该子例程。
梯级输入条件为假	梯级输出条件设置为假。

条件	梯形图操作
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

示例： 使能后，FOR 指令重复执行 *routine_2*，*value_2* 每次加 1。*value_2* > 10 或使能了 BRK 指令时， FOR 指令不再执行 *routine_2*。

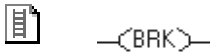


中断 (BRK)

BRK 指令用于中断执行 FOR 指令调用的例程。

操作数：

梯形图



无

结构化文本

```
EXIT;
```

在循环结构中使用 EXIT 语句。有关结构化文本结构的信息，请参见 附录 B。

说明： 使能后， BRK 指令退出例程并将控制器返回到紧随 FOR 指令的下一指令。

如果存在嵌套的 FOR 指令，BRK 指令将控制器返回到最内部的 FOR 指令。

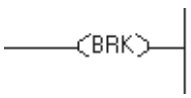
算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	梯级输出条件设置为真。 执行返回到紧随调用的 FOR 指令的下一指令。
后扫描	梯级输出条件设置为假。

示例： 使能后， BRK 指令停止执行当前例程并返回到紧随调用的 FOR 指令的下一指令。

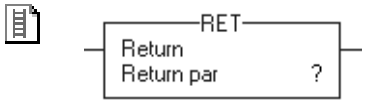


返回 (RET)

RET 指令用于返回到调用的 FOR 指令。

操作数：

梯形图



无

说明：

重要事项　请勿将 RET 指令放入主例程。如将 RET 指令放入主例程，会出现主要故障 (类型 4，代码 31)。

使能后，RET 指令返回到 FOR 指令。FOR 指令按步长递增索引值，然后再次执行子指令。如果索引值超出终止值， FOR 指令结束，然后执行紧随 FOR 指令的下一指令。

FOR 指令不使用参数。FOR 指令会忽略在 RET 指令中输入的任何参数。

也可使用 TND 指令结束子例程的执行 。

算术状态标志：　不受影响

故障条件：

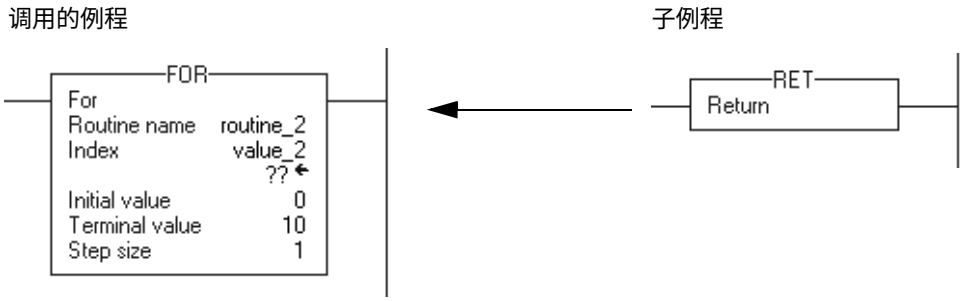
出现主要故障的条件	故障类型	故障代码
主例程内含有 RET 指令	4	31

执行：

条件：	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	将指定参数返回到调用的例程。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

示例： FOR 指令重复执行 *routine_2*，*value_2* 每次加 1。*value_2* > 10 或使能了 BRK 指令时， FOR 指令不再执行 *routine_2*。

RET 指令返回到调用的 FOR 指令。FOR 指令再次执行子例程并按步长递增索引值，或者，如果索引值超出终止值， FOR 指令结束，然后执行紧随 FOR 指令的下一指令。



注：

特殊指令
(FBC、 DDT、 DTR、 PID)

简介

特殊指令执行应用特定的操作。

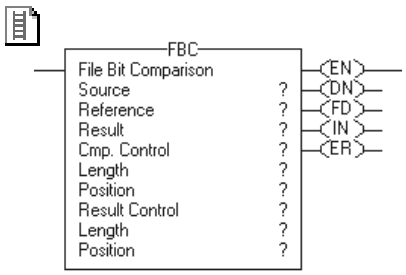
如果要	使用以下指令	在以下语言中可用	页码
将数据与已知的正确参考进行比较，并记录所有不匹配项	FBC	梯形图	486
将数据与已知的正确参考进行比较，记录所有不匹配项，并更新该参考使其与源匹配	DDT	梯形图	494
通过屏蔽传递源数据，并将结果与参考数据进行比较。然后将源写入参考，供下一次比较使用	DTR	梯形图	502
控制 PID 回路	PID	梯形图 结构化文本	505

文件位比较 (FBC)

FBC 指令将源数组中的位与参考数组中的位进行比较。

操作数：

梯形图



操作数	类型	格式	说明：
源 (Source)	DINT	数组标签	与参考进行比较的数组 不要在下标中使用 CONTROL.POS
参考 (Reference)	DINT	数组标签	与源进行比较的数组 不要在下标中使用 CONTROL.POS
结果 (Result)	DINT	数组标签	存储结果的数组 不要在下标中使用 CONTROL.POS
比较控制 (Cmp control)	CONTROL	结构	比较的控制结构
长度 (Length)	DINT	立即数	要比较的位数
位置 (Position)	DINT	立即数	源中的当前位置 初始值通常为 0
结果控制 (Result control)	CONTROL	结构	结果的控制结构
长度 (Length)	DINT	立即数	结果中的存储位置数
位置 (Position)	DINT	立即数	结果中的当前位置 初始值通常为 0

注 意

对比较控制结构和结果控制结构使用不同的标签。对这两个结构使用同一个标签可能会导致不可预知的操作，这可能引起设备损坏和 / 或人身伤害。

COMPARE 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 FBC 指令是否使能。
.DN	BOOL	当 FBC 指令比较 Source 和 Reference 数组中的最后一位后，完成位置位。
.FD	BOOL	FBC 指令每次记录不匹配项 (每次记录一个) 或记录所有不匹配项 (每次扫描全部记录) 后，找到位置位。
.IN	BOOL	禁止位指示 FBC 搜索模式。 0 = “ 所有 ” 模式 1 = “ 每次一个不匹配项 ” 模式
.ER	BOOL	当 compare.POS < 0、compare.LEN < 0、result.POS < 0 或 result.LEN < 0 时，错误位置位。指令停止执行，直到程序清零 .ER 位。
.LEN	DINT	长度值标识要比较的位数。
.POS	DINT	位置值标识当前位。

RESULT 结构

助记符	数据类型	说明
.DN	BOOL	结果数组已满时，完成位置位。
.LEN	DINT	长度值标识结果数组中的存储位置数。
.POS	DINT	位置值标识结果数组中的当前位置。

说明： 使能后，FBC 指令将源数组中的位与参考数组中的位进行比较，并在结果数组中记录各个不匹配项的位号。

重要事项

必须测试并确认指令不会更改您不希望更改的数据。

FBC 指令对连续内存进行操作。在某些情况下，该指令会越过数组搜索或写入标签的其它子元素。如果 length 过大且标签是用户自定义的数据类型，就会发生这种情况。

DDT 和 FBC 指令的区别是：DDT 指令每次发现不匹配项时，会更改参考位使其与源位匹配。而 FBC 指令不更改参考位。

选择搜索模式

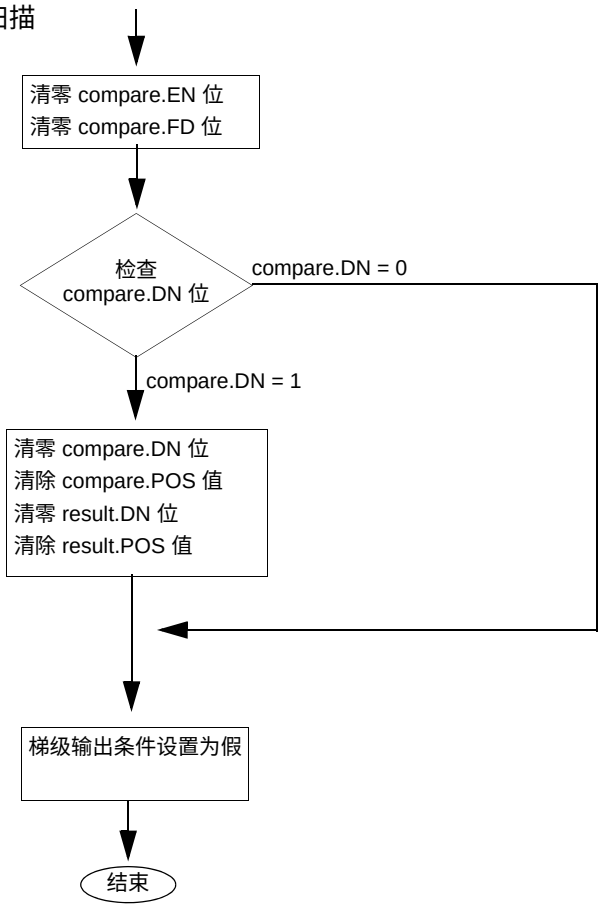
如果要	选择以下模式
每次检测一个不匹配项	置位比较 CONTROL 结构中的 .IN 位。 每次梯级输入条件从假变为真时， FBC 指令都会搜索源数组和参考数组之间的下一个不匹配项。找到不匹配项后，该指令会置位 .FD 位，记录该不匹配项的位置，并停止执行。
检测所有不匹配项	清零比较 CONTROL 结构中的 .IN 位。 每次梯级输入条件从假变为真时， FSC 指令都会搜索源数组和参考数组之间的所有不匹配项。

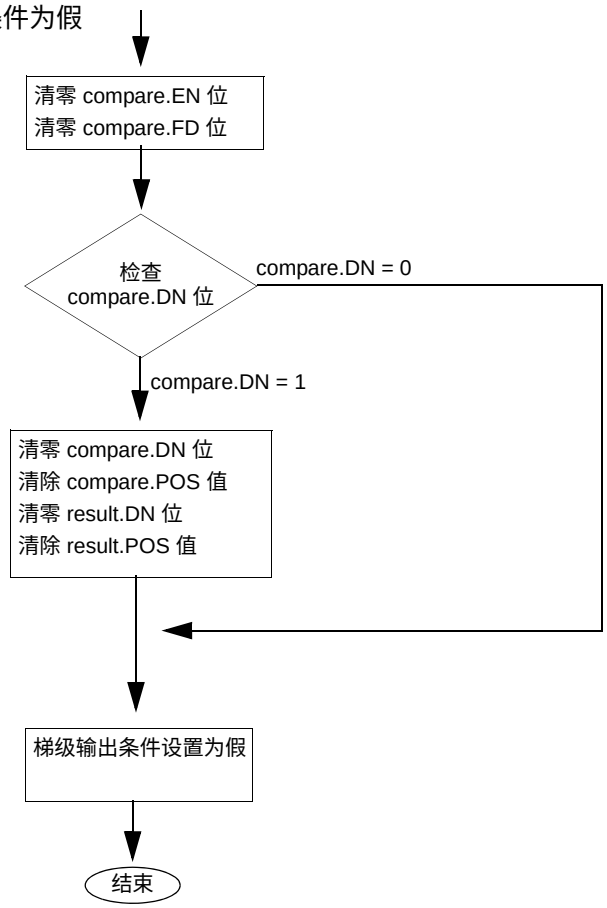
算术状态标志： 不受影响

故障条件：

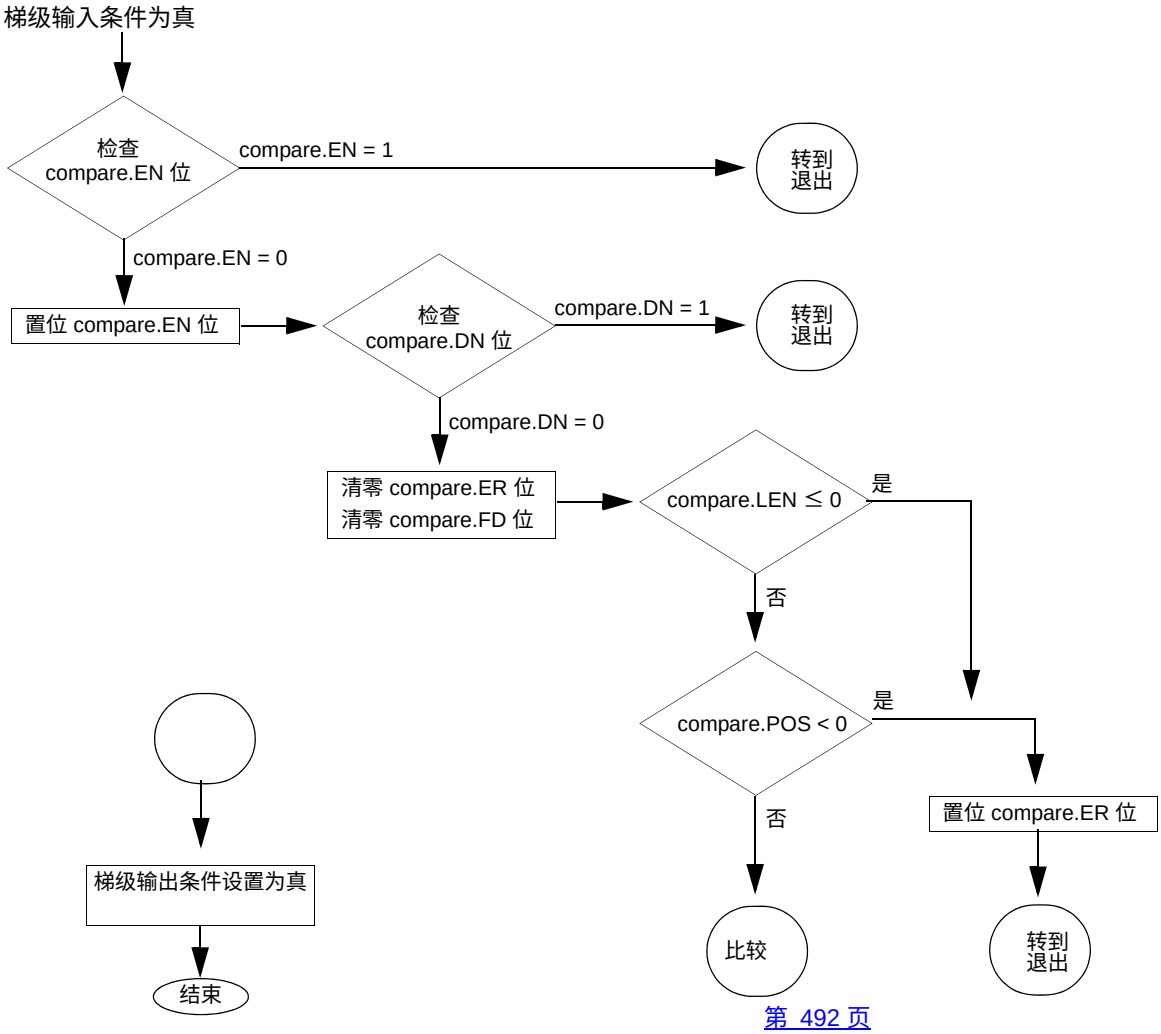
出现主要故障的条件	故障类型	故障代码
Result.POS > 结果数组尺寸	4	20

执行：

条件	梯形图操作
预扫描	

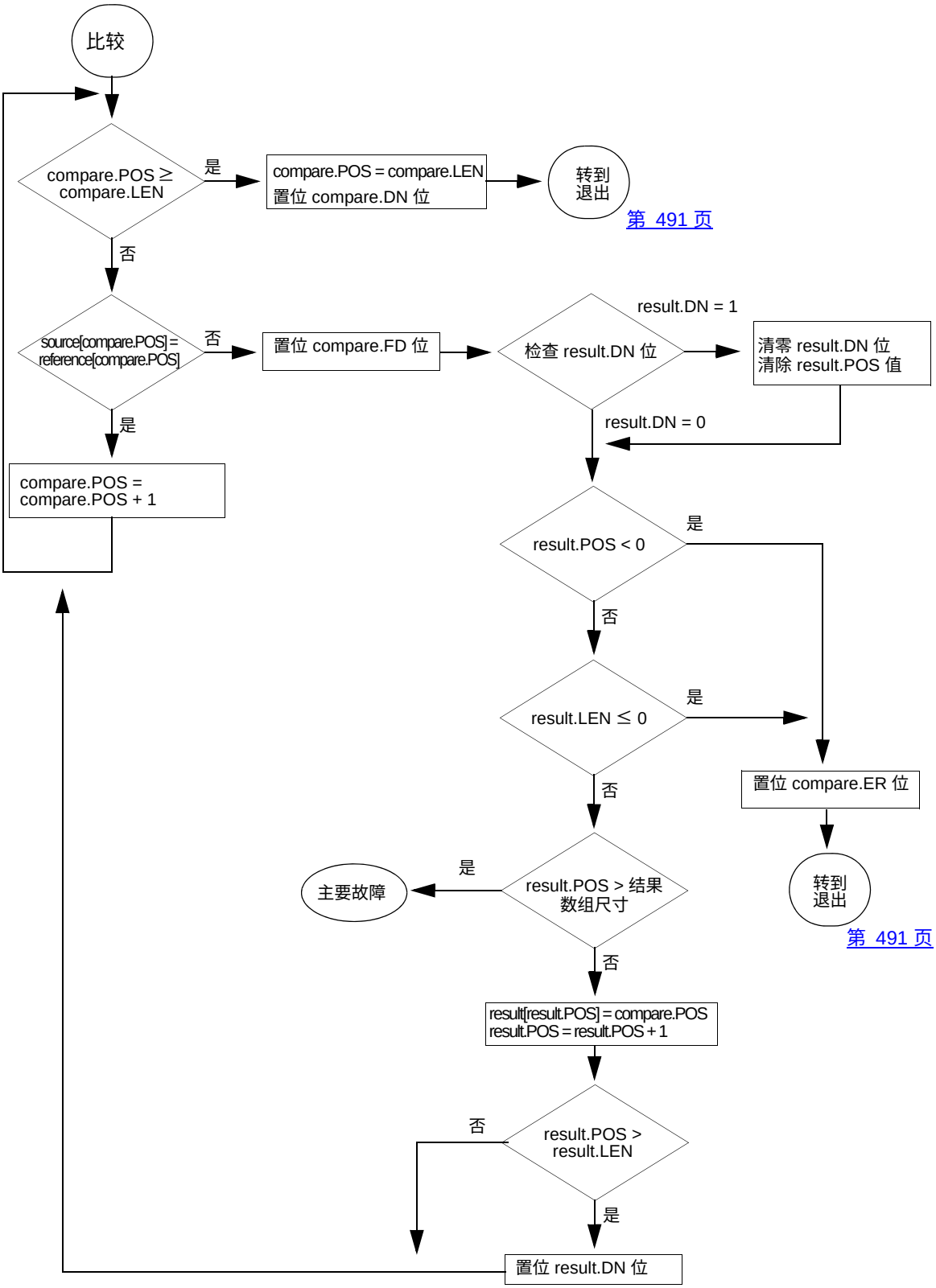
条件	梯形图操作
梯级输入条件为假	

条件	梯形图操作



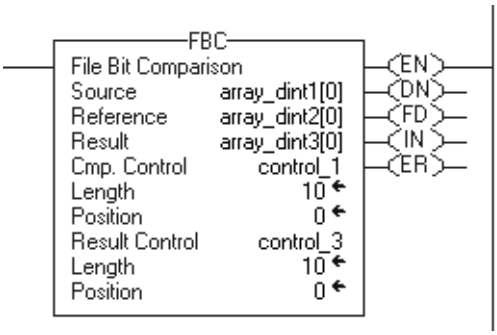
第 492 页

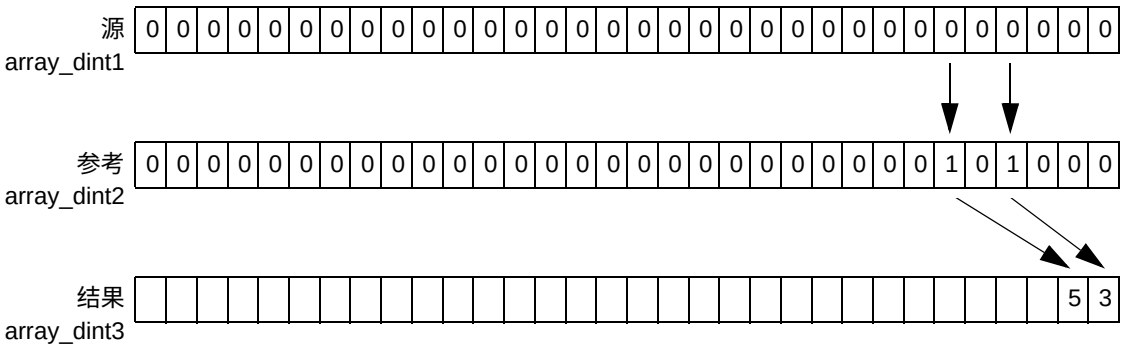
条件	梯形图操作



后扫描	梯级输出条件设置为假。

示例：　使能后，FBC 指令比较源 *array_dint1* 与参考 *array_dint2*，并将所有不匹配项的位置存储在结果 *array_dint3* 中。



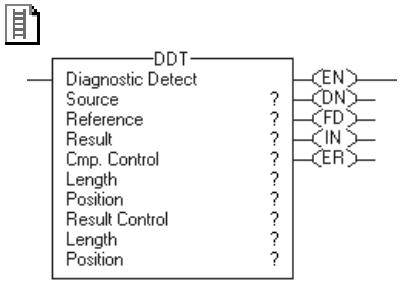


诊断检测 (DDT)

DDT 指令将源数组中的位与参考数组中的位进行比较，以确定状态的更改。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	DINT	数组标签	与参考进行比较的数组 **不要**在下标中使用 CONTROL.POS
参考 (Reference)	DINT	数组标签	与源进行比较的数组 **不要**在下标中使用 CONTROL.POS
结果 (Result)	DINT	数组标签	存储结果的数组 **不要**在下标中使用 CONTROL.POS
比较控制 (Cmp control)	CONTROL	结构	比较的控制结构
长度 (Length)	DINT	立即数	要比较的位数
位置 (Position)	DINT	立即数	源中的当前位置 初始值通常为 0
结果控制 (Result control)	CONTROL	结构	结果的控制结构
长度 (Length)	DINT	立即数	结果中的存储位置数
位置 (Position)	DINT	立即数	结果中的当前位置 初始值通常为 0

注 意

比较控制结构和结果控制结构要使用不同的标签。对这两个结构使用同一个标签可能会导致不可预知的操作，这可能引起设备损坏和 / 或人身伤害。

COMPARE 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示 DDT 指令是否使能。
.DN	BOOL	当 DDT 指令比较 Source 和 Reference 数组中的最后一位后，完成位置位。
.FD	BOOL	DDT 指令每次记录不匹配项 (每次记录一个) 或记录所有不匹配项 (每次扫描全部记录) 后，发现位置位。
.IN	BOOL	禁止位指示 DDT 搜索模式。 0 = “ 所有 ” 模式 1 = “ 每次一个不匹配项 ” 模式
.ER	BOOL	当 compare.POS < 0、compare.LEN < 0、result.POS < 0 或 result.LEN < 0 时，错误位置位。指令停止执行，直到程序清零 .ER 位。
.LEN	DINT	长度值标识要比较的位数。
.POS	DINT	位置值标识当前位。

RESULT 结构

助记符	数据类型	说明
.DN	BOOL	结果数组已满时，完成位置位。
.LEN	DINT	长度值标识结果数组中的存储位置数。
.POS	DINT	位置值标识结果数组中的当前位置。

说明： 使能后， DDT 指令将源数组中的位与参考数组中的位进行比较，在结果数组中记录各个不匹配项的位号，并更改参考位的值使其与对应源位的值匹配。

重要事项

必须测试并确认指令不会更改您不希望更改的数据。

DDT 指令对连续内存进行操作。在某些情况下，该指令会越过数组搜索或写入标签的其它子元素。如果 length 过大且标签是用户自定义的数据类型，就会发生这种情况。

DDT 和 FBC 指令的区别是：DDT 指令每次发现不匹配项时，其会更改参考位使其与源位匹配。而 FBC 指令不更改参考位。

选择搜索模式

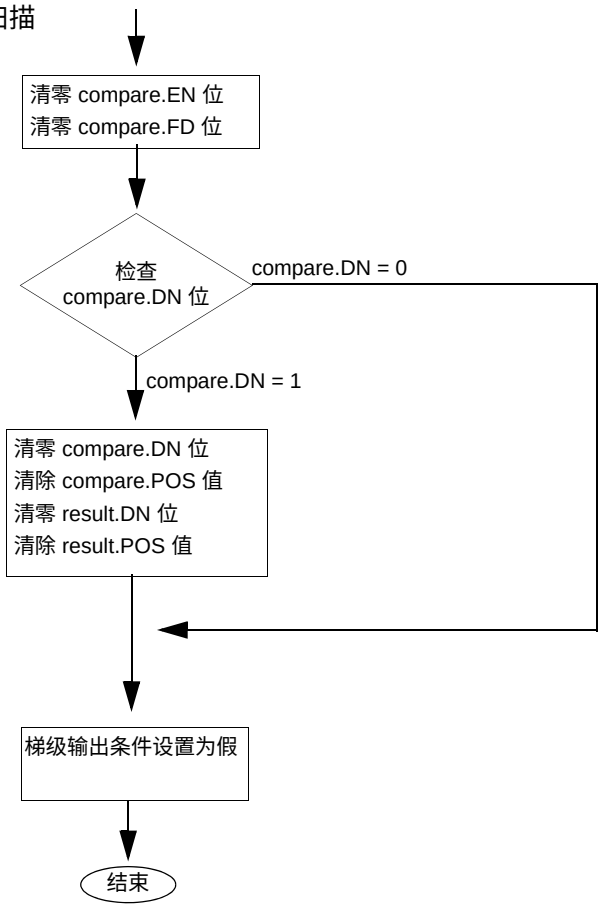
如果要	选择以下模式
每次检测一个不匹配项	置位比较 CONTROL 结构中的 .IN 位。 每次梯级输入条件从假变为真时， DDT 指令都会搜索源数组和参考数组之间的下一个不匹配项。找到不匹配项后，该指令会置位 .FD 位，记录该不匹配项的位置，并停止执行。
检测所有不匹配项	清零比较 CONTROL 结构中的 .IN 位。 每次梯级输入条件从假变为真时， DDT 指令都会搜索源数组和参考数组之间的所有不匹配项。

算术状态标志： 不受影响

故障条件：

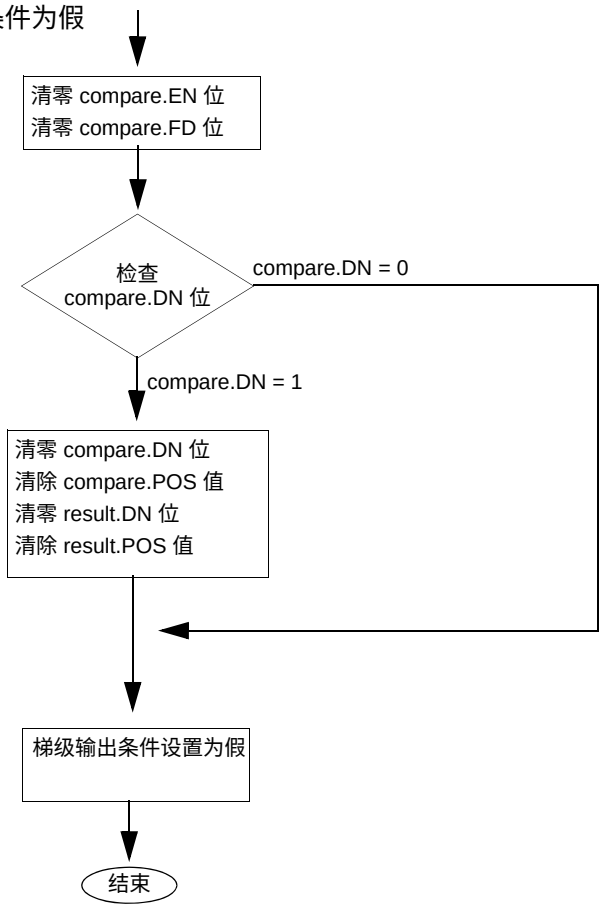
出现主要故障的条件	故障类型	故障代码
Result.POS > 结果数组尺寸	4	20

执行：

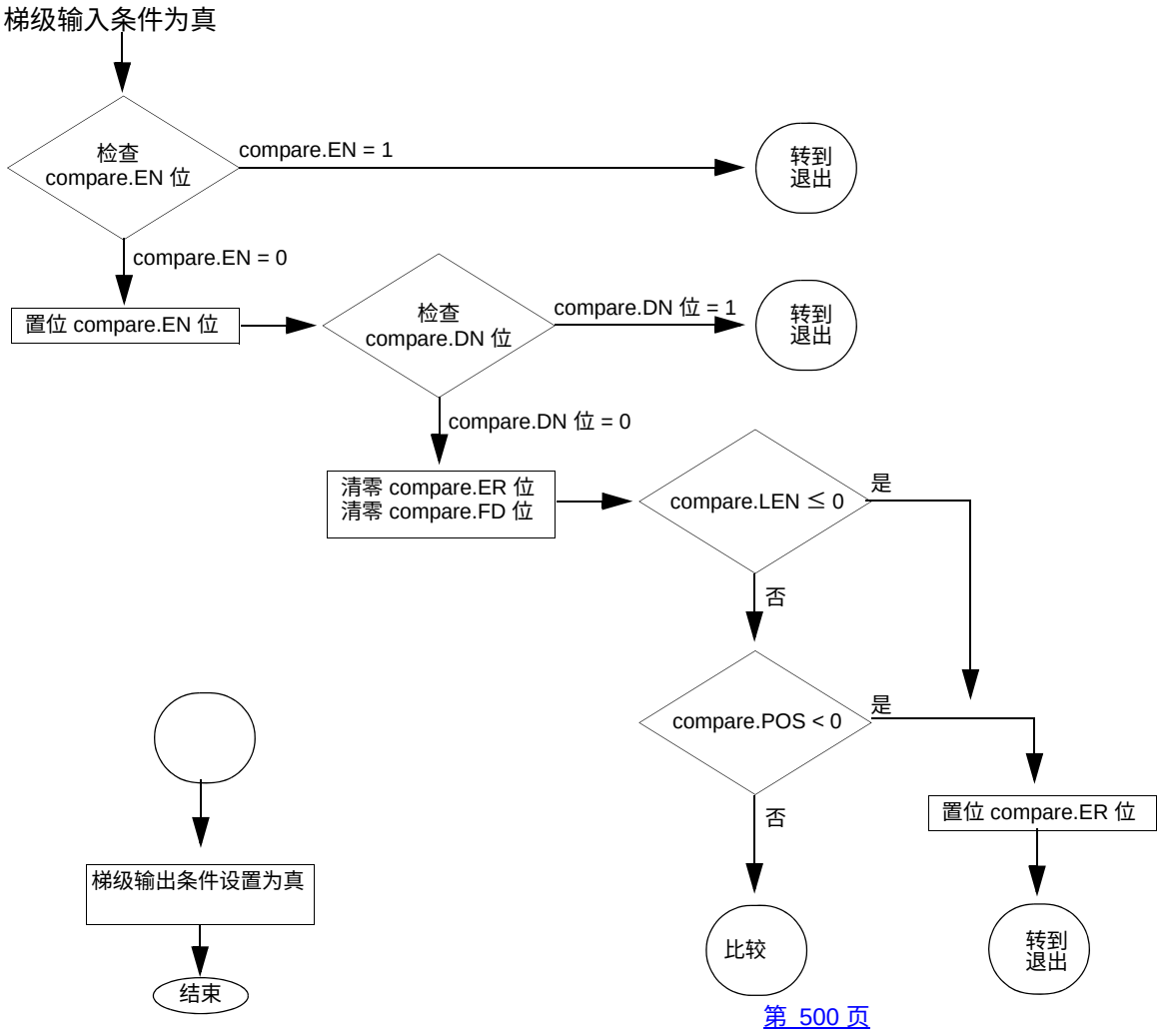
条件：	梯形图操作
预扫描	

条件：	梯形图操作

梯级输入条件为假

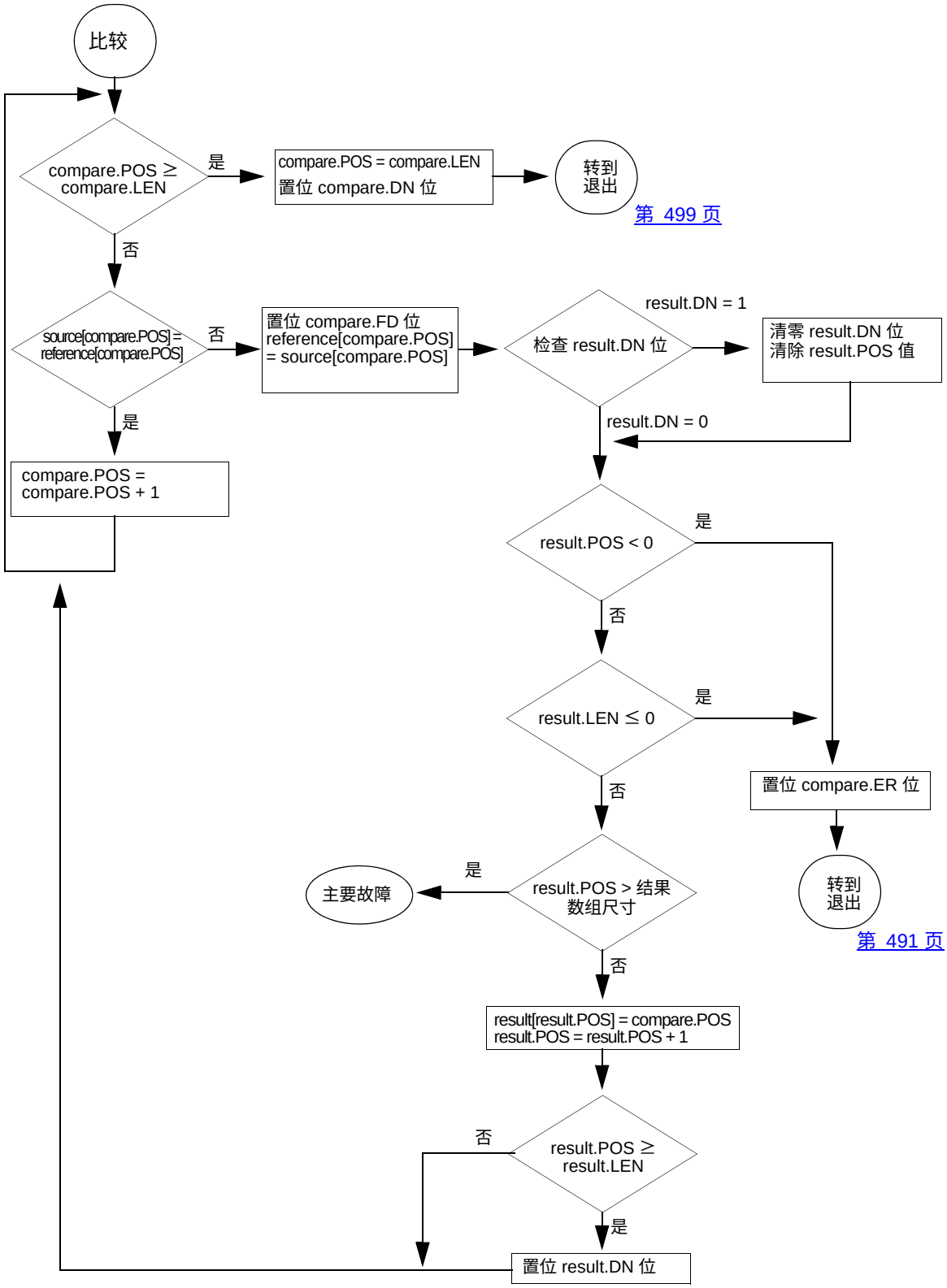


条件：	梯形图操作



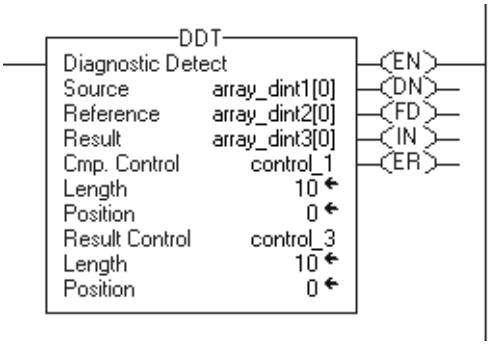
第 500 页

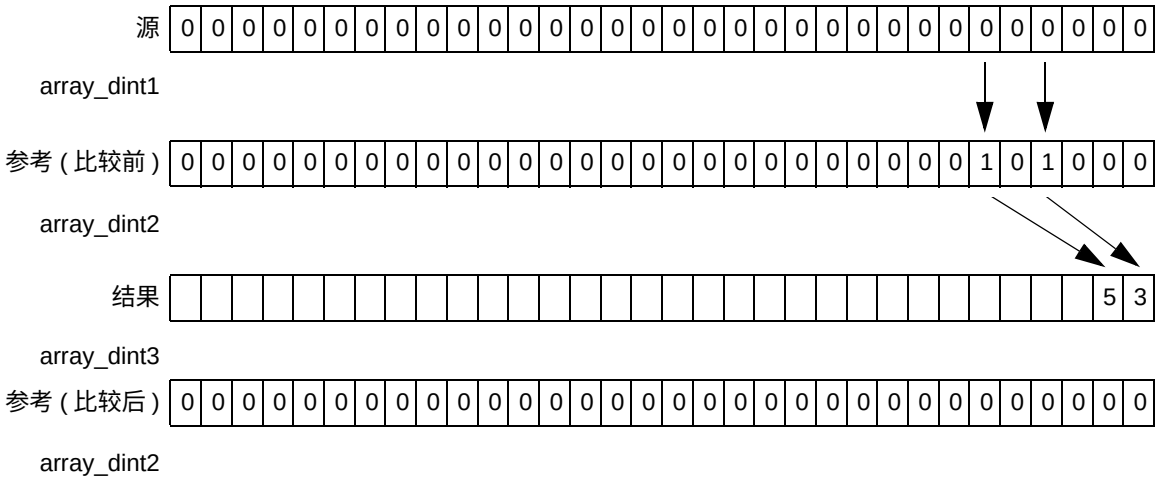
条件：	梯形图操作



后扫描	梯级输出条件设置为假。

示例：　使能后，DDT 指令比较源 *array_dint1* 与参考 *array_dint2*，并将所有不匹配项的位置存储在结果 *array_dint3* 中。控制器还会更改参考 *array_dint2* 中的不匹配位使其与源 *array_dint1* 相匹配。



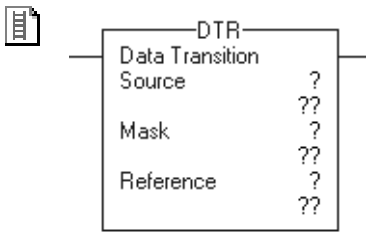


数据转换 (DTR)

DTR 指令通过屏蔽后传递源值并将结果与参考值进行比较。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	DINT	立即数 标签	与参考进行比较的数组
屏蔽码 (Mask)	DINT	立即数 标签	要阻止或传递的位
参考 (Reference)	DINT	标签	与源进行比较的数组

说明：

DTR 指令通过屏蔽后传递源值并将结果与参考值进行比较。 DTR 还将屏蔽后的源值写入参考值，供下一次比较使用。源保持不变。

屏蔽码中的“ 1 ”表示传递该数据位。屏蔽码中的“ 0 ”表示阻止该数据位。

如果屏蔽后的源值与参考值不同，梯级输出条件将变为真，并保持一次扫描的时间。如果屏蔽后的源值与参考值相同，梯级输出条件为假。

注 意

使用此指令进行在线编程会引起危险。如果参考值与源值不同，梯级输出条件将变为真。如果在控制器处于运行或远程运行模式时插入该指令，则需要小心谨慎操作。

键入立即数屏蔽码值

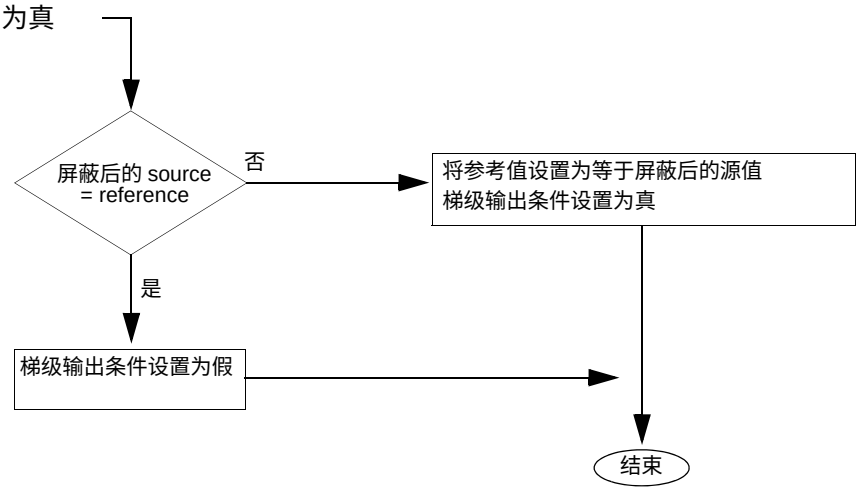
键入屏蔽码时，编程软件将默认为十进制值。如果想使用另一种格式键入屏蔽码，请在值的前面加上正确的前缀。

前缀	说明
16#	十六进制 例如， 16#0F0F
8#	八进制 例如， 8#16
2#	二进制 例如， 2#00110011

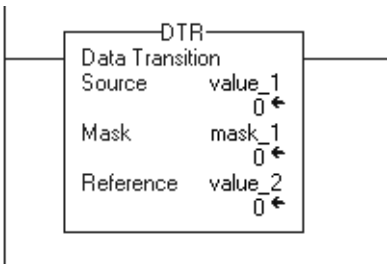
算术状态标志： 不受影响

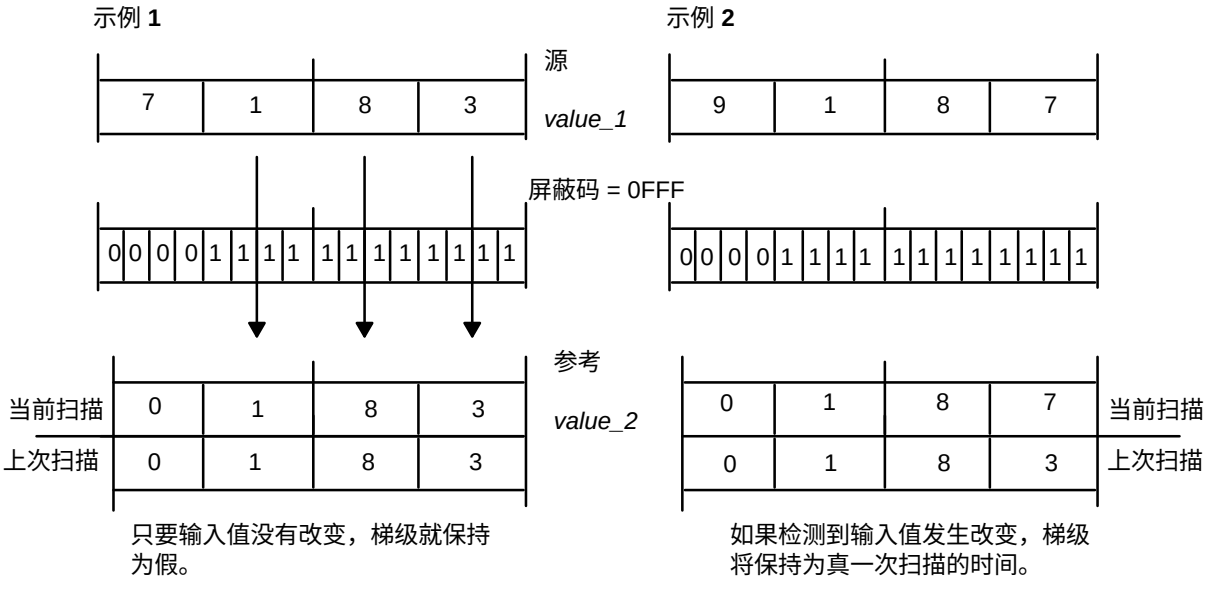
故障条件： 无

执行：

条件	梯形图操作
预扫描	Reference = Source AND Mask。 梯级输出条件设置为假。
梯级输入条件为假	Reference = Source AND Mask。 梯级输出条件设置为假。
梯级输入条件为真	
后扫描	梯级输出条件设置为假。

示例： 使能后， DTR 指令屏蔽 *value_1*。如果两个值不同，梯级输出条件设置为真。





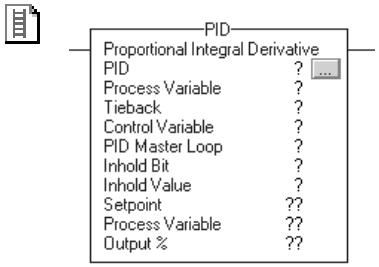
屏蔽码中的 0 使相应位保持不变。

比例积分微分 **(PID)**

PID 指令控制流量、压力、温度或料位等过程变量。

操作数：

梯形图



操作数	类型	格式	说明
PID	PID	结构	PID 结构
过程变量 (Process variable)	SINT INT DINT REAL	标签	要控制的值
跟随值 (Tieback)	SINT INT DINT REAL	立即数 标签	(可选) 硬件手动 / 自动站点的输出，它旁路控制器的输出 如果不想使用此参数，则输入 0。
控制变量 (Control variable)	SINT INT DINT REAL	标签	输出到最终控制设备 (阀门、气闸等) 的值 如果要使用死区， Control variable 的格式必须为 REAL，否则当误差处于死区内时，该值将被强制为 0。
PID 主回路 (PID master loop)	PID	结构	(可选) 主 PID 的 PID 标签 如果要执行串级控制且该 PID 是从回路，则输入主 PID 的名称。如果不想使用此参数，则输入 0。
保持位 (Inhold bit)	BOOL	标签	(可选)1756 模拟量输出通道的保持位的当前状态，用于支持无扰动重新启动 如果不想使用此参数，则输入 0。
保持值 (Inhold value)	SINT INT DINT REAL	标签	(可选)1756 模拟量输出通道的数据读回值，用于支持无扰动重新启动 如果不想使用此参数，则输入 0。
设定值 (Setpoint)			显示设定值的当前值
过程变量 (Process variable)			显示定标的过程变量的当前值
输出百分比 (Output %)			显示当前输出百分比值

```
PID(PID,ProcessVariable,
Tieback,ControlVariable,
PIDMasterLoop,InholdBit,
InHoldValue);
```

结构化文本

操作数与梯形图 PID 指令的操作数相同。但是，指定 Setpoint、Process Variable 和 Output % 的方式是访问 PID 结构的 .SP、.PV 和 .OUT 子元素，而不是将值包括在操作数列表中。

PID 结构

助记符：	数据类型	说明	
.CTL	DINT	.CTL 子元素访问一个 32 位字中的状态子元素 (位)。 PID 指令置位位 07 到 15。	
		位	对应子元素
		31	.EN
		30	.CT
		29	.CL
		28	.PVT
		27	.DOE
		26	.SWM
		25	.CA
		24	.MO
		23	.PE
		22	.NDF
		21	.NOBC
		20	.NOZC
		位	**PID** 指令置位的对应子元素
		15	.INI
		14	.SPOR
		13	.OLL
		12	.OLH
		11	.EWD
		10	.DVNA
		09	.DVPA
		08	.PVLA
		07	.PVHA
.SP	REAL	设定值	
.KP	REAL	独立	比例增益 (无单位)
		相关	控制器增益 (无单位)
.KI	REAL	独立	积分增益 (1/ 秒)
		相关	复位时间 (分 / 循环)

助记符：	数据类型	说明	
.KD	REAL	独立	微分增益 (秒)
		相关	比率时间 (分)
.BIAS	REAL	前馈或偏置百分比	
.MAXS	REAL	最大工程单位定标值	
.MINS	REAL	最小工程单位定标值	
.DB	REAL	死区工程单位	
.SO	REAL	设置输出百分比	
.MAXO	REAL	最大输出限值 (输出百分比)	
.MINO	REAL	最小输出限值 (输出百分比)	
.UPD	REAL	回路更新时间 (秒)	
.PV	REAL	已定标的 PV 值	
.ERR	REAL	已定标的误差值	
.OUT	REAL	输出百分比	
.PVH	REAL	过程变量报警上限	
.PVL	REAL	过程变量报警下限	
.DVP	REAL	正偏差报警限值	
.DVN	REAL	负偏差报警限值	
.PVDB	REAL	过程变量报警死区	
.DVDB	REAL	偏差报警死区	
.MAXI	REAL	最大 PV 值 (未定标输入)	
.MINI	REAL	最小 PV 值 (未定标输入)	
.TIE	REAL	手动控制的跟随值	
.MAXCV	REAL	最大 CV 值 (对应于 100%)	
.MINCV	REAL	最小 CV 值 (对应于 0%)	
.MINTIE	REAL	最小跟随值 (对应于 100%)	
.MAXTIE	REAL	最大跟随值 (对应于 0%)	

助记符：	数据类型	说明	
.DATA	REAL[17]	.DATA 子元素存储：	
		元素	说明
		.DATA[0]	积分累加值
		.DATA[1]	微分平滑临时值
		.DATA[2]	前次 .PV 值
		.DATA[3]	前次 .ERR 值
		.DATA[4]	前次有效 .SP 值
		.DATA[5]	百分比定标常数
		.DATA[6]	.PV 定标常数
		.DATA[7]	微分定标常数
		.DATA[8]	前次 .KP 值
		.DATA[9]	前次 .KI 值
		.DATA[10]	前次 .KD 值
		.DATA[11]	相关增益 .KP
		.DATA[12]	相关增益 .KI
		.DATA[13]	相关增益 .KD
		.DATA[14]	前次 .CV 值
		.DATA[15]	.CV 的定标系数
		.DATA[16]	跟随的定标系数
.EN	BOOL	已使能	
.CT	BOOL	串级类型 (0 = 从； 1 = 主)	
.CL	BOOL	串级回路 (0 = 否； 1 = 是)	
.PVT	BOOL	过程变量跟踪 (0 = 否； 1 = 是)	
.DOE	BOOL	微分对象 (0 = PV ； 1 = 误差)	
.SWM	BOOL	软件手动模式 (0 = 否 [自动] ； 1 = 是 [软件手动])	
.CA	BOOL	控制操作 (0 表示 E=SP-PV ； 1 表示 E=PV-SP)	
.MO	BOOL	站点模式 (0 = 自动； 1 = 手动)	
.PE	BOOL	PID 等式 (0 = 独立； 1 = 相关)	
.NDF	BOOL	禁止微分平滑 (0 = 使能微分平滑滤波器； 1 = 禁止微分平滑滤波器)	
.NOBC	BOOL	禁止偏置反向计算 (0 = 使能偏置反向计算； 1 = 禁止偏置反向计算)	
.NOZC	BOOL	禁止过零死区 (0 = 死区是过零死区； 1 = 死区不是过零死区)	
.INI	BOOL	PID 已初始化 (0 = 否； 1 = 是)	
.SPOR	BOOL	设定值超出范围 (0 = 否； 1 = 是)	

助记符：	数据类型	说明
.OLL	BOOL	CV 低于输出下限 (0 = 否； 1 = 是)
.OLH	BOOL	CV 高于输出上限 (0 = 否； 1 = 是)
.EWD	BOOL	误差在死区内 (0 = 否； 1 = 是)
.DVNA	BOOL	偏差下限报警 (0 = 否； 1 = 是)
.DVPA	BOOL	偏差上限报警 (0 = 否； 1 = 是)
.PVLA	BOOL	PV 下限报警 (0 = 否； 1 = 是)
.PVHA	BOOL	PV 上限报警 (0 = 否； 1 = 是)

说明： PID 指令通常从模拟量输入模块接收过程变量 (PV)，并调节模拟量输出模块的控制变量输出 (CV)，从而使过程变量保持为期望的设定值。

.EN 位指示执行状态。当梯级输入条件从假转变为真时， .EN 位置位。当梯级输入条件变为假时， .EN 位清零。 PID 指令不使用 .DN 位。只要梯级输入条件为真，每次扫描 PID 指令都执行。

算术状态标志： 不受影响

故障条件：

重要事项 这些故障是 PLC-5 控制器的主要故障。

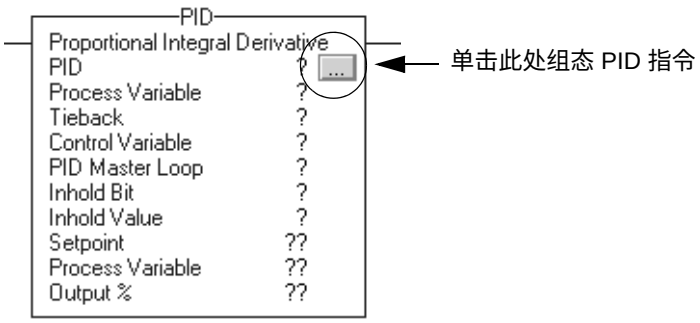
出现次要故障的条件	故障类型	故障代码
.UPD ≤ 0	4	35
设定值超出范围	4	36

执行：

条件	操作	操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	指令执行 PID 回路。	指令执行 PID 回路。
后扫描	梯级输出条件设置为假。	不执行任何操作。

组态 PID 指令

键入 PID 指令并指定 PID 结构后，使用组态页面指定 PID 指令如何工作。



指定整定

选择“ 调试 ”(Tuning) 页面。单击其它字段、单击“ 确定 ”(OK)、单击“ 应用 ”(Apply) 或按回车键后，更改将生效。

字段	指定
设定值 (SP) (Setpoint (SP))	键入设定值 (.SP)。
设置输出百分比 (Set output %)	键入设置输出百分比 (.SO)。 在软件手动模式下，此值用于输出。 在自动模式下，此值显示输出百分比。
输出偏置 (Output bias)	键入输出偏置百分比 (.BIAS)。
比例增益 (Kp) (Proportional gain (K_p))	键入比例增益 (.KP)。 对于独立增益，它是比例增益 (无单位)。 对于相关增益，它是控制器增益 (无单位)。
积分增益 (Ki) (Integral gain (K_i))	键入积分增益 (.KI)。 对于独立增益，它是积分增益 (1/ 秒)。 对于相关增益，它是复位时间 (分 / 循环)。
微分时间 (Kd) (Derivative time (K_d))	键入微分增益 (.KD)。 对于独立增益，它是微分增益 (秒)。 对于相关增益，它是比率时间 (分)。
手动模式	选择手动 (.MO) 或软件手动 (.SWM)。 如果同时选择这二者，手动模式将优先于软件手动模式。

指定组态

选择“ 组态 ”(Configuration) 页面。单击“ 确定 ”(OK) 或“ 应用 ”(Apply) 后更改才会生效。

字段	指定
PID 等式 (PID equation)	选择独立增益或相关增益 (.PE)。 如果希望三个增益 (P、 I 和 D) 独立操作，则选择独立增益。如果需要可影响所有这三项 (P、 I 和 D) 的总控制器增益，则使用相关增益。
控制操作 (Control action)	选择 E=PV-SP 或 E=SP-PV 作为控制方向 (.CA)。
微分对象 (Derivative of)	选择 PV 或误差 (.DOE)。 使用 PV 的微分可消除设定值变化引起的输出尖峰。如果算法可以容忍超调量，使用误差的微分可快速响应设定值变化。
回路更新时间 (Loop update time)	键入指令的更新时间 (.UPD)。
CV 上限 (CV high limit)	键入控制变量的上限 (.MAXO)。(1)
CV 下限 (CV low limit)	键入控制变量的下限 (.MINO)。 (1)
死区值 (Deadband value)	键入死区值 (.DB)。
禁止微分平滑 (No derivative smoothing)	使能或禁止此选项 (.NDF)。
禁止偏置计算 (No bias calculation)	使能或禁止此选项 (.NOBC)。
禁止过零死区 (No zero crossing in deadband)	使能或禁止此选项 (.NOZC)。
PV 跟踪 (PV tracking)	使能或禁止此选项 (.PVT)。
串级回路 (Cascade loop)	使能或禁止此选项 (.CL)。
串级类型 (Cascade type)	如果使能了串级回路，则选择从或主 (.CT)。

(1) 使用基于梯形图的 PID 指令时，如果设置 MAXO = MINO， PID 指令会将这些值复位为默认值。 MAXO = 100.0 且 MINO = 0.0

指定报警

选择“ 报警 ”(Alarm) 页面。单击“ 确定 ”(OK) 或“ 应用 ”(Apply) 后更改才会生效。

字段	指定
PV 上限 (PV high)	键入 PV 报警上限值 (.PVH)。
PV 下限 (PV low)	键入 PV 报警下限值 (.PVL)。
PV 死区 (PV deadband)	键入 PV 报警死区值 (.PVDB)。
正偏差 (Positive deviation)	键入正偏差值 (.DVP)。
负偏差 (Negative deviation)	键入负偏差值 (.DVN)。
偏差死区 (Deviation deadband)	键入偏差报警死区值 (.DVDB)。

指定定标

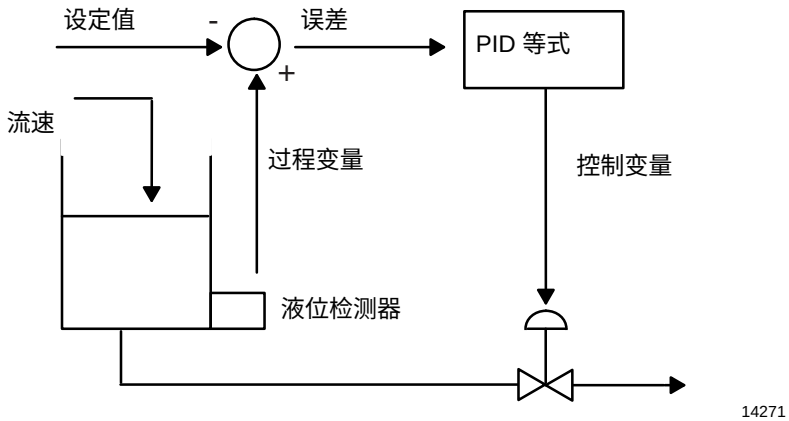
选择“ 定标 ”(Scaling) 页面。单击“ 确定 ”(OK) 或“ 应用 ”(Apply) 后更改才会生效。

字段	指定
PV 未定标最大值 (PV unscaled maximum)	键入最大 PV 值 (.MAXI)，该值等于从 PV 值的模拟量输入通道接收的最大未定标值。
PV 未定标最小值 (PV unscaled minimum)	键入最小 PV 值 (.MINI)，该值等于从 PV 值的模拟量输入通道接收的最小未定标值。
PV 工程单位最大值 (PV engineering units maximum)	键入与 .MAXI 相对应的最大工程单位 (.MAXS)(1)
PV 工程单位最小值 (PV engineering units minimum)	键入与 .MINI 相对应的最小工程单位 (.MINS)(1)
CV 最大值 (CV maximum)	键入与 100% 相对应的最大 CV 值 (.MAXCV)。
CV 最小值 (CV minimum)	键入与 0% 相对应的最小 CV 值 (.MINCV)。
最大跟随值 (Tieback maximum)	键入最大跟随值 (.MAXTIE)，该值等于从跟随值的模拟量输入通道接收的最大未定标值。
最小跟随值 (Tieback minimum)	键入最小跟随值 (.MINTIE)，该值等于从跟随值的模拟量输入通道接收的最小未定标值。
PID 已初始化 (PID Initialized)	如果在运行模式下更改定标常数，则将关闭此选项，这样将重新初始化内部缩小定标值 (.INI)。

(1) 使用基于梯形图的 PID 指令时，如果设置 MAXO = MINO， PID 指令会将这些值复位为默认值。 MAXO = 100.0 且 MINO = 0.0

使用 **PID** 指令

PID 闭环控制使过程变量维持为期望的设定值。下图显示了一个流速 / 液位的示例。



在上例中，将罐中的液位与设定值相比较。如果液位大于设定值，PID 等式将增大控制变量，并使罐的出口阀打开；从而降低罐中的液位。

PID 指令中使用的 PID 等式是一个可以选择使用独立增益或相关增益的位置形式等式。使用独立增益时，比例、积分和微分增益仅分别影响其特定的比例、积分或微分项。使用相关增益时，比例增益替换为影响所有这三项的控制器增益。可以使用任何一种形式的等式来执行同类型的控制。提供这两种等式的目的是，用户可以选择使用自己熟悉的那种等式。

增益选项	微分对象	公式
相关增益 (ISA 标准)	误差 (E)	$$CV = K_C\left[E + \frac{1}{T_i}\int_0^t Edt + T_d\frac{dE}{dt}\right] + BIAS$$
	过程变量 (PV)	E = SP - PV $$CV = K_C\left[E + \frac{1}{T_i}\int_0^t Edt - T_d\frac{dPV}{dt}\right] + BIAS$$ E = PV - SP $$CV = K_C\left[E + \frac{1}{T_i}\int_0^t Edt + T_d\frac{dPV}{dt}\right] + BIAS$$
独立增益	误差 (E)	$$CV = K_PE + K_i\int_0^t Edt + K_d\frac{dE}{dt} + BIAS$$
	过程变量 (PV)	E = SP - PV $$CV = K_PE + K_i\int_0^t Edt - K_d\frac{dPV}{dt} + BIAS$$ E = PV - SP $$CV = K_PE + K_i\int_0^t Edt + K_d\frac{dPV}{dt} + BIAS$$

其中：

变量	说明
K_P	比例增益 (无单位) $K_p = K_c$(无单位)
K_i	积分增益 (秒 $^{-1}$) 要在 K_i(积分增益) 和 T_i(复位时间) 之间进行转换，请使用： $K_i = \frac{K_C}{60\,T_i}$
K_d	微分增益 (秒) 要在 K_d(微分增益) 和 T_d(比率时间) 之间进行转换，请使用： $K_d = K_c\,(T_d)\,60$
K_C	控制器增益 (无单位)
T_i	复位时间 (分 / 循环)
T_d	比率时间 (分)
SP	设定值
PV	过程变量
E	误差 [(SP-PV) 或 (PV-SP)]
BIAS	前馈或偏置
CV	控制变量
dt	回路更新时间

如果不想使用 PID 等式的特定项，只需将其增益设置为零。例如，如果不需要微分作用，则将 K_d 或 T_d 设置为零。

抗积分饱和与从手动模式到自动模式的无扰动转换

PID 指令通过防止在 CV 输出达到由 .MAXO 和 .MINO 对其设置的最大值或最小值时积分项进行累加，来自动避免积分饱和。累加的积分项保持冻结，直到 CV 输出下降到低于其上限或上升到高于其下限。随后正常的积分累加将自动恢复。

PID 指令支持两种手动控制模式。

手动控制模式	说明
软件手动 (.SWM)	也称为设置输出模式 用户可以通过软件设置输出百分比 设置输出 (.SO) 值用作回路输出。设置输出值通常来自操作员接口设备的操作员输入。
手动 (.MO)	将跟随值作为输入，并调整其内部变量以在输出中生成相同的值。 根据 .MINTIE 和 .MAXTIE 值，将 PID 指令的跟随输入定标为 0-100%，并将其用作回路的输出。跟随输入通常来自旁路控制器输出的硬件手动 / 自动站点的输出。 重要说明：如果这两个模式位都设置为开，手动模式将优先于软件手动模式。

PID 指令还自动提供从软件手动模式到自动模式，或从手动模式到自动模式的无扰动转换。 PID 指令反向计算积分累加项的值， CV 输出跟踪软件手动模式下的设置输出 (.SO) 值和手动模式下的跟随输入需要该值。这样，当回路切换到自动模式时， CV 输出从设置输出或跟随值开始，因此输出值不会发生 “ 扰动 ”。

即使没有使用积分控制 (即 $K_i = 0$)， PID 指令也可自动提供从手动模式到自动模式的无扰动转换。在这种情况下，指令修改 .BIAS 项，使 CV 输出跟踪设置输出或跟随值。恢复自动控制后， .BIAS 项将保持其最后的值。可以通过置位 PID 数据结构中的 .NOBC 位来禁止对 .BIAS 项的反向运算。请注意，如果将 .NOBC 设置为真，不使用积分控制时， PID 指令将不再提供从手动模式到自动模式的无扰动转换。

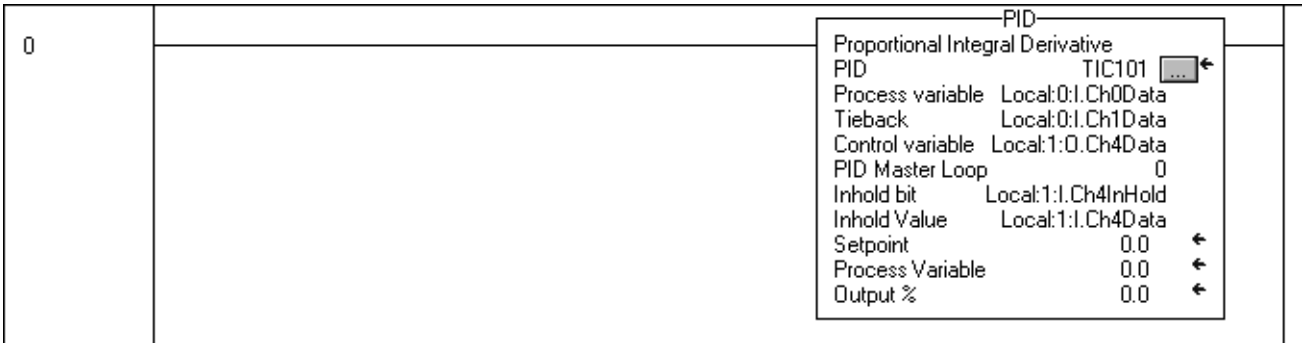
PID 指令计时

需要对 PID 指令和过程变量的采样进行定期更新。该更新时间与所控制的物理过程有关。对于非常慢的回路 (如温度回路)，每秒一次甚至更长的更新时间通常足以获得良好的控制效果。对于更快的回路 (如压力或流量回路)，可能需要诸如每 250 ms 一次这样的更新时间。仅少数情况 (例如，开卷机卷轴上的张力控制) 需要每隔 10 ms 或更快地更新回路。

由于 PID 指令在计算中使用时基，所以需要同步该指令的执行与过程变量 (PV) 的采样。

执行 PID 指令的最简单方法是将 PID 指令置于周期性任务中。将回路更新时间 (.UPD) 设置为等于周期性任务速率，并确保每次扫描周期性任务时都执行 PID 指令。

梯形图



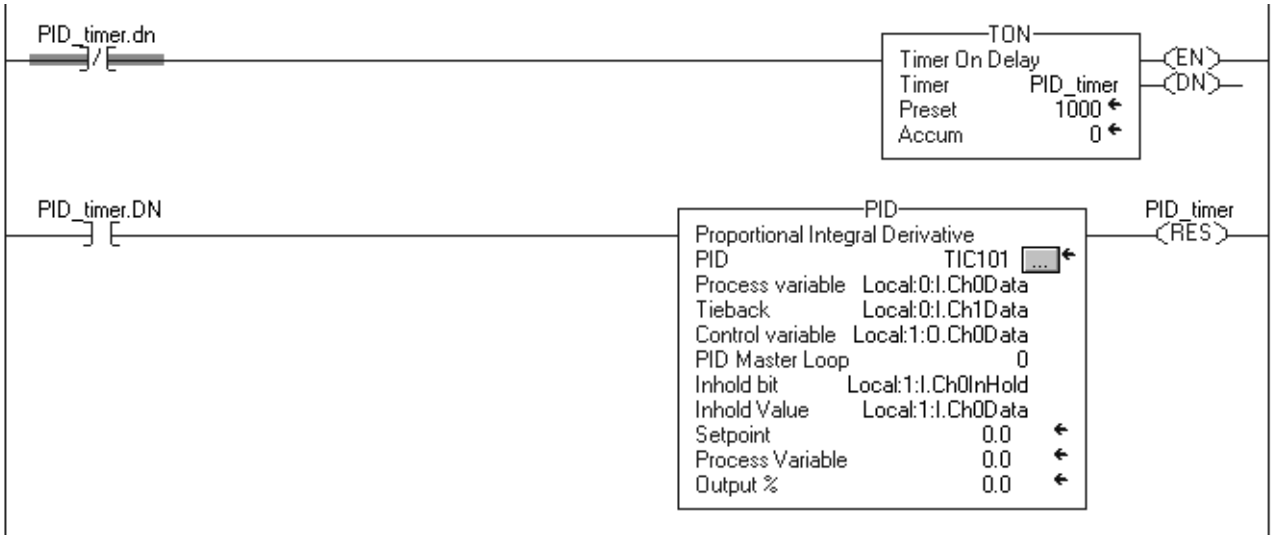
结构化文本

```
PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
     Local:1:O.Ch4Data,0,Local:1:I.Ch4InHold,
     Local:1:I.Ch4Data);
```

使用周期性任务时，确保采用比周期性任务速率明显快的速率将用于过程变量的模拟量输入更新到处理器。理想情况下，至少应使用比周期性任务速率快 5...10 倍的速率将过程变量发送到控制器。这样可最大限度降低过程变量的实际采样和 PID 回路执行之间的时间差。例如，如果 PID 回路位于 250 ms 的周期性任务中，则使用 250 ms 的回路更新时间 (.UPD = .25)，并组态模拟量输入模块至少每隔 25 ms 到 50 ms 左右生成一次数据。

另一种执行 PID 指令的方法 (精确性略低) 是将该指令置于连续任务中，并使用计时器完成位来触发 PID 指令的执行。

梯形图



结构化文本

```
PID_timer.pre := 1000

TONR(PID_timer);

IF PID_timer.DN THEN

     PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
     Local:1:O.Ch0Data,0,Local:1:I.Ch0InHold,
     Local:1:I.Ch0Data);

END_IF;
```

在此方法中，应将 PID 指令的回路更新时间设置为等于计时器预设值。与使用周期性任务时一样，应将模拟量输入模块设置为以明显比循环更新时间快的速率生成过程变量。仅当回路的回路更新时间至少比连续任务在最差情况下的执行时间长几倍时，才可使用 PID 执行的计时器方法。

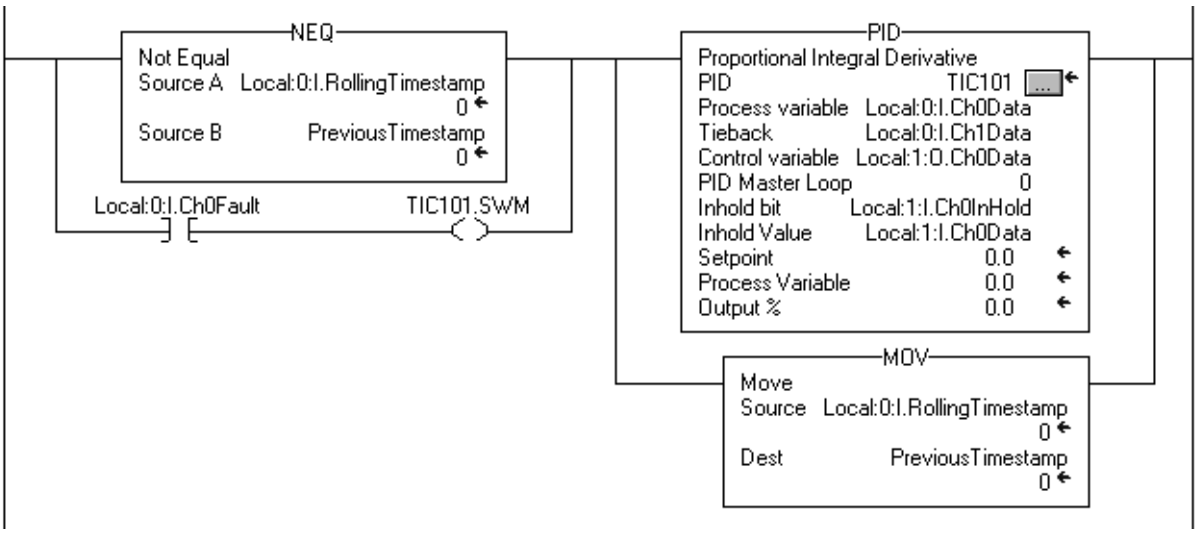
执行 PID 指令的最精确方法是使用 1756 模拟量输入模块的实时采样 (RTS) 功能。模拟量输入模块以设置模块时组态的实时采样速率对其输入进行采样。当模块的实时采样周期结束时，模块将更新其输入并更新模块生成的滚动时间戳 (由模拟量输入数据结构的 RollingTimestamp 子元素表示)。

时间戳的范围是 0 到 32,767 ms。系统监视时间戳。如果时间戳发生更改，则说明收到新的过程变量采样。每次时间戳发生更改时，执行 PID 指令一次。由于过程变量采样由模拟量输入模块来驱动，因此输入采样时间非常准确，应将 PID 指令使用的回路更新时间设置为等于模拟量输入模块的 RTS 时间。

要确保不丢失过程变量的采样，应以比 RTS 时间快的速率执行逻辑。例如，如果 RTS 时间为 250 ms，则可将 PID 逻辑置于每 100 ms 运行一次的周期性任务中，从而确保不会丢失采样。甚至可以将 PID 逻辑置于连续任务中，只要能确保以比每 250 毫秒一次快的频率更新逻辑即可。

下图显示了一个 RTS 执行方法的示例。收到新模拟量输入数据时执行 PID 指令。如果模拟量输入模块发生故障或被拆除，控制器将停止接收滚动时间戳， PID 回路将停止执行。应监视 PV 模拟量输入的状态位，如果其表明状态不良，则将回路强置为软件手动模式，并在每次扫描时执行该回路。这样操作员仍可手动更改 PID 回路的输出。

梯形图



结构化文本

```
IF (Local:0:I.Ch0Fault) THEN
     TIC101.SWM [:=] 1;
ELSE
     TIC101.SWM := 0;
END_IF;

IF (Local:0:I.RollingTimestamp<>PreviousTimestamp) OR
     (Local:0:I.Ch0Fault) THEN

     PreviousTimestamp := Local:0:I.RollingTimestamp;

     PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
     Local:1:O.Ch0Data,0,Local:1:I.Ch0InHold,
     Local:1:I.Ch0Data);

END_IF;
```

无扰动重新启动

PID 指令可与 1756 模拟量输出模块进行交互，从而在控制器从程序模式切换到运行模式时或者在控制器上电时，支持无扰动重新启动。

当 1756 模拟量输出模块与控制器的通信中断时或检测到控制器处于程序模式时，模拟量输出模块会将其输出设置为组态模块时指定的故障条件值。当控制器随后返回到运行模式时或与模拟量输出模块重新建立通信时，可以通过 PID 指令中的 Inhold bit 和 Inhold Value 参数使 PID 指令自动将其控制变量输出复位为等于模拟量输出。

用于设置无扰动重新启动的指令。

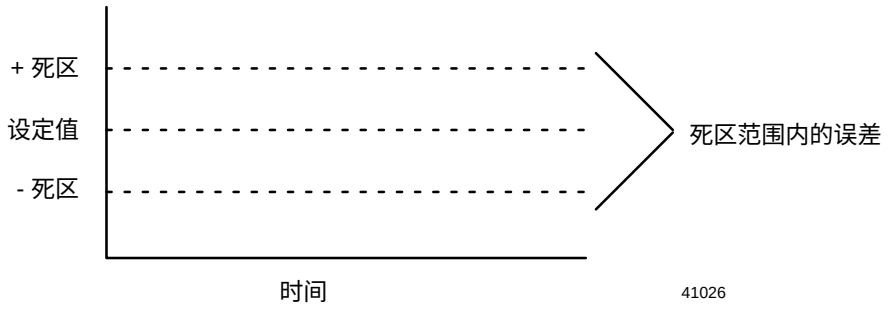
执行如下操作	详细信息
对从 PID 指令接收控制变量的 1756 模拟量输出模块的通道进行组态	在模块的特定通道的属性页中选中“ 保持以进行初始化 ”(hold for initialization) 复选框。 该选项通知模拟量输出模块：当控制器返回到运行模式或与模块重新建立通信时，该模块应将模拟量输出保持为其当前值，直到控制器发送的值与输出通道使用的当前值相匹配 (在变化范围的 0.1% 内)。控制器的输出利用 .BIAS 项以一定的斜率过渡到当前保持的输出值。这种过渡方式类似于自动无扰动转换。
在 PID 指令中输入保持位标签和保持值标签	1756 模拟量输出模块在其输入数据结构中为每个通道返回两个值。保持状态位 (如 .Ch2InHold) 为真，表示模拟量输出通道正在保持其值。数据读回值 (如 .Ch2Data) 以工程单位显示当前输出值。 将保持状态位的标签作为 PID 指令的保持位参数输入。将数据读回值的标签作为保持值参数输入。 当保持位变为真时， PID 指令将保持值传送到控制变量输出，并重新初始化以支持从该值进行无扰动重新启动。当模拟量输出模块接收到从控制器返回的该值时，将关闭保持状态位，从而允许 PID 指令正常启动控制。

微分平滑

微分平滑滤波器加强了微分计算。这种一阶低通数字滤波器有助于最大限度减少由 PV 中的噪音引起的较大微分项尖峰。微分增益值越大，这种平滑作用就越强。如果过程需要非常大的微分增益值 (例如， $K_d > 10$)，则可禁止微分平滑。要禁止微分平滑，请选择“ 组态 ”(Configuration) 页面中的“ 禁止微分平滑 ”(No derivative smoothing) 选项或置位 PID 结构中的 .NDF 位。

设置死区

使用可调整死区，可以在设置值的上下选择一个误差范围，只要误差保持在此范围内，输出就不会改变。使用死区，可以控制过程变量与设置点匹配的接近程度，在该程度内不会改变输出。死区还有助于最大限度减少最终控制设备的磨损。



过零是一种死区控制，当过程变量进入死区时，过零允许指令将误差用于计算，直到过程变量经过设定值。一旦过程变量经过设定值 (误差过零并改变符号)，只要过程变量维持在死区内，输出就不会改变。

死区按您指定的值从设定值向上和向下扩展。输入零可禁止死区。死区的定标单位与设定值的相同。可以通过选择“ 组态 ”(Configuration) 页面的“ 禁止过零死区 ”(no zero crossing for deadband) 选项或置位 PID 结构中的 .NOZC 位，来使用无过零功能的死区。

如果要使用死区， Control variable 的格式必须为 REAL，否则当误差处于死区内时，该值将被强制为 0。

使用输出限制

可以对控制输出设置输出限值 (输出百分比)。当指令检测到输出达到限值时，其将置位报警位并阻止输出超过下限或上限。

前馈或输出偏置

可以通过将 .BIAS 值馈给 PID 指令的前馈 / 偏置值，前馈来自系统的干扰。

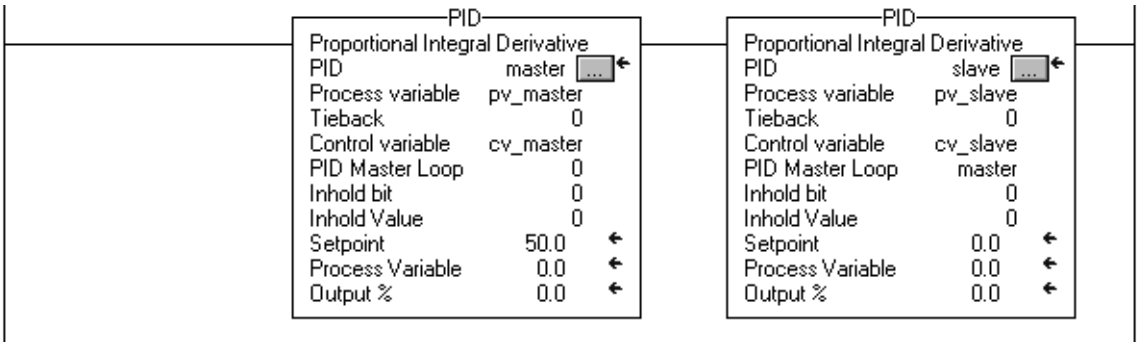
前馈值是改变过程变量之前给予控制变量的提前量。前馈通常用于控制存在传递滞后的过程。例如，与等待过程变量受混合的影响而发生变化的速度相比，代表 “ 注入暖混合物的冷水 ” 的前馈值可能会更快地增大输出值。

不使用积分控制时，常常使用偏置值。在这种情况下，可以通过调整偏置值来将输出保持在所需范围内，从而保持 PV 接近设定值。

串级回路

PID 通过将主回路的百分比输出分配给从回路的设定值，来级联两个回路。从回路根据从回路的 .MAXS 和 .MINS 值，自动将主回路的输出值转换为从回路设定值使用的正确工程单位。

梯形图



结构化文本

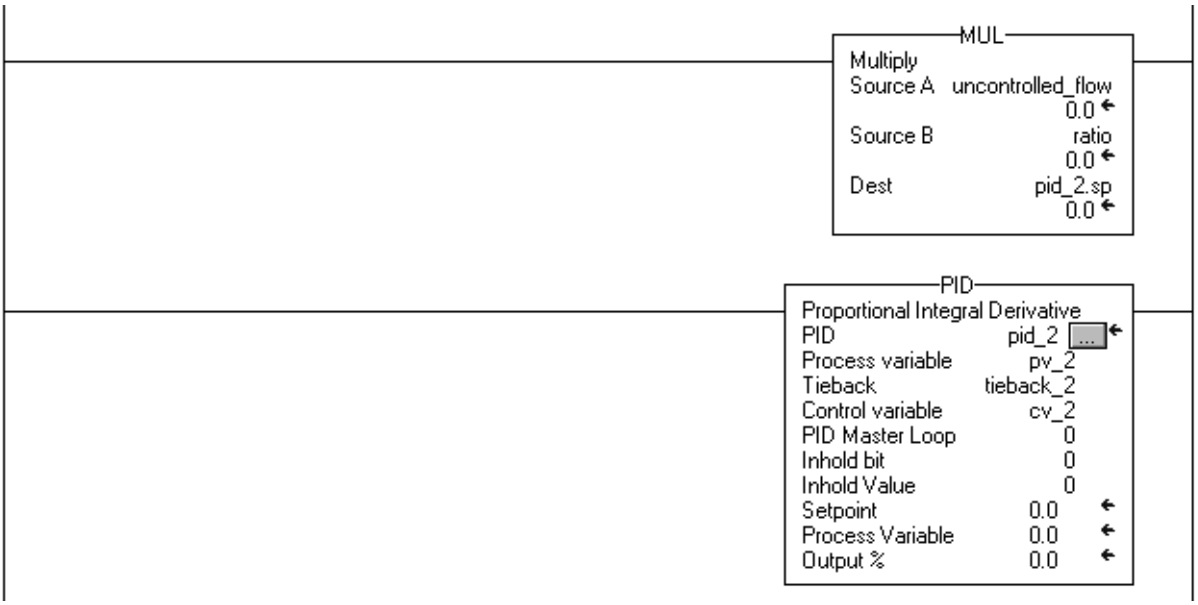
```
PID(master,pv_master,0,cv_master,0,0,0);

PID (slave,pv_slave,0,cv_slave,master,0,0);
```

控制比率

使用以下参数可以保持两个值具有一定的比率：

- 未控制值
- 控制值 (PID 指令使用的最终设定值)
- 这两个值间的比率

梯形图



结构化文本

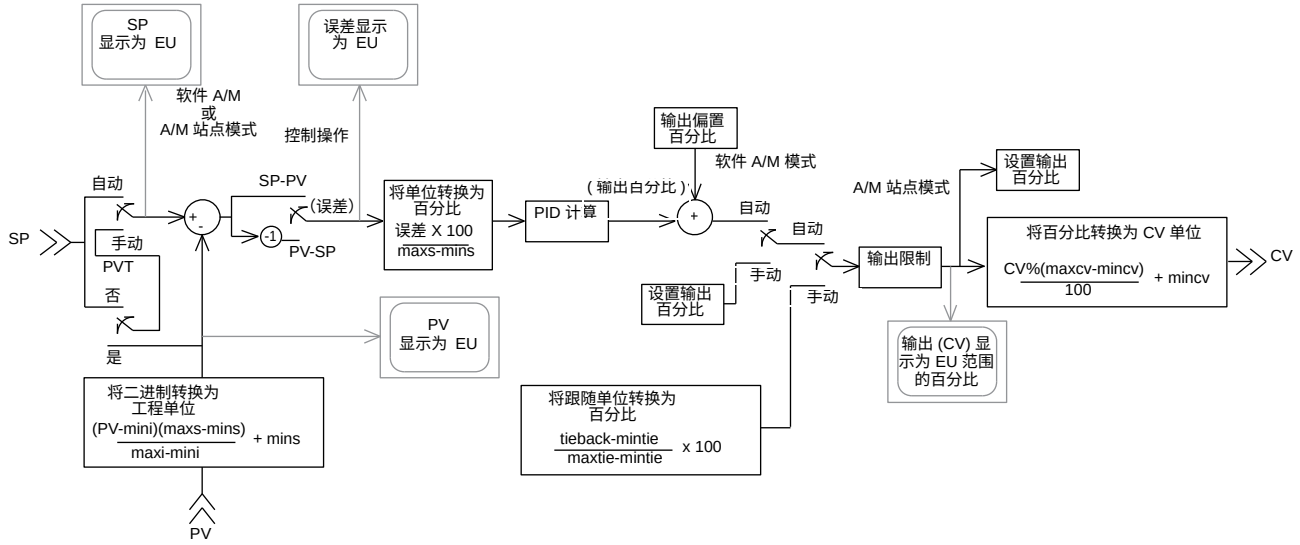
```
pid_2.sp := uncontrolled_flow * ratio

PID(pid_2,pv_2,tieback_2,cv_2,0,0,0);
```

对于以下乘法参数	输入以下值
目标 (Destination)	控制值
源 A (Source A)	未控制值
源 B (Source B)	比率 (Ratio)

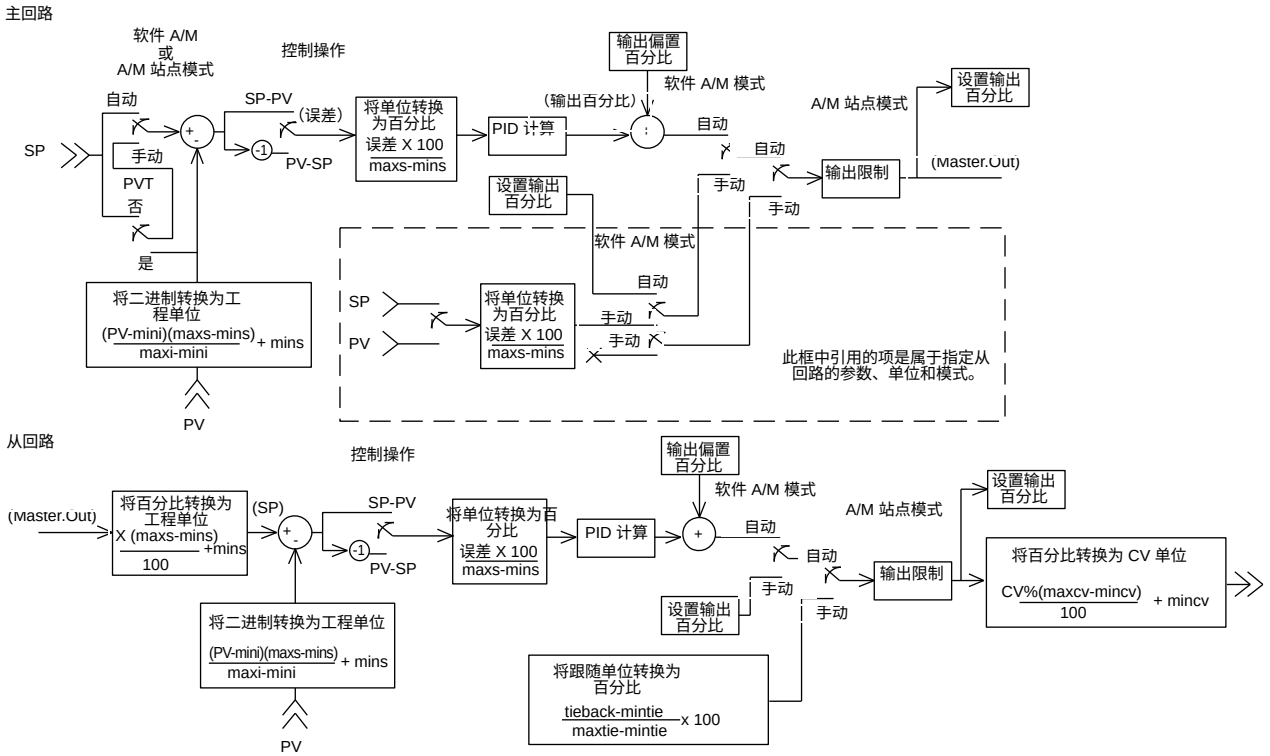
PID 原理

下图显示了 PID 指令的过程流。

PID 过程



具有主 / 从回路的 PID 过程



注：

三角函数指令
(SIN、COS、TAN、ASN、ASIN、ACS、ACOS、ATN、ATAN)

简介

三角函数指令使用三角函数进行算术运算。

如果要	使用以下指令	在以下语言中可用	页码
计算值的正弦值	SIN	梯形图 结构化文本 功能块	528
计算值的余弦值	COS	梯形图 结构化文本 功能块	531
计算值的正切值	TAN	梯形图 结构化文本 功能块	534
计算值的反正弦值	ASN ASIN(1)	梯形图 结构化文本 功能块	537
计算值的反余弦值	ACS ACOS(1)	梯形图 结构化文本 功能块	540
计算值的反正切值	ATN ATAN(1)	梯形图 结构化文本 功能块	543

(1) 仅限结构化文本。

可以混合使用不同数据类型，但可能会造成精度降低和舍入错误，而且指令的执行时间也会变长。请检查溢出状态位 (S:V)，查看结果是否被截断。

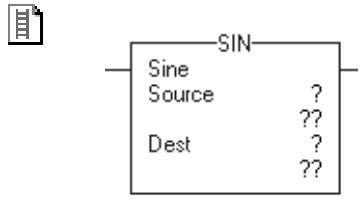
对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

正弦 (SIN)

SIN 指令计算源值 (以弧度为单位) 的正弦值，并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT REAL	立即数 标签	求该值的正弦值
目标 (Destination)	SINT INT DINT REAL	标签	要存储结果的标签

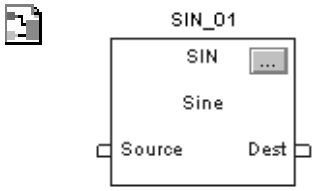
结构化文本

```
dest := SIN(source);
```

使用 SIN 作为函数。该函数计算 *source* 的正弦值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
SIN 标签	FBD_MATH_ADVANCED	结构	SIN 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 源值必须大于等于 -205887.4 ($-2\pi x2^{15}$) 且小于等于 205887.4 ($2\pi x2^{15}$)。目标标签中的结果值始终大于等于 -1 且小于等于 1。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

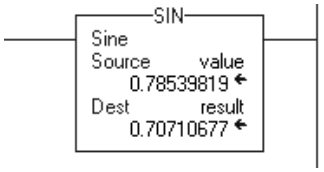
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的正弦值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的正弦值并将结果存储在 *result* 中。

梯形图



结构化文本

```
result := SIN(value);
```

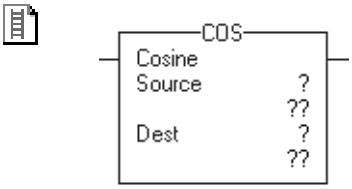
功能块

余弦 (COS)

COS 指令计算源值 (以弧度为单位) 的余弦值，并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT **REAL**	立即数 标签	求该值的余弦值
目标 (Destination)	SINT INT DINT **REAL**	标签	要存储结果的标签

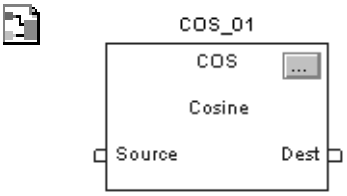
结构化文本

```
dest := COS(source);
```

使用 COS 作为函数。该函数计算 *source* 的余弦值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
COS 标签	FBD_MATH_ADVANCED	结构	COS 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 源值必须大于等于 -205887.4 ($-2\pi x2^{15}$) 且小于等于 205887.4 ($2\pi x2^{15}$)。目标标签中的结果值始终大于等于 -1 且小于等于 1。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

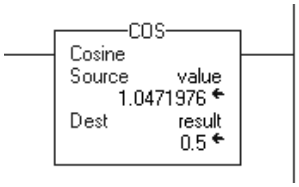
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的余弦值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的余弦值并将结果存储在 *result* 中。

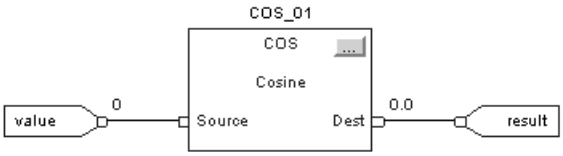
梯形图



结构化文本

```
result := COS(value);
```

功能块

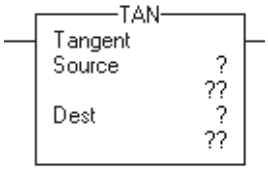


正切 (TAN)

TAN 指令计算源值 (以弧度为单位) 的正切值，并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT **REAL**	立即数 标签	求该值的正切值
目标 (Destination)	SINT INT DINT **REAL**	标签	要存储结果的标签

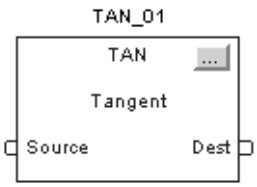
结构化文本

```
dest := TAN(source);
```

使用 TAN 作为函数。该函数计算 *source* 的正切值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
TAN 标签	FBD_MATH_ADVANCED	结构	TAN 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 源值必须大于等于 -102943.7 ($-2\pi x2^{14}$) 且小于等于 102943.7 ($2\pi x2^{14}$)。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

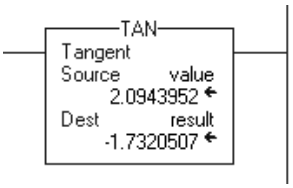
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的正切值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的正切值并将结果存储在 *result* 中。

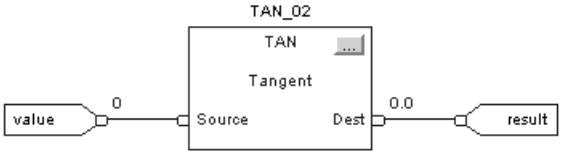
梯形图



结构化文本

```
result := TAN(value);
```

功能块

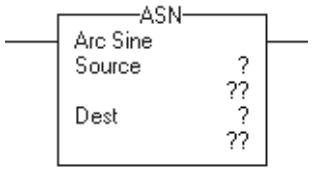


反正弦 (ASN)

ASN 指令计算源值的反正弦值并将结果存储在目标标签 (以弧度为单位) 中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT REAL	立即数 标签	求该值的反正弦值
目标 (Destination)	SINT INT DINT REAL	标签	要存储结果的标签

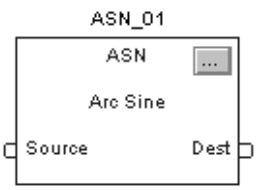
结构化文本

```
dest := ASIN(source);
```

使用 ASIN 作为函数。该函数计算 *source* 的反正弦值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
ASN 标签	FBD_MATH_ADVANCED	结构	ASN 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 源值必须大于等于 -1 且小于等于 1。目标标签中的结果值始终大于等于 -π/2 且小于等于 π/2(其中 π = 3.141593)。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

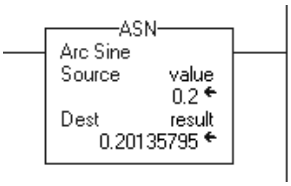
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的反正弦值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的反正弦值并将结果存储在 *result* 中。

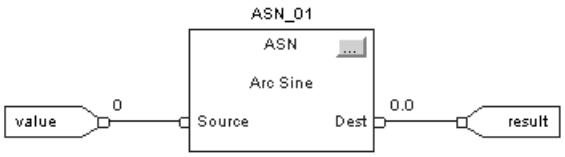
梯形图



结构化文本

```
result := ASIN(value);
```

功能块

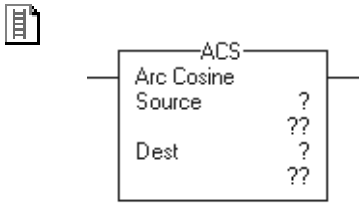


反余弦 (ACS)

ACS 指令计算源值的反余弦值并将结果存储在目标标签 (以弧度为单位) 中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT **REAL**	立即数 标签	求该值的反余弦值
目标 (Destination)	SINT INT DINT **REAL**	标签	要存储结果的标签

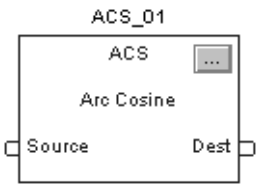
结构化文本

```
dest := ACOS(source);
```

使用 ACOS 作为函数。该函数计算 *source* 的反余弦值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
ACS 标签	FBD_MATH_ADVANCED	结构	ACS 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明：
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 源值必须大于等于 -1 且小于等于 1。目标标签中的结果值始终大于等于 0 或小于等于 π(其中 π = 3.141593)。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

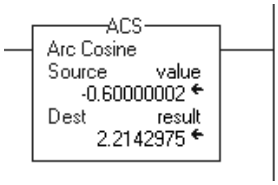
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的反余弦值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件：	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的反余弦值并将结果存储在 *result* 中。

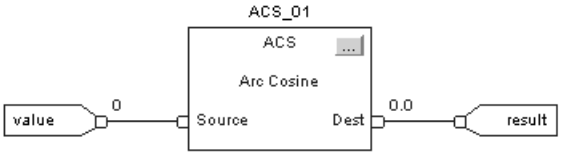
梯形图



结构化文本

```
result := ACOS(value);
```

功能块

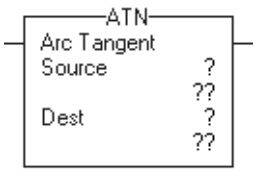


反正切 (ATN)

ATN 指令计算源值的反正切值并将结果存储在目标标签 (以弧度为单位) 中。

操作数：

梯形图



操作数：	类型	格式	说明
源 (Source)	SINT INT DINT **REAL**	立即数 标签	求该值的反正切值
目标 (Destination)	SINT INT DINT **REAL**	标签	要存储结果的标签

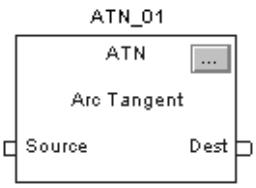
结构化文本

```
dest := ATAN(source);
```

使用 ATAN 作为函数。该函数计算 *source* 的反正切值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
ATN 标签	FBD_MATH_ADVANCED	结构	ATN 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	**数据类型**	**说明**
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 目标标签中的结果值始终大于等于 -π/2 且小于等于 π/2(其中 π = 3.141593)。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

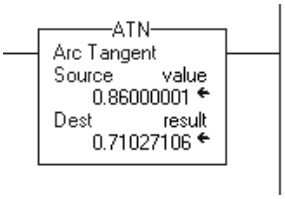
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的反正切值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的反正切值并将结果存储在 *result* 中。

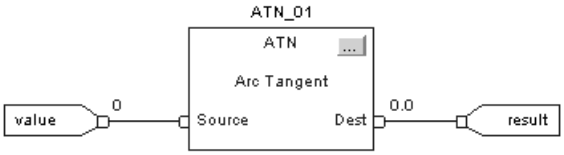
梯形图



结构化文本

```
result := ATAN(value);
```

功能块



注：

高级数学指令
（LN、 LOG、 XPY）

简介

高级数学指令包括下列指令。

如果要	使用以下指令	在以下语言中可用	页码
计算值的自然对数	LN	梯形图 结构化文本 功能块	548
计算值的以 10 为底的对数	LOG	梯形图 结构化文本 功能块	551
计算一个值的另一个值次幂	XPY	梯形图 结构化文本(1) 功能块	554

(1) 没有对等的结构化文本指令。在表达式中使用操作符。

可以混合使用不同数据类型，但可能会造成精度降低和舍入错误，而且指令的执行时间也会变长。请检查 S:V 位，查看结果是否被截断。

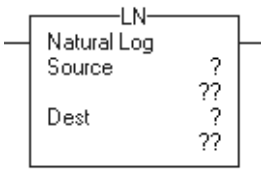
对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

自然对数 (LN)

LN 指令计算源值的自然对数并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT REAL	立即数 标签	求该值的自然对数
目标 (Destination)	SINT INT DINT REAL	标签	要存储结果的标签

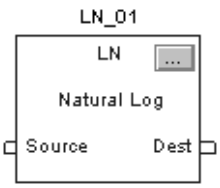
结构化文本

```
dest := LN(source);
```

使用 LN 作为函数。该函数计算 *source* 的自然对数，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
LN 标签	FBD_MATH_ADVANCED	结构	LN 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。

说明： 源值必须大于零，否则溢出状态位 (S:V) 置位。目标标签中的结果值大于等于 -87.33655 且小于等于 88.72284。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

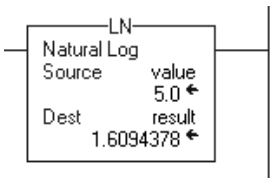
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的自然对数并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的自然对数并将结果存储在 *result* 中。

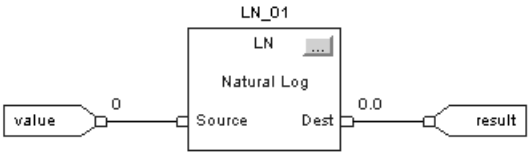
梯形图示例



结构化文本

```
result := LN(value);
```

功能块

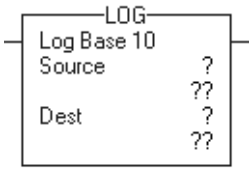


以 10 为底的对数 (LOG)

LOG 指令计算源值的以 10 为底的对数，并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT **REAL**	立即数 标签	计算该值的对数值
目标 (Destination)	SINT INT DINT **REAL**	标签	要存储结果的标签

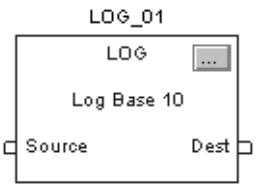
结构化文本

```
dest := LOG(source);
```

使用 LOG 作为函数。该函数计算 *source* 的对数值，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
LOG 标签	FBD_MATH_ADVANCED	结构	LOG 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	数学指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 源值必须大于零，否则溢出状态位 (S:V) 置位。目标标签中的结果值大于等于 -37.92978 且小于等于 38.53184。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

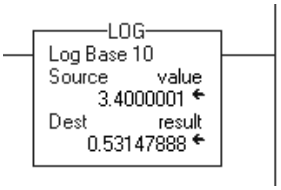
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值的对数值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 计算 *value* 的对数值并将结果存储在 *result* 中。

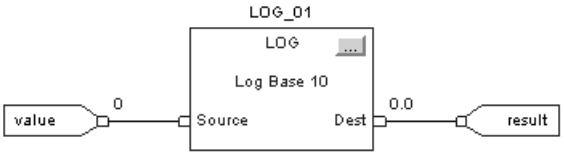
梯形图



结构化文本

```
result := LOG(value);
```

功能块

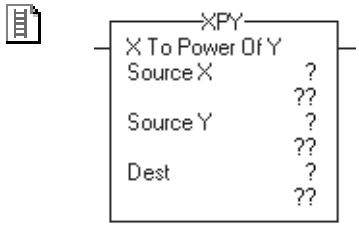


X 的 Y 次幂 (XPY)

XPY 指令计算源 A (X) 的源 B (Y) 次幂，将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 X (Source X)	SINT INT DINT **REAL**	立即数 标签	底数
源 Y (Source Y)	SINT INT DINT **REAL**	立即数 标签	指数
目标 (Destination)	SINT INT DINT **REAL**	标签	要存储结果的标签

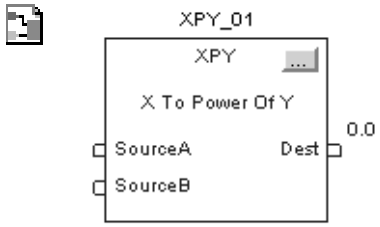
结构化文本

```
dest := sourceX ** sourceY;
```

在表达式中使用两个相连的乘号 “ ** ” 作为运算符。该表达式计算 *sourceX* 的 *sourceY* 次幂，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
XPY 标签	FBD_MATH	结构	XPY 结构

FBD_MATH 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 X (Source X)	REAL	底数。 任何浮点数都有效
源 Y (Source Y)	REAL	指数。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	数学指令的结果。将为该输出设置算术状态标志。

说明： 如果源值 X 为负，源值 Y 必须是整数值，否则将产生次要故障。

XPY 指令使用以下算法：*Destination* = X**Y

控制器评估 x^0=1 和 0^x=0。

算术状态标志： 算术状态标志将受到影响。

故障条件：

出现次要故障的条件	故障类型	故障代码
Source X 为负且 Source Y 不是整数	4	4

执行：

梯形图

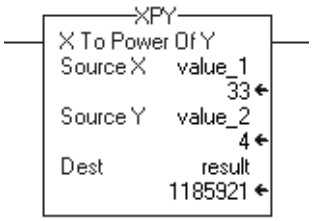
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器计算源值 X 的源值 Y 次幂并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： XPY 指令计算 *value_1* 的 *value_2* 次幂，并将结果存储在 *result* 中。

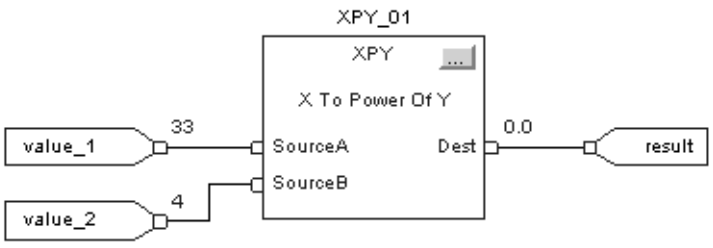
梯形图



结构化文本

```
result := (value_1 ** value_2);
```

功能块



数学转换指令
(DEG、 RAD、 TOD、 FRD、 TRN、 TRUNC)

简介

数学转换指令用于转换值。

如果要	使用以下指令	在以下语言中可用	页码
将弧度转换为角度	DEG	梯形图 结构化文本 功能块	558
将角度转换为弧度	RAD	梯形图 结构化文本 功能块	561
将整数值转换为 BCD 码	TOD	梯形图 功能块	564
将 BCD 码转换为整数值	FRD	梯形图 功能块	567
删除值的小数部分	TRN TRUNC(1)	梯形图 结构化文本 功能块	569

(1) 仅限结构化文本。

可以混合使用不同数据类型，但可能会造成精度降低和舍入错误，而且指令的执行时间也会变长。请检查 S:V 位，查看结果是否被截断。

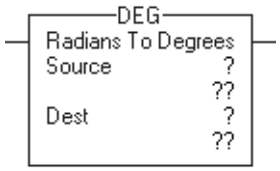
对于梯形图指令，如果数据类型加粗，则说明是最佳的数据类型。如果指令的所有操作数都使用同一种最佳数据类型 (通常是 DINT 或 REAL)，则指令执行起来会更快，所需的内存也会更少。

角度 (DEG)

DEG 指令将源 (以弧度为单位) 转换为角度，并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT REAL	立即数 标签	要转换为角度的值
目标 (Destination)	SINT INT DINT REAL	标签	要存储结果的标签

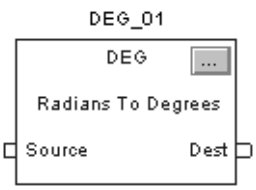
结构化文本

```
dest := DEG(source);
```

使用 DEG 作为函数。该函数将 *source* 转换为角度，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
DEG 标签	FBD_MATH_ADVANCED	结构	DEG 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	转换指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	转换指令的结果。将为该输出设置算术状态标志。

说明： DEG 指令使用以下算法：
源值 e*180/π(其中 π = 3.141593)

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

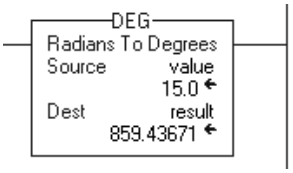
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器将源值转换为角度并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 将 *value* 转换为角度并将结果存储在 *result* 中。

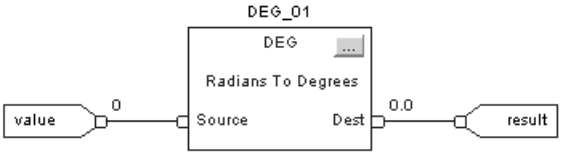
梯形图



结构化文本

```
result := DEG(value);
```

功能块

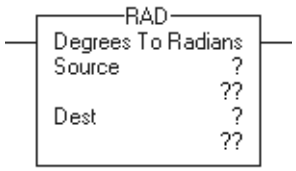


弧度 (RAD)

RAD 指令将源 (以角度为单位) 转换为弧度，并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT REAL	立即数 标签	要转换为弧度的值
目标 (Destination)	SINT INT DINT REAL	标签	要存储结果的标签

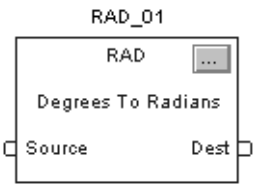
结构化文本

```
dest := RAD(source);
```

使用 RAD 作为函数。该函数将 *source* 转换为弧度，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
RAD 标签	FBD_MATH_ADVANCED	结构	RAD 结构

FBD_MATH_ADVANCED 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	转换指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	REAL	转换指令的结果。将为该输出设置算术状态标志。

说明： RAD 指令使用以下算法：
源值 *π/180(其中 π = 3.141593)

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

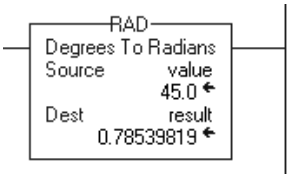
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器将源值转换为弧度并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例　将 *value* 转换为弧度并将结果存储在 *result* 中。

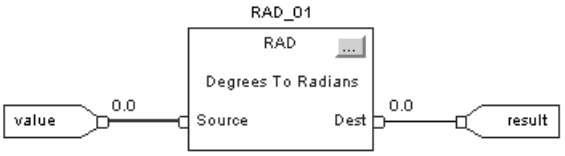
梯形图



结构化文本

```
result := RAD(value);
```

功能块

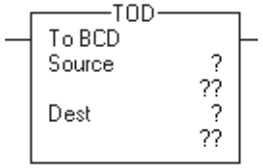


转换为 BCD (TOD)

TOD 指令将十进制值 (0 Source 99,999,999) 转换为 BCD 值，并将结果存储在 Destination 中。

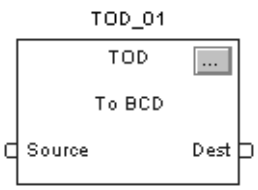
操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT	立即数 标签	要转换为十进制值的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
目标 (Destination)	SINT INT DINT	标签	存储结果

功能块



操作数	类型	格式	说明
TOD 标签	FBD_CONVERT	结构	TOD 结构

FBD_CONVERT 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	DINT	转换指令的输入。 任何整数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	DINT	转换指令的结果。将为该输出设置算术状态标志。

说明： BCD 是二 - 十进制代码数字系统，其使用 4 位二进制计数表示一个十进制数 (0…9)。

如果输入负的源值，指令将出现次要故障并清除目标标签。

算术状态标志： 算术状态标志将受到影响。

故障条件：

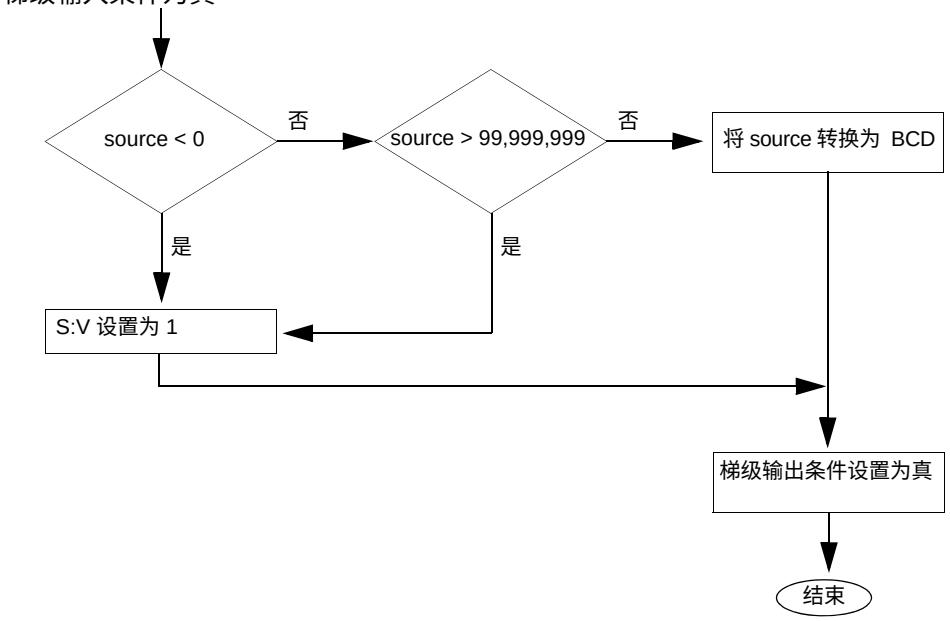
出现次要故障的条件	故障类型	故障代码
Source < 0	4	4

执行：

梯形图

条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。

梯级输入条件为真



梯级输入条件为真	控制器将源值转换为 BCD 并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

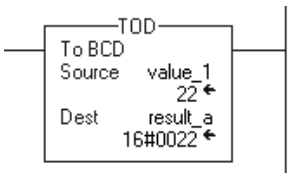
功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。

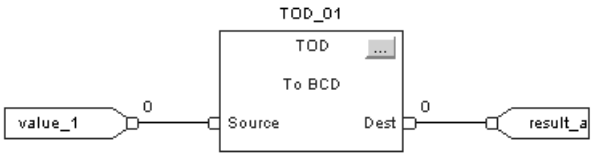
条件	操作
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： TOD 指令将 *value_1* 转换为 BCD 码，并将结果存储在 *result_a* 中。

梯形图



功能块

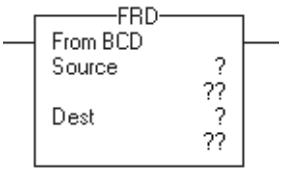


转换为整数 (FRD)

FRD 指令将 BCD 码 (源) 转换为十进制值，并将结果存储在目标标签中。

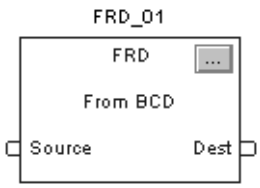
操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	SINT INT DINT	立即数 标签	要转换为十进制值的值
	SINT 或 INT 标签将通过填零转换为 DINT 值。		
目标 (Destination)	SINT INT DINT	标签	存储结果

功能块



操作数	类型	格式：	说明
FRD 标签	FBD_CONVERT	结构	FRD 结构

FBD_CONVERT 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	DINT	转换指令的输入。 任何整数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	DINT	转换指令的结果。将为该输出设置算术状态标志。

说明： FRD 指令将 BCD 码 (源) 转换为十进制值，并将结果存储在目标标签中。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

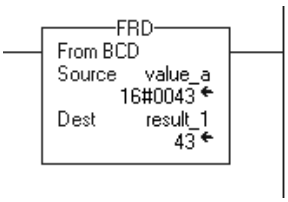
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器将源值转换为十进制值并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

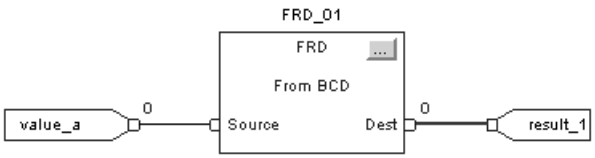
条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： FRD 指令将 *value_a* 转换为十进制值，并将结果存储在 *result_1* 中。

梯形图



功能块

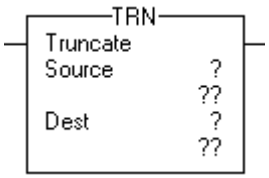


截断 (TRN)

TRN 指令删除 (截断) 源值的小数部分，并将结果存储在目标标签中。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	REAL	立即数 标签	要截断的值
目标 (Destination)	SINT INT DINT REAL	标签	要存储结果的标签

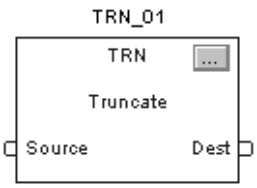
结构化文本

```
dest := TRUNC(source);
```

使用 TRUNC 作为函数。该函数将 *source* 截断，并将结果存储在 *dest* 中。

有关结构化文本中表达式的语法的信息，请参见结构化文本编程。

功能块



操作数	类型	格式	说明
TRN 标签	FBD_TRUNCATE	结构	TRN 结构

FBD_TRUNCATE 结构

输入参数	数据类型	说明
EnableIn	BOOL	使能输入。如果为清零状态，则不执行该指令，并且不更新输出。 默认为置位状态。
源 (Source)	REAL	转换指令的输入。 任何浮点数都有效
输出参数	数据类型	说明
EnableOut	BOOL	该指令生成有效结果。
目标 (Dest)	DINT	转换指令的结果。将为该输出设置算术状态标志。

说明： 截断不是对值四舍五入，而是非小数部分保持不变，无论小数部分的值是多少。

算术状态标志： 算术状态标志将受到影响。

故障条件： 无

执行：

梯形图

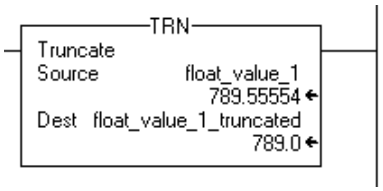
条件	操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	控制器删除源值的小数部分并将结果存储在目标标签中。 梯级输出条件设置为真。
后扫描	梯级输出条件设置为假。

功能块

条件	操作
预扫描	不执行任何操作。
指令第一次扫描	不执行任何操作。
指令第一次运行	不执行任何操作。
EnableIn 处于清零状态	清零 EnableOut。
EnableIn 处于置位状态	执行该指令。 置位 EnableOut。
后扫描	不执行任何操作。

示例： 删除 *float_value_1* 的小数部分，保持非小数部分不变，并将结果存储在 *float_value_1_truncated* 中。

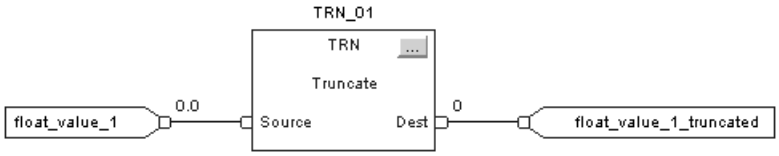
梯形图



结构化文本

```
float_value_1_truncated := TRUNC(float_value_1);
```

功能块



注：

ASCII 串口指令
(ABL、 ACB、 ACL、 AHL、 ARD、 ARL、 AWA、 AWT)

简介

使用 ASCII 串口指令可读写 ASCII 字符。 .

重要事项

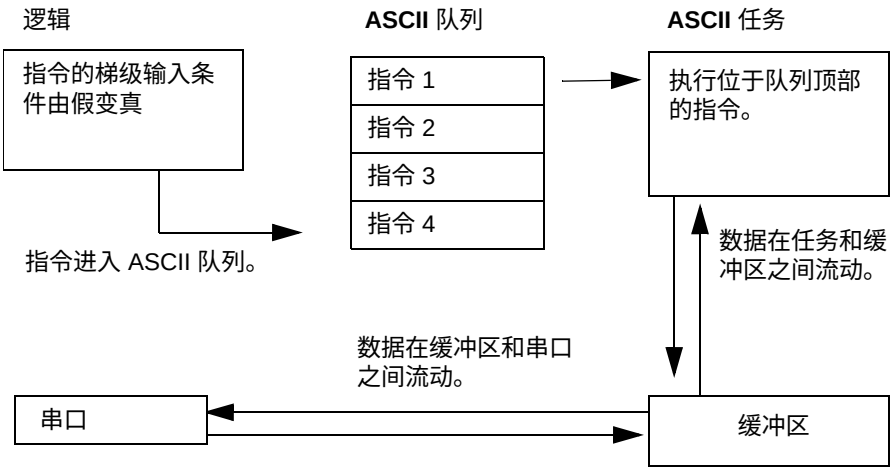
要使用 ASCII 串口指令，必须组态控制器的串口。有关步骤，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。

1756-L7*x* 控制器没有串口，无法使用 ASCII 读 / 写指令。此外，也不能将 ASCII 读 / 写指令重新定向到 USB 端口。

如果要	例如	使用以下指令	在以下语言中可用	页码
缓冲区包含终止符时测定	检查包含终止符的数据	ABL	梯形图 结构化文本	578
对缓冲区中的字符进行计数	在读取缓冲区之前检查所需的字符数	ACB	梯形图 结构化文本	581
清空缓冲区 清除当前正在执行或位于队列中的 ASCII 串口指令	• 在启动时删除缓冲区中的旧数据。 • 将缓冲区与设备同步。	ACL	梯形图 结构化文本	583
获取串口控制线的状态 打开或关闭 DTR 信号 打开或关闭 RTS 信号	导致调制解调器挂起	AHL	梯形图 结构化文本	585
读取固定数目的字符	从每次传输时发送相同数目字符的设备读取数据	ARD	梯形图 结构化文本	589
读取不定数目的字符，直到读取到第一组终止符 (包括在计数范围内)	从每次传输时发送不同数目字符的设备读取数据	ARL	梯形图 结构化文本	593
发送字符并自动附加一个或两个字符以标记数据的结束	发送始终使用相同终止符的信息	AWA	梯形图 结构化文本	597
发送字符	发送使用各种终止符的信息	AWT	梯形图 结构化文本	602

指令执行

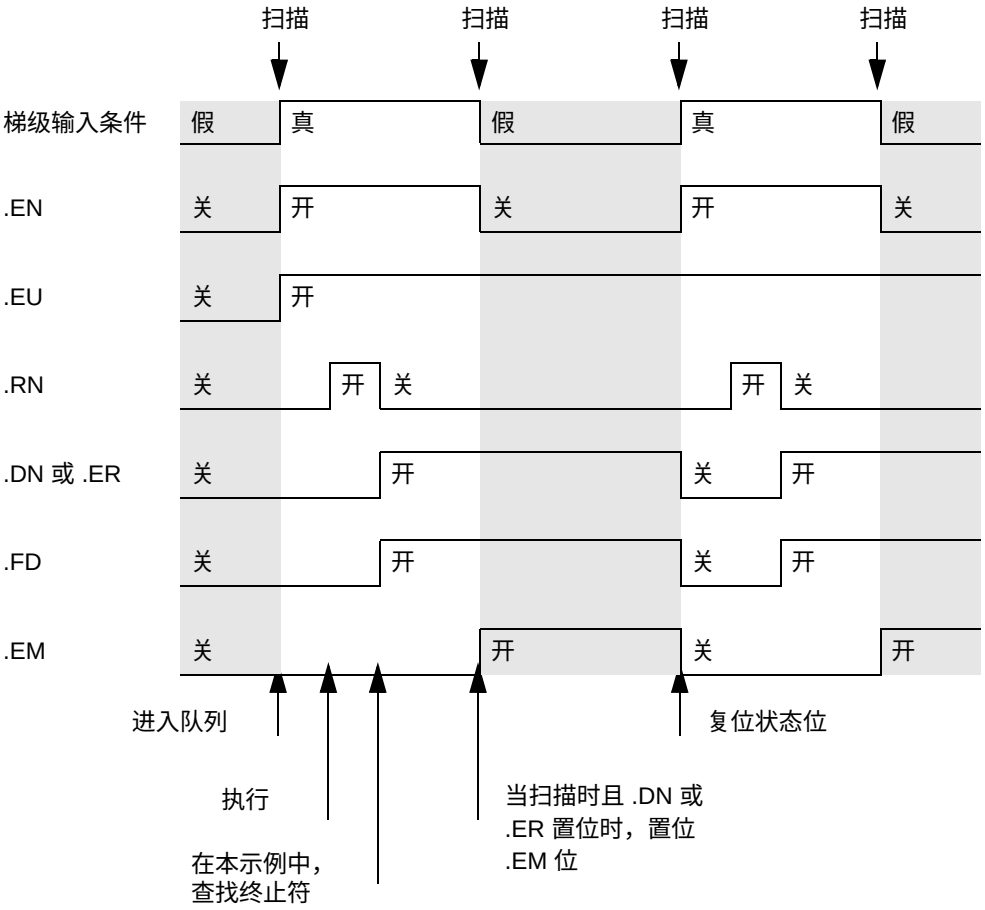
ASCII 串口指令的执行与逻辑扫描异步。



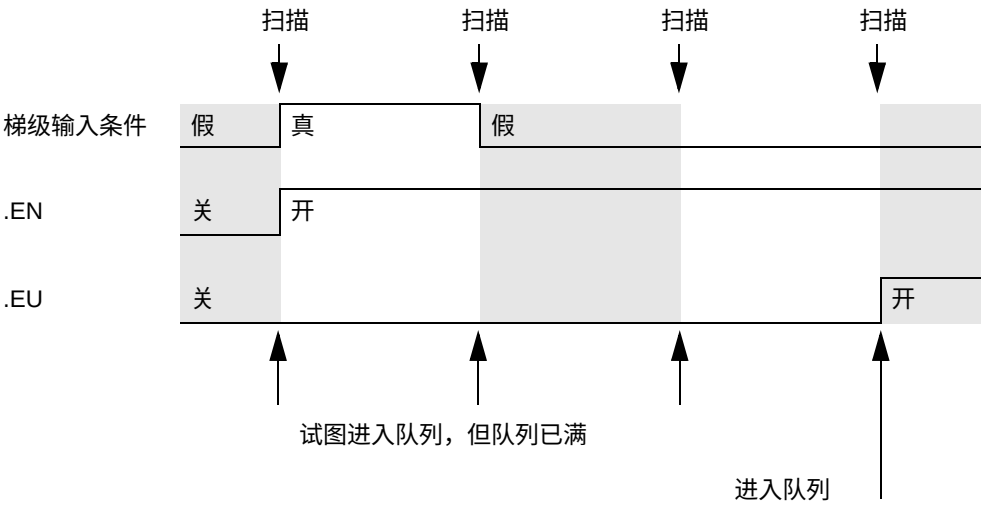
各 ASCII 串口指令 (ACL 除外) 均使用 SERIAL_PORT_CONTROL 结构来执行以下功能：

- 控制指令的执行
- 提供指令的状态信息

下面的时序图描述了当 ABL 指令测试到缓冲区含有终止符时状态位的变化。



ASCII 队列最多可容纳 16 条指令。队列已满后，每次后续扫描指令时其它指令试图进入队列，如下所示。

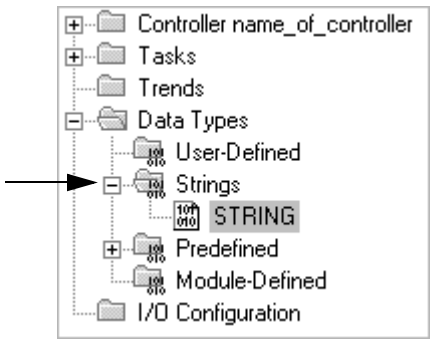


ASCII 错误代码

如果 ASCII 串口指令执行失败，其 SERIAL_PORT_CONTROL 结构的 ERROR 子元素将包含下列十六进制错误代码之一。

十六进制代码	含义
16#2	调制解调器转为离线。
16#3	CTS 信号在通信期间丢失。
16#4	串口处于系统模式。
16#A	在执行指令前， .UL 位置位。这将阻止指令的执行。
16#C	控制器从运行模式切换到程序模式。这将停止 ASCII 串口指令的执行并清空队列。
16#D	在“ 控制器属性 ”(Controller Properties) 对话框的“ 用户协议 ”(User Protocol) 页面中，更改并应用了缓冲区尺寸或响应模式参数。这将停止 ASCII 串口指令的执行并清空队列。
16#E	ACL 指令已执行。
16#F	串口组态从用户模式切换到系统模式。这将停止 ASCII 串口指令的执行并清空 ASCII 串口指令队列。
16#51	字符串标签的 LEN 值为负或大于字符串标签的 DATA 尺寸。
16#54	串口控制长度大于缓冲区尺寸。
16#55	串口控制长度为负或大于源或目标的大小。

字符串数据类型



可将 ASCII 字符存储于使用字符串数据类型的标签中。

- 可以使用默认的 STRING 数据类型。它最多可存储 82 个字符。
- 可以创建新的字符串数据类型来存储更少或更多的字符。

要创建新的字符串数据类型，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。

每个字符串数据类型包含以下几项。

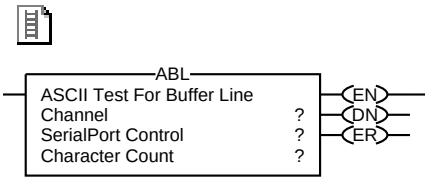
名称	数据类型	说明	备注
LEN	DINT	字符串中的字符数	以下情况下， LEN 会自动更新到新的字符数： • 使用 “ 字符串浏览器 ” (String Browser) 对话框键入字符。 • 使用读取、转换或处理字符串的指令。 LEN 显示当前字符串的长度。 DATA 子元素可能包含附加的旧字符，这些不在计数之列。
DATA	SINT 数组	字符串中的 ASCII 字符	• 要访问字符串中的字符，请键入标签名称。 例如，要访问 *string_1* 标签的字符，请键入 *string_1*。 • DATA 数组中的每个元素包含一个字符。 • 您可以创建新的字符串数据类型来存储更少或更多的字符。

缓冲区行的 ASCII 测试 (ABL)

ABL 指令对缓冲区中的字符进行计数，直到第一个终止符 (包括在计数范围内)。

操作数：

梯形图



操作数	类型	格式	说明
通道 (Channel)	DINT	立即数 标签	0
串口控制 (Serial Port Control)	SERIAL_PORT_ CONTROL	标签	控制操作的标签
字符计数 (Character Count)	DINT	立即数	0 在执行过程中，显示缓冲区中的字符数，包括第一组终止符。

结构化文本

```
ABL(Channel
    SerialPortControl);
```

操作数与梯形图 ABL 指令的操作数相同。可通过 SERIAL_PORT_CONTROL 结构的 .POS 子元素访问 Character Count 值。

SERIAL_PORT_CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示指令是否使能。
.EU	BOOL	队列位指示指令是否已进入 ASCII 队列。
.DN	BOOL	完成位指示指令完成的时间，但它与逻辑扫描异步。
.RN	BOOL	运行位指示指令是否正在执行。
.EM	BOOL	空位指示指令是否完成，但它与逻辑扫描同步。
.ER	BOOL	错误位指示指令失败 (出现错误) 的时间。
.FD	BOOL	发现位指示指令是否发现终止符。
.POS	DINT	位置测定缓冲区中的字符数，直到第一组终止符 (包括在计数范围内)。仅当指令发现终止符后，才会返回此数目。
.ERROR	DINT	错误包含一个指明错误原因的十六进制值。

说明　ABL 指令在缓冲区中搜索第一组终止符。如果指令发现终止符，其将执行以下操作：

- 置位 .FD 位。
- 对缓冲区中的字符进行计数，直到第一组终止符 (包括在计数范围内)。

在“ 控制器属性 ”(Controller Properties) 对话框的“ 用户协议 ”(User Protocol) 页面中，定义被指令视为终止符的 ASCII 字符。

请按以下指导原则对 ABL 指令进行编程。

1. 将控制器的串口组态为用户模式，并定义用作终止符的字符。
2. 这是一条触发执行指令。
   - 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
   - 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。

算术状态标志：　不受影响

故障条件：　无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	该指令在梯级输入条件从清零状态跳变到置位状态的时候执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令对缓冲区中的字符进行计数。 .EN 位置位。 清零 .UL 以外的其余状态位。 指令尝试进入 ASCII 队列。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 连续测试缓冲区搜索终止符。

梯形图

结构化文本

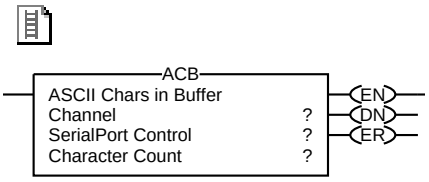
```
ABL(0,MV_line);
```

缓冲区中的 ASCII 字符 (ACB)

ACB 指令对缓冲区中的字符进行计数。

操作数：

梯形图



操作数	类型	格式	键入
通道 (Channel)	DINT	立即数 标签	0
串口控制 (Serial Port Control)	SERIAL_PORT_ CONTROL	标签	控制操作的标签
字符计数 (Character Count)	DINT	立即数	0 在执行过程中，显示缓冲区中的字符数。

结构化文本

```
ACB(Channel
    SerialPortControl);
```

操作数与梯形图 ACB 指令的操作数相同。但是，指定 Character Count 值的方式是访问 SERIAL_PORT_CONTROL 结构的 .POS 子元素，而不是将值包括在操作数列表中。

SERIAL_PORT_CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示指令是否使能。
.EU	BOOL	队列位指示指令是否已进入 ASCII 队列。
.DN	BOOL	完成位指示指令完成的时间，但它与逻辑扫描异步。
.RN	BOOL	运行位指示指令是否正在执行。
.EM	BOOL	空位指示指令是否完成，但它与逻辑扫描同步。
.ER	BOOL	错误位指示指令失败 (出现错误) 的时间。
.FD	BOOL	发现位指示指令是否找到字符。
.POS	DINT	位置测定缓冲区中的字符数，直到第一组终止符 (包括在计数范围内)。
.ERROR	DINT	错误包含一个指明错误原因的十六进制值。

说明： ACB 指令对缓冲区中的字符计数。

请按以下指导原则对 ACB 指令进行编程。

1. 将控制器的串口组态为用户模式。
2. 这是一条触发执行指令。
   - 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
   - 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。

算术状态标志： 不受影响

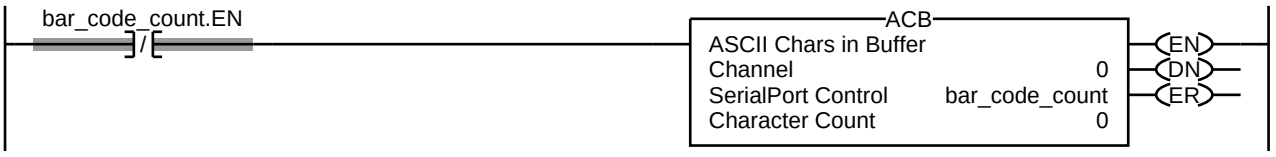
故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	该指令在梯级输入条件从清零状态跳变到置位状态的时候执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令对缓冲区中的字符进行计数。 .EN 位置位。 清零 .UL 以外的其余状态位。 指令尝试进入 ASCII 队列。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 对缓冲区中的字符进行连续计数。

梯形图



结构化文本

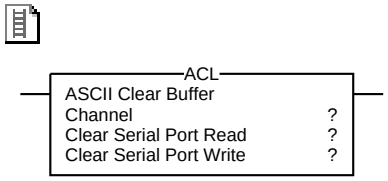
```
ACB(0,bar_code_count);
```

ASCII 清空缓冲区 (ACL)

ACL 指令立即清空缓冲区和 ASCII 队列。

操作数：

梯形图



操作数	类型	格式	键入
通道 (Channel)	DINT	立即数 标签	0
清除串口读 (Clear Serial Port Read)	BOOL	立即数 标签	要清空缓冲区并从队列中删除 ARD 和 ARL 指令，请键入 Yes。
清除串口写 (Clear Serial Port Write)	BOOL	立即数 标签	要从队列中删除 AWA 和 AWT 指令，请键入 Yes。

结构化文本

```
ACL(Channel,
    ClearSerialPortRead,
    ClearSerialPortWrite);
```

操作数与梯形图 ACL 指令的操作数相同。

说明： ACL 指令立即执行下列操作中的一个或两个：

- 清空字符缓冲区并清空读取指令的 ASCII 队列
- 清空写入指令的 ASCII 队列

请按以下指导原则对 ACL 指令进行编程。

1. 组态控制器的串口：

如果应用项目	则
使用 ARD 或 ARL 指令	选择用户模式
没有使用 ARD 或 ARL 指令	选择系统模式或用户模式

2. 要确定指令是否已从队列中删除或中止，请检查相应指令是否出现以下情况。
   - .ER 位置位。
   - .ERROR 子元素为 16#E。

算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	指令将执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令清除指定的指令并清空缓冲区。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 控制器进入运行模式时，清空缓冲区和 ASCII 码队列。

梯形图

结构化文本

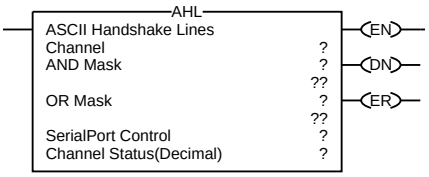
```
osri_1.InputBit := S:FS;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     ACL(0,0,1);
END_IF;
```

ASCII 握手线 (AHL)

AHL 指令获取控制线的状态，并打开或关闭 DTR 和 RTS 信号。

操作数：

梯形图



操作数	类型	格式	键入
通道 (Channel)	DINT	立即数 标签	0
ANDMask	DINT	立即数 标签	请参见说明。
ORMask	DINT	立即数 标签	
串口控制 (Serial Port Control)	SERIAL_PORT_CONTROL	标签	控制操作的标签
通道状态 (十进制)(Channel Status (Decimal))	DINT	立即数	0 在执行过程中，显示控制线的状态。

如要了解以下控制线的状态	检查以下位：
CTS	0
RTS	1
DSR	2
DCD	3
DTR	4
接收到 XOFF 字符	5

结构化文本

```
AHL(Channel,ANDMask,ORMask,
    SerialPortControl);
```

操作数与梯形图 AHL 指令的操作数相同。但是，指定 Character Status 值的方式是访问 SERIAL_PORT_CONTROL 结构的 .POS 子元素，而不是将值包括在操作数列表中。

SERIAL_PORT_CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示指令是否使能。
.EU	BOOL	队列位指示指令是否已进入 ASCII 队列。
.DN	BOOL	完成位指示指令完成的时间，但它与逻辑扫描异步。
.RN	BOOL	运行位指示指令是否正在执行。
.EM	BOOL	空位指示指令是否完成，但它与逻辑扫描同步。
.ER	BOOL	错误位指示指令失败 (出现错误) 的时间。
.FD	BOOL	发现位不适用于此指令。
.POS	DINT	位置存储控制行的状态。
.ERROR	DINT	错误包含一个指明错误原因的十六进制值。

说明： AHL 指令可以：

- 获取串口控制行的状态。
- 打开或关闭数据终端就绪 (DTR) 信号。
- 打开或关闭请求发送 (RTS) 信号。

请按以下指导原则对 AHL 指令进行编程。

1. 组态控制器的串口。

如果应用项目	则
使用 ARD 或 ARL 指令	选择用户模式
没有使用 ARD 或 ARL 指令	选择系统模式或用户模式

2. 通过下表可为 ANDMask 和 ORMask 操作数选择正确的值。

要将 DTR	并将 RTS	输入以下 ANDMask 值	并输入以下 ORMask 值
关闭	关闭	3	0
	打开	1	2
	保持不变	1	0
打开	关闭	2	1
	打开	0	3
	保持不变	0	1
保持不变	关闭	2	0
	打开	0	2
	保持不变	0	0

3. 这是一条触发执行指令。
- 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
- 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。

算术状态标志： 不受影响

故障条件：

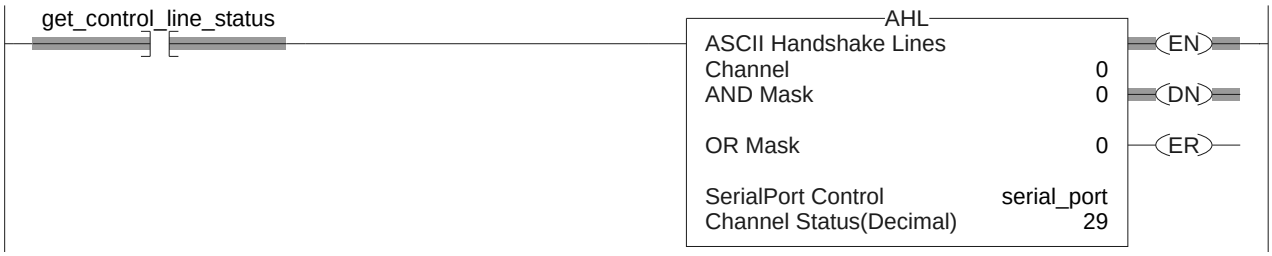
类型	代码	原因	恢复方法
4	57	AHL 指令执行失败，因为串口设置为不握手。	以下两法任选其一： • 更改串口的控制线设置。 • 删除 AHL 指令。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	该指令在梯级输入条件从清零状态跳变到置位状态的时候执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令获取控制线的状态，并打开或关闭 DTR 和 RTS 信号。 .EN 位置位。 清零 .UL 以外的其余状态位。 指令尝试进入 ASCII 队列。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 当 *get_control_line_status* 变成置位状态时，获取串口控制线的状态，并将该状态存储到 Channel Status 操作数中。要查看特定控制线的状态，请监视 SerialPortControl 标签并展开 POS 子元素。

梯形图



结构化文本

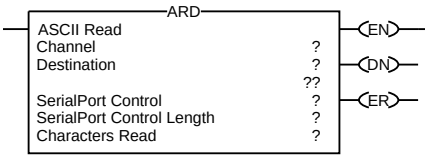
```
osri_1.InputBit := get_control_line_status;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     AHL(0,0,0,serial_port);
END_IF;
```

ASCII 读取 (ARD)

ARD 指令从缓冲区中移除字符，并将它们存储到目标标签中。

操作数：

梯形图



操作数	类型	格式	键入	备注
通道 (Channel)	DINT	立即数 标签	0	
目标 (Destination)	字符串 SINT INT DINT	标签	字符要移动 (读取) 到的标签： • 对于字符串数据类型，键入标签名称。 • 对于 SINT、 INT 或 DINT 数组，键入数组的第一个元素。	• 如果要比较、转换或处理字符，请用字符串数据类型。 • 字符串数据类型包括： • 默认的 STRING 数据类型 • 所创建的任何新的字符串数据类型
串口控制 (Serial Port Control)	SERIAL_PORT_ CONTROL	标签	控制操作的标签	
串口控制长度 (Serial Port Control Length)	DINT	立即数	要移动 (读取) 到目标标签的字符数	• Serial Port Control Length 必须小于或等于目标标签的尺寸。 • 如果要将 Serial Port Control Length 设置为与目标标签的尺寸相等，则键入 0。
读取的字符 (Characters Read)	DINT	立即数	0	在执行过程中，显示已读取的字符数。

结构化文本

```
ARD(Channel,Destination,
    SerialPortControl);
```

操作数与梯形图 ARD 指令的操作数相同。但是，指定 Serial Port Control Length 和 Characters Read 值的方式是访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 子元素，而不是将值包括在操作数列表中。

SERIAL_PORT_CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示指令是否使能。
.EU	BOOL	队列位指示指令是否已进入 ASCII 队列。
.DN	BOOL	完成位指示指令完成的时间，但它与逻辑扫描异步。
.RN	BOOL	运行位指示指令是否正在执行。
.EM	BOOL	空位指示指令是否完成，但它与逻辑扫描同步。
.ER	BOOL	错误位指示指令失败 (出现错误) 的时间。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示要移动 (读取) 到目标标签的字符数。
.POS	DINT	位置显示已读取的字符数。
.ERROR	DINT	错误包含一个指明错误原因的十六进制值。

说明： ARD 指令从缓冲区中移除指定数目的字符，并将它们存储到目标标签中。

- ARD 指令连续执行，直到移除指定数目的字符 (Serial Port Control Length)。
- 执行 ARD 指令期间，不执行其它 ASCII 串口指令。

请按以下指导原则对 ARD 指令进行编程。

1. 将控制器的串口组态为用户模式。
2. 使用 ACB 指令的结果来触发 ARD 指令。这样可防止 ARD 指令在其等待所需数目的字符时阻塞 ASCII 队列。
3. 这是一条触发执行指令。
   - 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
   - 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。
4. 要在指令完成时触发后续操作，请检查 EM 位。

算术状态标志： 不受影响

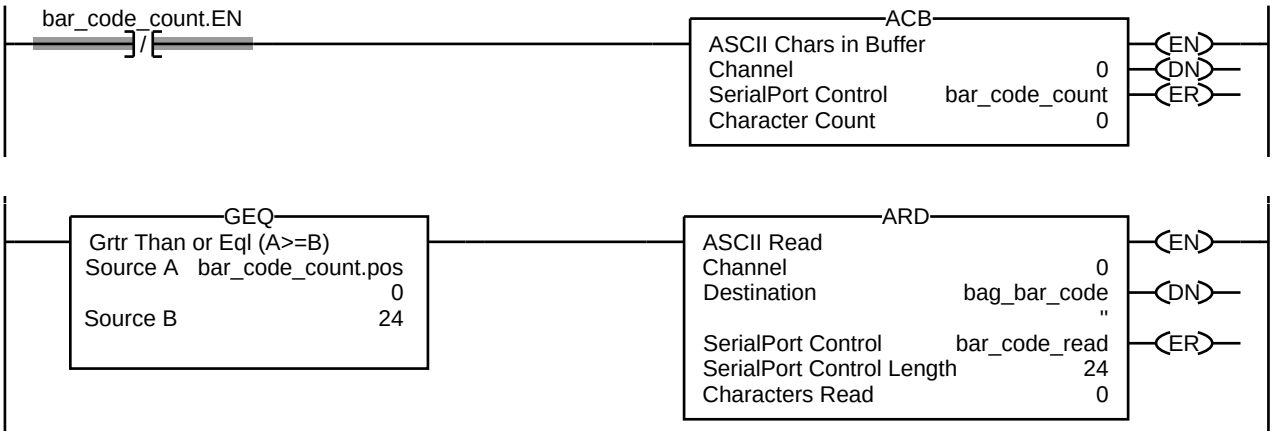
故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	该指令在梯级输入条件从清零状态跳变到置位状态的时候执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令从缓冲区中移除字符，并将它们存储到目标标签中。 .EN 位置位。 清零 .UL 以外的其余状态位。 指令尝试进入 ASCII 队列。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 条形码阅读器将条形码发送到控制器的串口 (通道 0)。每个条形码包含 24 个字符。为了确定控制器接收条形码的时间， ACB 指令对缓冲区中的字符进行连续计数。当缓冲区至少包含 24 个字符时，说明控制器已接收条形码。 ARD 指令将条形码移动到 *bag_bar_code* 标签 (一个字符串) 的 DATA 子元素中。

梯形图



结构化文本

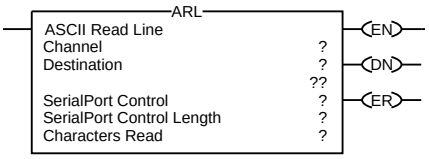
```
ACB(0,bar_code_count);
IF bar_code_count.POS >= 24 THEN
    bar_code_read.LEN := 24;
    ARD(0,bag_bar_code,bar_code_read);
END_IF;
```

ASCII 读取行 (ARL)

ARL 指令从缓冲区中移除指定字符，并将它们存储到目标标签中。

操作数：

梯形图



操作数	类型	格式	键入	备注
通道 (Channel)	DINT	立即数 标签	0	
目标 (Destination)	字符串 SINT INT DINT	标签	字符要移动 (读取) 到的标签： • 对于字符串数据类型，键入标签名称。 • 对于 SINT、 INT 或 DINT 数组，键入数组的第一个元素。	• 如果要比较、转换或处理字符，请使用字符串数据类型。 • 字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
串口控制 (Serial Port Control)	SERIAL_PORT_ CONTROL	标签	控制操作的标签	
串口控制长度 (Serial Port Control Length)	DINT	立即数	在发现终止符前可读取的最大字符数	• 键入任何信息可包含的最大字符数 (即，在发现终止符前何时停止读取)。 例如，如果信息的长度范围是 3 到 6 个字符，则键入 6。 • Serial Port Control Length 必须小于或等于目标标签的尺寸。 • 如果要将 Serial Port Control Length 设置为与目标标签的尺寸相等，则键入 0。
读取的字符 (Characters Read)	DINT	立即	0	在执行过程中，显示已读取的字符数。

```
ARL(Channel,Destination,
    SerialPortControl);
```

结构化文本

操作数与梯形图 ARL 指令的操作数相同。但是，指定 Serial Port Control Length 和 Characters Read 值的方式是访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 子元素，而不是将值包括在操作数列表中。

SERIAL_PORT_CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示指令是否使能。
.EU	BOOL	队列位指示指令是否已进入 ASCII 队列。
.DN	BOOL	完成位指示指令完成的时间，但它与逻辑扫描异步。
.RN	BOOL	运行位指示指令是否正在执行。
.EM	BOOL	空位指示指令是否完成，但它与逻辑扫描同步。
.ER	BOOL	错误位指示指令失败 (出现错误) 的时间。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示可移动到目标标签的最大字符数 (即，在发现终止符前何时停止读取)。
.POS	DINT	位置显示已读取的字符数。
.ERROR	DINT	错误包含一个指明错误原因的十六进制值。

说明： ARL 指令从缓冲区中移除字符，并将它们存储到目标标签中。

- ARL 指令连续执行，直到移除以下字符数：
  - 第一组终止符
  - 指定的字符数 (Serial Port Control Length)
- 执行 ARL 指令期间，不执行其它 ASCII 串口指令。

请按以下指导原则对 ARL 指令进行编程。

1. 组态控制器的串口。

   a. 选择用户模式。
   b. 定义用作终止符的字符。

2. 使用 ABL 指令的结果来触发 ARL 指令。这样可防止 ARL 指令在其等待终止符时阻塞 ASCII 队列。

3. 这是一条触发执行指令。
   - 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
   - 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。

4. 要在指令完成时触发后续操作，请检查 EM 位。

算术状态标志： 不受影响

故障条件： 无

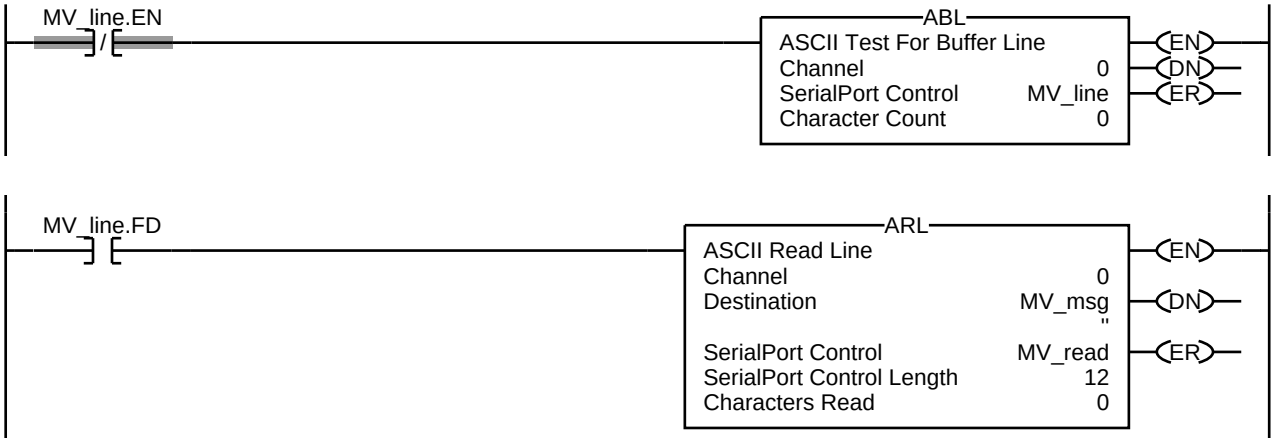
执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	该指令在梯级输入条件从清零状态跳变到置位状态的时候执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令从缓冲区中移除指定字符，并将它们存储到 Destination 中。 .EN 位置位。 清零 .UL 以外的其余状态位。 指令尝试进入 ASCII 队列。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 继续测试缓冲区搜索来自 MessageView 终端的信息。由于各信息均以回车 ($r) 结束，所以在 “ 控制器属性 ” (Controller Properties) 对话框的 “ 用户协议 ” (User Protocol) 页面中将回车组态为终止符。ABL 找到回车时，置位 FD 位。

ABL 指令找到回车 (*MV_line.FD* 置位) 时，说明控制器已接收一条完整的信息。 ARL 指令从缓冲区中移除字符，直到回车符 (包括在内)，并将它们存储到 *MV_msg* 标签 (一个字符串) 的 DATA 子元素中。

梯形图



结构化文本

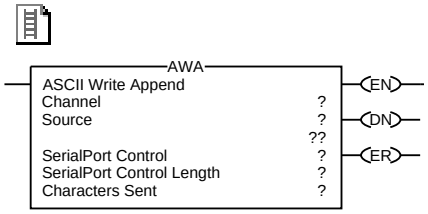
```
ABL(0,MV_line);


osri_1.InputBit := MVLine.FD;
OSRI(osri_1);


IF (osri_1.OutputBit) THEN
     mv_read.LEN := 12;
     ARL(0,MV_msg,MV_read);
END_IF;
```

ASCII 写入附加 (AWA)

AWA 指令将源标签中指定数目的字符发送到串行设备，并附加一个或两个预定义字符。

操作数：

梯形图



操作数	类型	格式	键入	备注
通道 (Channel)	DINT	立即数 标签	0	
源 (Source)	字符串 SINT INT DINT	标签	包含待发送字符的标签： • 对于字符串数据类型，键入标签名称。 • 对于 SINT、 INT 或 DINT 数组，键入数组的第一个元素。	• 如果要比较、转换或处理字符，请使用字符串数据类型。 • 字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
串口控制 (Serial Port Control)	SERIAL_PORT_ CONTROL	标签	控制操作的标签	
串口控制长度 (Serial Port Control Length)	DINT	立即数	要发送的字符数	• Serial Port Control Length 必须小于或等于源标签的尺寸。 • 如果要将 Serial Port Control Length 设置为与源标签中的字符数相等，则键入 0。
发送的字符 (Characters Sent)	DINT	立即数	0	在执行过程中，显示已发送的字符数。

结构化文本

```
AWA(Channel,Source,
    SerialPortControl);
```

操作数与梯形图 AWA 指令的操作数相同。但是，指定 Serial Port Control Length 和 Characters Sent 值的方式是访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 子元素，而不是将值包括在操作数列表中。

SERIAL_PORT_CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示指令是否使能。
.EU	BOOL	队列位指示指令是否已进入 ASCII 队列。
.DN	BOOL	完成位指示指令完成的时间，但它与逻辑扫描异步。
.RN	BOOL	运行位指示指令是否正在执行。
.EM	BOOL	空位指示指令是否完成，但它与逻辑扫描同步。
.ER	BOOL	错误位指示指令失败 (出现错误) 的时间。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示要发送的字符数。
.POS	DINT	位置显示已发送的字符数。
.ERROR	DINT	错误包含一个指明错误原因的十六进制值。

说明： AWA 指令：

- 将源标签中指定数目的字符 (Serial Port Control Length) 发送到与控制器串口相连的设备。
- 向字符末尾添加 (附加) 一个或两个在 “ 控制器属性 ” (Controller Properties) 对话框的 “ 用户协议 ” (User Protocol) 页面中定义的字符。

请按以下指导原则对 AWA 指令进行编程。

1. 组态控制器的串口。

   a. 应用程序是否还包含 ARD 或 ARL 指令？

如果	则
是	选择用户模式
无	选择系统模式或用户模式

   b. 定义要附加到数据的字符。

2. 这是一条触发执行指令。
   - 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
   - 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。

3. 每次指令执行时，是否始终发送相同数目的字符？

如果	则
是	在 Serial Port Control Length 中，键入要发送的字符数。
无	在执行指令前，将源标签的 LEN 子元素设置为 Serial Port Control 标签的 LEN 子元素。

算术状态标志： 不受影响

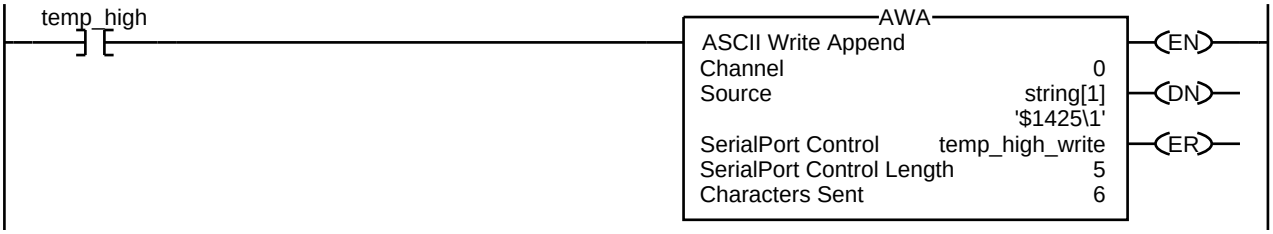
故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	该指令在梯级输入条件从清零状态跳变到置位状态的时候执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令发送指定数目的字符，并附加一个或两个预定义的字符。 .EN 位置位。 清零 .UL 以外的其余状态位。 指令尝试进入 ASCII 队列。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例 **1**： 当温度超出上限 (*temp_high* 置位) 时， AWA 指令向与控制器串口相连的 MessageView 终端发送一条信息。该信息包含 *string[1]* 标签 (一个字符串) 的 DATA 子元素的五个字符。(*$14* 计作一个字符。它是 Ctrl-T 字符的十六进制代码。) 该指令还发送 (附加) 控制器属性中定义的字符。在本示例中， AWA 指令发送标记信息结束的回车 ($0D)。

梯形图

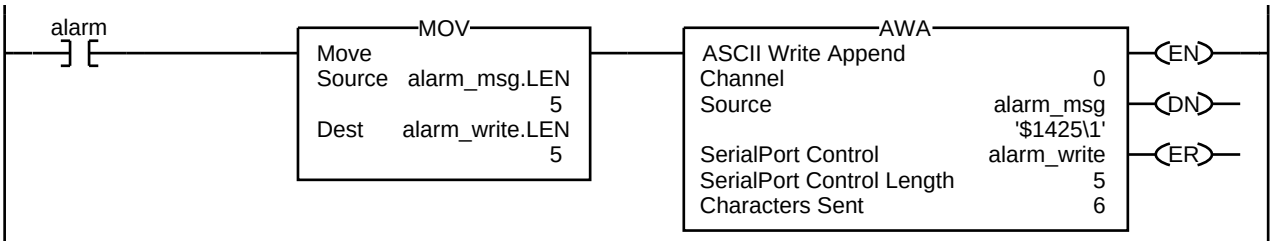


结构化文本

```
IF temp_high THEN

    temp_high_write.LEN := 5;

    AWA(0,string[1],temp_high_write);

    temp_high := 0;

END_IF;
```

示例 **2**： *alarm* 置位时， AWA 指令发送 *alarm_msg* 中的指定数目的字符，并附加终止符。由于 *alarm_msg* 中的字符数是变化的，所以梯级首先将字符串的长度 (*alarm_msg*.LEN) 移动到 AWA 指令的 Serial Port Control Length (*alarm_write*.LEN)。在 *alarm_msg* 中，*$14* 计作一个字符。它是 Ctrl-T 字符的十六进制代码。

梯形图



结构化文本

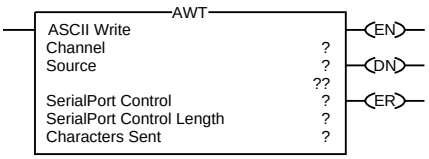
```
osri_1.InputBit := alarm;
OSRI(osri_1);


IF (osri_1.OutputBit) THEN
     alarm_write.LEN := alarm_msg.LEN;
     AWA(0,alarm_msg,alarm_write);
END_IF;
```

ASCII 写入 (AWT)

AWT 指令将源标签中指定数目的字符发送到串行设备。

操作数：

梯形图



操作数	类型	格式	键入	备注
通道 (Channel)	DINT	立即数 标签	0	
源 (Source)	字符串 SINT INT DINT	标签	包含待发送字符的标签： • 对于字符串数据类型，键入标签名称。 • 对于 SINT、 INT 或 DINT 数组，键入数组的第一个元素。	• 如果要比较、转换或处理字符，请使用字符串数据类型。 • 字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
串口控制 (Serial Port Control)	SERIAL_PORT_ CONTROL	标签	控制操作的标签	
串口控制长度 (Serial Port Control Length)	DINT	立即数	要发送的字符数	• Serial Port Control Length 必须小于或等于源标签的尺寸。 • 如果要将 Serial Port Control Length 设置为与源标签中的字符数相等，则键入 0。
发送的字符 (Characters Sent)	DINT	立即数	0	在执行过程中，显示已发送的字符数。

结构化文本

```
AWT(Channel,Source,
    SerialPortControl);
```

操作数与梯形图 AWT 指令的操作数相同。但是，指定 Serial Port Control Length 和 Characters Sent 值的方式是访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 子元素，而不是将值包括在操作数列表中。

SERIAL_PORT_CONTROL 结构

助记符	数据类型	说明
.EN	BOOL	使能位指示指令是否使能。
.EU	BOOL	队列位指示指令是否已进入 ASCII 队列。
.DN	BOOL	完成位指示指令完成的时间，但它与逻辑扫描异步。
.RN	BOOL	运行位指示指令是否正在执行。
.EM	BOOL	空位指示指令是否完成，但它与逻辑扫描同步。
.ER	BOOL	错误位指示指令失败 (出现错误) 的时间。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示要发送的字符数。
.POS	DINT	位置显示已发送的字符数。
.ERROR	DINT	错误包含一个指明错误原因的十六进制值。

说明： AWT 指令将源标签中指定数目的字符 (Serial Port Control Length) 发送到与控制器串口相连的设备。

请按以下指导原则对 AWT 指令进行编程。

1. 组态控制器的串口。

如果应用程序	则
使用 ARD 或 ARL 指令	选择用户模式
没有使用 ARD 或 ARL 指令	选择系统模式或用户模式

2. 这是一条触发执行指令。
   - 在梯形图中，每次将梯级输入条件从清零跳变为置位时，都应执行此指令。
   - 在结构化文本中，请为指令设置限定条件，以便仅在触发时才执行此指令。

3. 每次指令执行时，是否始终发送相同数目的字符？

如果	则
是	在 Serial Port Control Length 中，键入要发送的字符数。
无	在执行指令前，将源标签的 LEN 子元素移动到 Serial Port Control 标签的 LEN 子元素。

算术状态标志： 不受影响

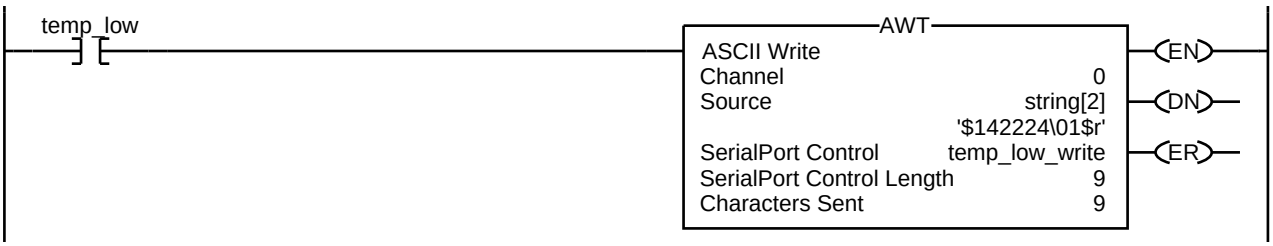
故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	该指令在梯级输入条件从清零状态跳变到置位状态的时候执行。 梯级输出条件设置为真。	N/A
EnableIn 处于置位状态	N/A	EnableIn 始终置位。 执行该指令。
指令执行	该指令发送指定数目的字符。 .EN 位置位。 清零 .UL 以外的其余状态位。 指令尝试进入 ASCII 队列。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例 **1**： 当温度达到下限 (*temp_low* 置位) 时，AWT 指令向与控制器串口相连的 MessageView 终端发送一条信息。该信息包含 *string[2]* 标签 (一个字符串) 的 DATA 子元素的九个字符。 (*$14* 计作一个字符。它是 Ctrl-T 字符的十六进制代码。) 最后一个字符是回车 ($r)，它标记信息的结束。

梯形图

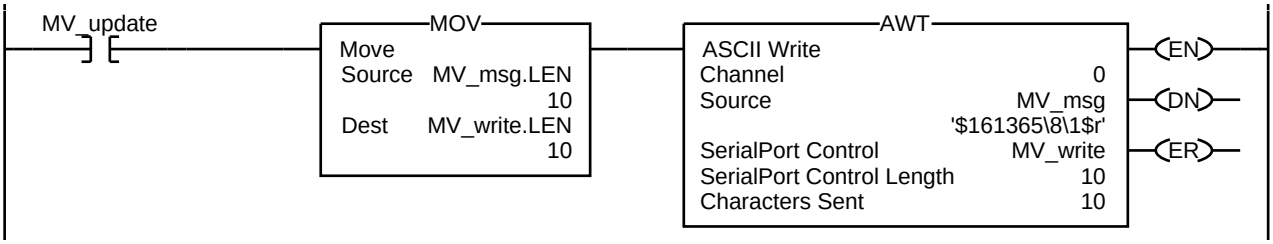


结构化文本

```
osri_1.InputBit := temp_low;
OSRI(osri_1);
IF (osri_1.OutputBit) THEN
     temp_low_write.LEN := 9;
     AWT(0,string[2],temp_low_write);
END_IF;
```

示例 **2**： 当 *MV_update* 置位时， AWT 指令发送 *MV_msg* 中的字符。由于 *MV_msg* 中的字符数是变化的，所以梯级首先将字符串的长度 (*MV_msg*.LEN) 移动到 AWT 指令的 Serial Port Control Length (*MV_write*.LEN)。在 *MV_msg* 中， *$16* 计作一个字符。它是 Ctrl-V 字符的十六进制代码。

梯形图



结构化文本

```
osri_1.InputBit := MV_update;
OSRI(osri_1);


IF (osri_1.OutputBit) THEN

     MV_write.LEN := Mv_msg.LEN;

     AWT(0,MV_msg,MV_write);

END_IF;
```

注：

ASCII 字符串指令
(CONCAT、 DELETE、 FIND、 INSERT、 MID)

简介

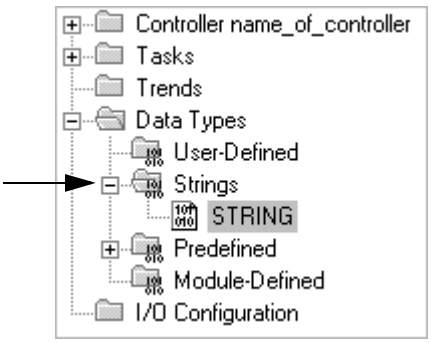
使用 ASCII 字符串指令可修改和创建由 ASCII 字符构成的字符串。

如果要	例如	使用以下指令	可用编程语言	页码
将字符添加到字符串末尾	将终止符或分隔符添加到字符串中	CONCAT	梯形图 结构化文本	609
删除字符串中的字符	移除字符串中的头字符或控制字符	DELETE	梯形图 结构化文本	611
确定子字符串的起始字符	找出字符串中的一组字符	FIND	梯形图 结构化文本	613
将字符插入字符串	创建使用变量的字符串	INSERT	梯形图 结构化文本	615
提取字符串中的字符	提取条形码中的信息	MID	梯形图 结构化文本	617

还可以使用以下指令比较或转换 ASCII 字符。

如果要	使用以下指令	页码
将字符串与另一字符串进行比较	CMP	214
查看字符是否等于特定字符	EQU	219
查看字符是否不等于特定字符	NEQ	250
查看字符是否大于等于特定字符	GEQ	219
查看字符是否大于特定字符	GRT	227
查看字符是否小于等于特定字符	LEQ	231
查看字符是否小于特定字符	LES	235
重新排列 INT、 DINT 或 REAL 标签的字节	SWPB	307
在字符串数组中查找字符串	FSC	354
将字符转换为 SINT、 INT、 DINT 或 REAL 值	STOD	621

如果要	使用以下指令	页码
将字符转换为 REAL 值	STOR	624
将 SINT、 INT、 DINT 或 REAL 值转换为由 ASCII 字符构成的字符串	DTOS	627
将 REAL 值转换为由 ASCII 字符构成的字符串	RTOS	629



字符串数据类型

可将 ASCII 字符存储于使用字符串数据类型的标签中。

- 可以使用默认的 STRING 数据类型。它最多可存储 82 个字符。
- 可以创建新的字符串数据类型来存储更少或更多的字符。

要创建新的字符串数据类型，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。

每个字符串数据类型包含以下几项。

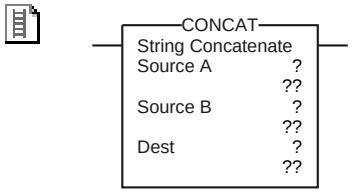
名称	数据类型	说明	备注
LEN	DINT	字符串中的字符数	以下情况下， LEN 会自动更新到新的字符数： • 使用 “ 字符串浏览器 ” (String Browser) 对话框键入字符。 • 使用读取、转换或处理字符串的指令。 LEN 显示当前字符串的长度。 DATA 子元素可能包含附加的旧字符，这些不在计数之列。
DATA	SINT 数组	字符串中的 ASCII 字符	• 要访问字符串中的字符，请键入标签名称。 例如，要访问 *string_1* 标签的字符，请键入 *string_1*。 • DATA 数组中的每个元素包含一个字符。 • 您可以创建新的字符串数据类型来存储更少或更多的字符。

字符串串连 (CONCAT)

CONCAT 指令可将 ASCII 字符添加到字符串末尾。

操作数：

梯形图



操作数	类型	格式	键入内容	备注
源 A (Source A)	字符串	标签	包含起始字符的标签	字符串数据类型包括：
源 B (Source B)	字符串	标签	包含结束字符的标签	• 默认的 STRING 数据类型。
目标 (Destination)	字符串	标签	要存储结果的标签	• 所创建的任何新的字符串数据类型。

结构化文本

```
CONCAT(SourceA,SourceB,
    Dest);
```

操作数与梯形图 CONCAT 指令的操作数相同。

说明： CONCAT 指令将 “ 源 A ” 中的字符与 “ 源 B ” 中的字符相组合，并将结果放在 “ 目标 ” 中。

- “ 源 A ” 中的字符在前，“ 源 B ” 中的字符在后。
- 除非 “ 源 A ” 和 “ 目标 ” 是同一标签，否则 “ 源 A ” 保持不变。

算术状态标志： 不受影响

故障条件：

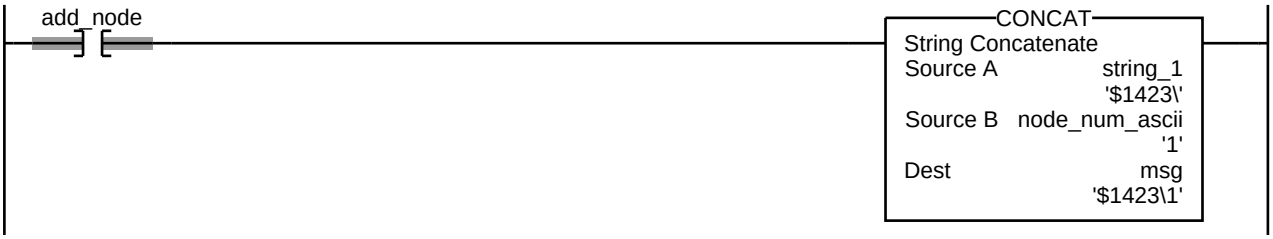
类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令将字符串串连在一起。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 为了在 MessageView 终端中触发信息，控制器必须发送包含信息号和节点号的 ASCII 字符串。 *String_1* 包含信息号。当 *add_node* 置位时， CONCAT 指令将 *node_num_ascii*(节点号) 中的字符添加到 *string_1* 中字符的结尾，然后将结果存储在 *msg* 中。

梯形图



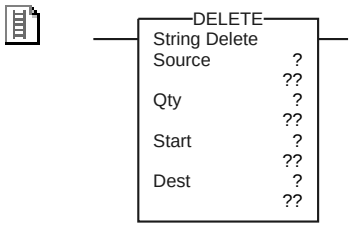
结构化文本

```
IF add_node THEN
     CONCAT(string_1,node_num_ascii,msg);
     add_node := 0;
END_IF;
```

字符串删除 (DELETE)

DELETE 指令不会将所有 ASCII 字符从字符串中自动删除。而是通过算法，根据 “ 源 ” 的起始位置、数量和尺寸来确定删除字符串中的哪些字符。

操作数：



梯形图

DELETE 指令：

- 将 “ 源 ” 中的字符串复制到 “ 目标 ”，同时忽略已删除的字符并用所复制的字符的数目更新 “ 目标 ” 字符串。
- 通过 “ 源 ” 字符串中字符的位置和要删除的字符的数目更新 “ 目标 ” 字符串的长度。
- 保持 “ 源 ” 不变，除非 “ 源 ” 和 “ 目标 ” 是同一标签。

操作数	类型	格式	键入内容	备注
源 (Source)	字符串	标签	包含要从中删除字符的字符串的标签	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
数量 (Quantity)	SINT INT **DINT**	立即数 标签	要删除的字符数	“ 起始 ” 值加上 “ 数量 ” 值必须小于等于 “ 源 ” 的 DATA 尺寸。
起始 (Start)	SINT INT **DINT**	立即数 标签	要删除的第一个字符的位置	键入介于 1 和 “ 源 ” 的 DATA 尺寸之间的数值。
目标 (Destination)	字符串	标签	要存储结果的标签	

结构化文本

```
DELETE(Source,Qty,Start,
    Dest);
```

操作数与梯形图 DELETE 指令的操作数相同。

算术状态标志： 不受影响

故障条件：

类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	56	“ 起始 ” 值或 “ 数量 ” 值无效。	1. 检查 “ 起始 ” 值是否介于 1 和 “ 源 ” 的 DATA 尺寸之间。 2. 检查 “ 起始 ” 值加上 “ 数量 ” 值是否小于等于 “ 源 ” 的 DATA 尺寸。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令删除指定字符。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 来自终端的 ASCII 信息包含头字符。控制器读取数据后 (*term_read.EM* 置位)， DELETE 指令将删除此头字符。

梯形图

结构化文本

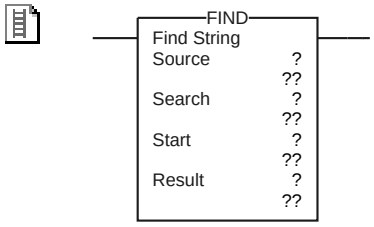
```
IF term_read.EM THEN
     DELETE(term_input,1,1,term_text);
     term_read.EM := 0;
END_IF;
```

查找字符串 (FIND)

FIND 指令用于在另一字符串内找出指定字符串的起始位置。

操作数：

梯形图



操作数	类型	格式	键入内容	备注
源 (Source)	字符串	标签	要在其中进行搜索的字符串	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
搜索 (Search)	字符串	标签	要查找的字符串	
起始 (Start)	SINT INT **DINT**	立即数 标签	“ 源 ” 中开始进行搜索的位置	键入介于 1 和 “ 源 ” 的 DATA 尺寸之间的数值。
结果 (Result)	SINT INT **DINT**	标签	存储要查找的字符串的起始位置的标签	

结构化文本

```
FIND(Source,Search,Start,
    Result);
```

操作数与上述梯形图 FIND 指令的操作数相同。

说明： FIND 指令在 “ 源 ” 字符串中搜索 “ 搜索 ” 字符串。如果指令查找 “ 搜索 ” 字符串，“ 结果 ” 将显示 “ 搜索 ” 字符串在 “ 源 ” 字符串中的起始位置。

算术状态标志： 不受影响

故障条件：

类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	56	“ 起始 ” 值无效。	检查 “ 起始 ” 值是否介于 1 和 “ 源 ” 的 DATA 尺寸之间。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令搜索指定字符。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 来自 MessageView 终端的信息包含几段信息。每段信息用反斜线字符 [\ ] 分隔开。为了找到一段信息， FIND 指令搜索反斜线字符并在 *find_pos* 中记录反斜线字符的位置。

梯形图

结构化文本

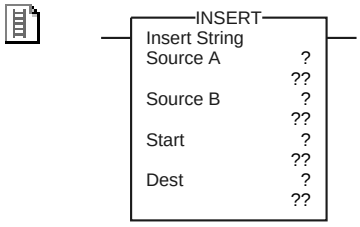
```
IF MV_read.EM THEN

     FIND(MV_msg,find,1,find_pos);

     MV_read.EM := 0;

END_IF;
```

插入字符串 (INSERT)

INSERT 指令用于将 ASCII 字符添加到字符串内的指定位置。

操作数：

梯形图



操作数	类型	格式	键入内容	备注
源 A (Source A)	字符串	标签	要添入字符的字符串	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
源 B (Source B)	字符串	标签	包含要添加的字符的字符串	
起始 (Start)	SINT INT **DINT**	立即数 标签	“ 源 A ” 中要添加字符的位置	键入介于 1 和 “ 源 ” 的 DATA 尺寸之间的数值。
结果 (Result)	字符串	标签	要存储结果的字符串	

结构化文本

```
INSERT(SourceA,SourceB,
    Start,Dest);
```

操作数与梯形图 INSERT 指令的操作数相同。

说明： INSERT 指令将 “ 源 B ” 中的字符添加到 “ 源 A ” 中的指定位置，并将结果放在 “ 目标 ” 中。

- “ 起始 ” 定义 “ 源 B ” 在 “ 源 A ” 中的添加位置。
- 除非 “ 源 A ” 和 “ 目标 ” 是同一标签，否则 “ 源 A ” 保持不变。

算术状态标志： 不受影响

故障条件：

类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	56	“ 起始 ” 值无效。	检查 “ 起始 ” 值是否介于 1 和 “ 源 ” 的 DATA 尺寸之间。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令插入指定字符。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 当 *temp_high* 置位时， INSERT 指令将 *string_2* 中的字符添加到 *string_1* 中的位置 2，并将结果放入 *string_3*：

梯形图

结构化文本

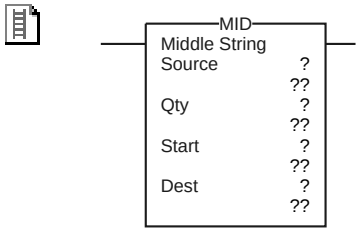
```
IF temp_high THEN
     INSERT(string_1,string_2,2,string_3);
     temp_high := 0;
END_IF;
```

中间字符串 (MID)

MID 指令用于从字符串中复制指定数目的 ASCII 字符，并将其存储在另一字符串中。

操作数：

梯形图



操作数	类型	格式	键入内容	备注
源 (Source)	字符串	标签	要从中复制字符的字符串	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
数量 (Quantity)	SINT INT **DINT**	立即数 标签	要复制的字符数	“ 起始 ” 值加上 “ 数量 ” 值必须小于等于 “ 源 ” 的 DATA 尺寸。
起始 (Start)	SINT INT **DINT**	立即数 标签	要复制的第一个字符的位置	键入介于 1 和 “ 源 ” 的 DATA 尺寸之间的数值。
目标 (Destination)	字符串	标签	要将字符复制到的字符串	

结构化文本

```
MID(Source,Qty,Start,
    Dest);
```

操作数与梯形图 MID 指令的操作数相同。

说明： MID 指令从 “ 源 ” 中复制一组字符，并将结果放在 “ 目标 ” 中。

- “ 起始 ” 位置和 “ 数量 ” 定义要复制的字符。
- 除非 “ 源 ” 和 “ 目标 ” 是同一标签，否则 “ 源 ” 保持不变。

算术状态标志： 不受影响

故障条件：

类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	56	“ 起始 ” 值或 “ 数量 ” 值无效。	1. 检查 “ 起始 ” 值是否介于 1 和 “ 源 ” 的 DATA 尺寸之间。 2. 检查 “ 起始 ” 值加上 “ 数量 ” 值是否小于等于 “ 源 ” 的 DATA 尺寸。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令从字符串中复制指定字符，并将其保存在另一字符串中。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 在机场的行李处理传送带中，每个包裹都得到一个条形码。条形码的字符 9…17 是航班号和包裹的到港机场。读取条形码后 (*bag_read.EM* 置位)， MID 指令将航班号和到港机场复制到 *bag_flt_and_dest* 字符串中。

梯形图

结构化文本

```
IF bag_read.EM THEN
     MID(bar_barcode,9,9,bag_flt_and_dest);
     bag_read.EM := 0;
END_IF;
```

ASCII 转换指令
(STOD、 STOR、 DTOS、 RTOS、 UPPER、 LOWER)

简介

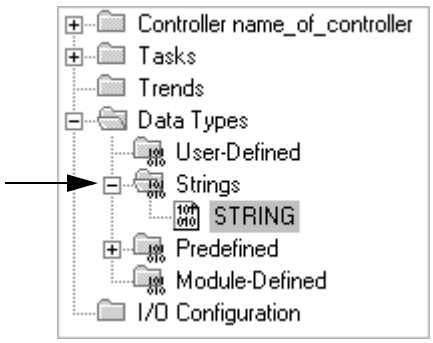
使用 ASCII 转换指令可更改数据的格式。

如果要	例如	使用以下指令	可用编程语言	页码
将整型值的 ASCII 表示转换为 SINT、INT、DINT 或 REAL 值	将台秤或其它 ASCII 设备的值转换为整数，以便在逻辑中使用	STOD	梯形图 结构化文本	621
将浮点值的 ASCII 表示转换为 REAL 值	将台秤或其它 ASCII 设备的值转换为 REAL 值，以便在逻辑中使用	STOR	梯形图 结构化文本	624
将 SINT、INT、DINT 或 REAL 值转换为由 ASCII 字符构成的字符串	将变量转换为 ASCII 字符串，以便将其发送到 MessageView 终端	DTOS	梯形图 结构化文本	627
将 REAL 值转换为由 ASCII 字符构成的字符串	将变量转换为 ASCII 字符串，以便将其发送到 MessageView 终端	RTOS	梯形图 结构化文本	629
将由 ASCII 字符构成的字符串中的字母转换为大写形式	将操作员输入的内容转换为全大写形式，以便可在数组中进行搜索	UPPER	梯形图 结构化文本	631
将由 ASCII 字符构成的字符串中的字母转换为小写形式	将操作员输入的内容转换为全小写形式，以便可在数组中进行搜索	LOWER	梯形图 结构化文本	633

还可以使用以下指令比较或处理 ASCII 字符。

如果要	使用以下指令	页码
将字符添加到字符串末尾	CONCAT	609
删除字符串中的字符	DELETE	611
确定子字符串的起始字符	FIND	613
将字符插入字符串	INSERT	615
提取字符串中的字符	MID	617
重新排列 INT、 DINT 或 REAL 标签的字节	SWPB	307
将字符串与另一字符串进行比较	CMP	214
查看字符是否等于特定字符	EQU	219
查看字符是否不等于特定字符	NEQ	250
查看字符是否大于等于特定字符	GEQ	223
查看字符是否大于特定字符	GRT	227
查看字符是否小于等于特定字符	LEQ	231
查看字符是否小于特定字符	LES	235
在字符串数组中查找字符串	FSC	354

字符串数据类型



可将 ASCII 字符存储于使用字符串数据类型的标签中。

- 可以使用默认的 STRING 数据类型。它最多可存储 82 个字符。
- 可以创建新的字符串数据类型来存储更少或更多的字符。

要创建新的字符串数据类型，请参见 《Logix5000 控制器通用步骤编程手册》 (出版号 1756-PM001)。

每个字符串数据类型包含以下几项。

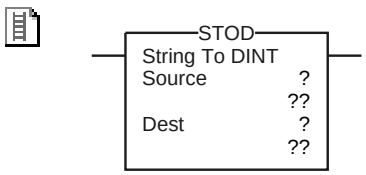
名称	数据类型	说明	备注
LEN	DINT	字符串中的字符数	以下情况下， LEN 会自动更新到新的字符数： • 使用 “ 字符串浏览器 ” (String Browser) 对话框键入字符。 • 使用读取、转换或处理字符串的指令。 LEN 显示当前字符串的长度。 DATA 子元素可能包含附加的旧字符，这些不在计数之列。
DATA	SINT 数组	字符串中的 ASCII 字符	• 要访问字符串中的字符，请键入标签名称。 例如，要访问 *string_1* 标签的字符，请键入 *string_1*。 • DATA 数组中的每个元素包含一个字符。 • 您可以创建新的字符串数据类型来存储更少或更多的字符。

字符串转换为 DINT (STOD)

STOD 指令用于将整数的 ASCII 表示转换为整型值或 REAL 值。

操作数：

梯形图



操作数	类型	格式	键入内容	备注
源 (Source)	字符串	标签	含有 ASCII 形式的值的标签	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
目标 (Destination)	SINT INT **DINT** REAL	标签	要存储整型值的标签	如果 “ 源 ” 值为浮点数，则指令只转换该数小数点前的部分 (无论 “ 目标 ” 采用何种数据类型)。

结构化文本

```
STOD(Source,Dest);
```

操作数与梯形图 STOD 指令的操作数相同。

说明： STOD 将 “ 源 ” 值转换为整数，并将结果放在 “ 目标 ” 中。

- 该指令可转换正数和负数。
- 如果 “ 源 ” 字符串含有非数字字符， STOD 将转换第一组连续数字：
  - 该指令将跳过任何起始控制字符或非数字字符 (数字前的负号除外)。
  - 如果字符串含有多组由分隔符 (如 /) 分隔的数字，指令将仅转换第一组数字。

算术状态标志： 算术状态标志将受到影响。

故障条件

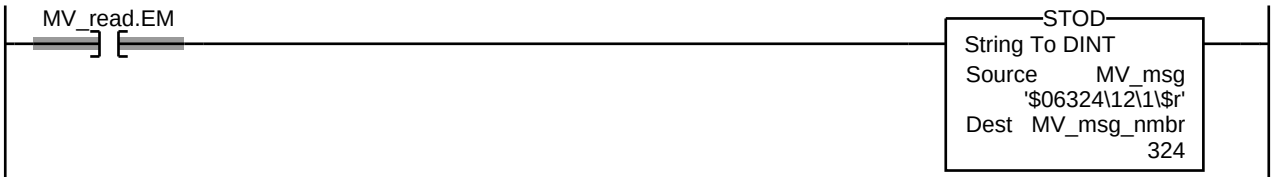
类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	53	输出数超出目标数据类型的限制。	以下两法任选其一： • 减小 ASCII 值的尺寸。 • 使用更大的目标数据类型。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	SC 置位。 “ 目标 ” 清零。 此指令对 “ 源 ” 进行转换。 如果结果为零，则 S:Z 置位	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 当 *MV_read.EM* 置位时，STOD 指令将 *MV_msg* 中的第一组数字字符转换为整型值。指令跳过起始控制字符 ($06) 并在分隔符 (\ ) 处停止。

梯形图



结构化文本

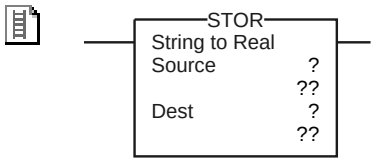
```
IF MV_read.EM THEN
     STOD(MV_msg,MV_msg_nmbr);
     MV_read.EM := 0;
END_IF;
```

字符串转换为 REAL (STOR)

STOR 指令用于将浮点值的 ASCII 表示转换为 REAL 值。

操作数：

梯形图操作数



操作数	类型	格式	键入内容	备注
源 (Source)	字符串	标签	含有 ASCII 形式的值的标签	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。
目标 (Destination)	REAL	标签	要存储 REAL 值的标签	

结构化文本

```
STOR(Source,Dest);
```

操作数与梯形图 STOR 指令的操作数相同。

说明： STOR 将 “ 源 ” 转换为 REAL 值并将结果放在 “ 目标 ” 中。

- 该指令可转换正数和负数。
- 如果 “ 源 ” 字符串含有非数字字符， STOR 将转换第一组连续数字，包括小数点 [.]：
  - 该指令将跳过任何起始控制字符或非数字字符 (数字前的负号除外)。
  - 如果字符串含有多组由分隔符 (如 /) 分隔的数字，指令将仅转换第一组数字。

算术状态标志： 算术状态标志将受到影响。

故障条件：

类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	53	输出数超出目标数据类型的限制。	以下两法任选其一： • 减小 ASCII 值的尺寸。 • 使用更大的目标数据类型。

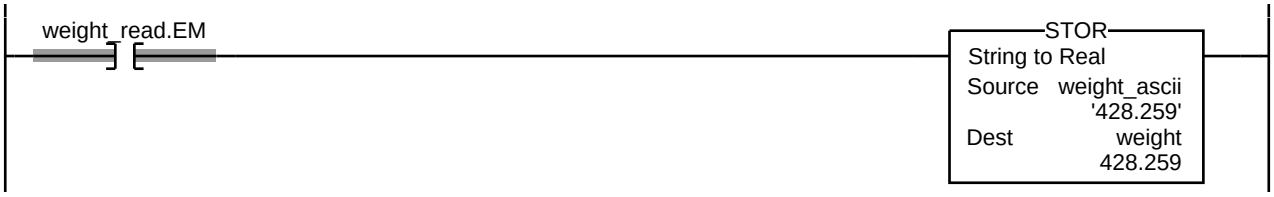
执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	S:C 置位。 “ 目标 ” 清零。 此指令对 “ 源 ” 进行转换。 如果结果为零，则 S:Z 置位	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 读取秤上的重量后 (*weight_read.EM* 置位)， STOR 指令将 *weight_ascii* 中的数字字符转换为 REAL 值。

您可能会看到 “ 源 ” 和 “ 目标 ” 的小数部分略有不同。

梯形图



结构化文本

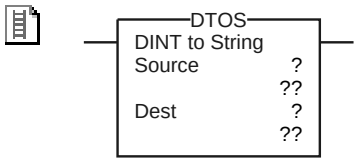
```
IF weight_read.EM THEN
     STOR(weight_ascii,weight);
     weight_read.EM := 0;
END_IF;
```

DINT 转换为字符串 (DTOS)

DTOS 指令用于生成值的 ASCII 表示。

操作数：

梯形图



操作数	类型	格式	键入内容	备注
源 (Source)	SINT INT **DINT** REAL	标签	含有该值的标签	如果 “ 源 ” 是 REAL 值，此指令将其转换为 DINT 值。请参见 REAL 转换为整型 第 648 页。
目标 (Destination)	字符串	标签	要存储 ASCII 值的标签	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。

结构化文本

```
DTOS(Source,Dest);
```

操作数与梯形图 DTOS 指令的操作数相同。

说明： DTOS 将 “ 源 ” 转换为由 ASCII 字符构成的字符串，并将结果放在 “ 目标 ” 中。

算术状态标志： 不受影响

故障条件：

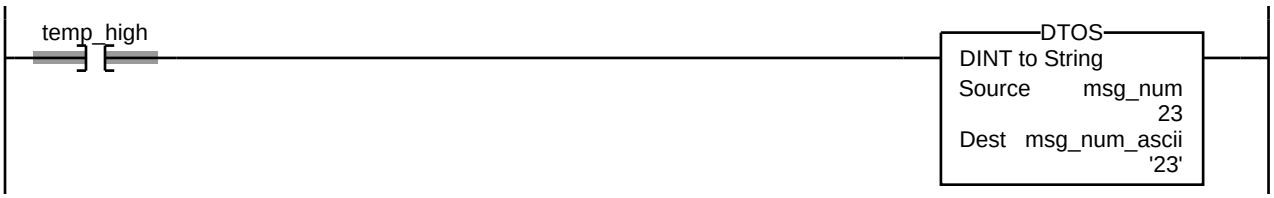
类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	52	输出字符串比 “ 目标 ” 大。	创建一个尺寸足以容纳输出字符串的新字符串数据类型。用这个新的字符串数据类型作为 “ 目标 ” 的数据类型。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令对 “ 源 ” 进行转换。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 当 *temp_high* 置位时， DTOS 指令将 *msg_num* 中的值转换为由 ASCII 字符构成的字符串，并将结果放入 *msg_num_ascii*。后面的梯级会插入 *msg_num_ascii* 或将其与其它字符串串连，以为显示终端生成完整的信息。

梯形图



结构化文本

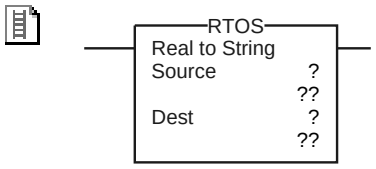
```
IF temp_high THEN

     DTOS(msg_num,msg_num_ascii);

     temp_high := 0;

END_IF;
```

REAL 转换为字符串 (RTOS)

RTOS 指令用于生成 REAL 值的 ASCII 表示。

操作数：

梯形图



操作数	类型	格式	键入内容	备注
源 (Source)	REAL	标签	含有 REAL 值的标签	
目标 (Destination)	字符串	标签	要存储 ASCII 值的标签	字符串数据类型包括： • 默认的 STRING 数据类型。 • 所创建的任何新的字符串数据类型。

结构化文本

```
RTOS(Source,Dest);
```

操作数与梯形图 RTOS 指令的操作数相同。

说明： RTOS 将 “ 源 ” 转换为由 ASCII 字符构成的字符串，并将结果放在 “ 目标 ” 中。

算术状态标志： 不受影响

故障条件：

类型	代码	原因	恢复方法
4	51	字符串标签的 LEN 值大于字符串标签的 DATA 尺寸。	1. 确认没有指令写入字符串标签的 LEN 子元素中。 2. 在 LEN 值中键入字符串包含的字符数。
4	52	输出字符串比 “ 目标 ” 大。	创建一个尺寸足以容纳输出字符串的新字符串数据类型。用这个新的字符串数据类型作为 “ 目标 ” 的数据类型。

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令对 “ 源 ” 进行转换。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 当 *send_data* 置位时， RTOS 指令将 *data_1* 中的值转换为由 ASCII 字符构成的字符串，并将结果放入 *data_1_ascii*。后面的梯级会插入 *data_1_ascii* 或将其与其它字符串串连，以为显示终端生成完整的信息。

您可能会看到 “ 源 ” 和 “ 目标 ” 的小数部分略有不同。

梯形图

结构化文本

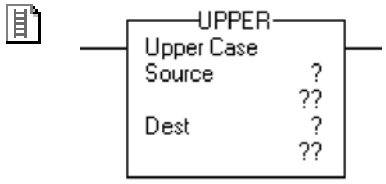
```
IF send_data THEN
     RTOS(data_1,data_1_ascii);
     send_data := 0;
END_IF;
```

大写 (UPPER)

UPPER 指令用于将字符串中的字母字符转换为大写字符。

操作数：

梯形图



操作数	类型	格式	说明
源 (Source)	字符串	标签	含有要转换为大写形式的字符的标签
目标 (Destination)	字符串	标签	要存储大写形式的字符的标签

结构化文本

```
UPPER(Source,Dest);
```

操作数与梯形图 UPPER 指令的操作数相同。

说明： UPPER 指令将 “ 源 ” 中的所有字母转换为大写形式，并将结果放在 “ 目标 ” 中。

- ASCII 字符区分大小写。大写的 “ A ” ($41) 不等于小写的 “ a ” ($61)。
- 如果操作员直接输入 ASCII 字符，请在比较这些字符前将这些字符转换为全大写或全小写形式。

“ 源 ” 字符串中的非字母字符将保持不变。

算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令将 “ 源 ” 转换为大写形式。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 为了查找特定项的信息，操作员将该项的目录号输入到 ASCII 终端。控制器从终端读取输入后 (*terminal_read.EM* 置位)，UPPER 指令将 *catalog_number* 中的字符转换为全大写形式，并将结果存储在 *catalog_number_upper_case* 中。然后，后面的梯级在数组中搜索与 *catalog_number_upper_case* 中的字符匹配的字符。

梯形图

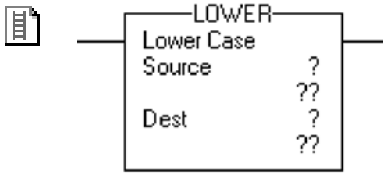
结构化文本

```
IF terminal_read.EM THEN

     UPPER(catalog_number,catalog_number_upper_case);

     terminal_read.EM := 0;

END_IF;
```

小写 (LOWER)

LOWER 指令用于将字符串中的字母字符转换为小写字符。

操作数：



梯形图

操作数	类型	格式	说明
源 (Source)	字符串	标签	含有要转换为小写形式的字符的标签
目标 (Destination)	字符串	标签	要存储小写形式的字符的标签

```
LOWER(Source,Dest);
```

结构化文本

操作数与梯形图 LOWER 指令的操作数相同。

说明： LOWER 指令将 “ 源 ” 中的所有字母转换为小写形式，并将结果放在 “ 目标 ” 中。

- ASCII 字符区分大小写。大写的 “ A ” ($41) 不等于小写的 “ a ” ($61)。
- 如果操作员直接输入 ASCII 字符，请在比较这些字符前将这些字符转换为全大写或全小写形式。

“ 源 ” 字符串中的非字母字符将保持不变。

算术状态标志： 不受影响

故障条件： 无

执行：

条件	梯形图操作	结构化文本操作
预扫描	梯级输出条件设置为假。	不执行任何操作。
梯级输入条件为假	梯级输出条件设置为假。	N/A
梯级输入条件为真	执行该指令。 梯级输出条件设置为真。	N/A
EnableIn 置位	N/A	EnableIn 始终置位。 执行该指令。
指令执行	此指令将 “ 源 ” 转换为小写形式。	
后扫描	梯级输出条件设置为假。	不执行任何操作。

示例： 为了查找特定项的信息，操作员将项编号输入到 ASCII 终端。控制器从终端读取输入后 (*terminal_read.EM* 置位)， LOWER 指令将 *item_number* 中的字符转换为全小写形式，并将结果存储在 *item_number_lower_case* 中。然后，后面的梯级在数组中搜索与 *item_number_lower_case* 中的字符匹配的字符。

梯形图

结构化文本

```
IF terminal_read.EM THEN

     LOWER(item_number,item_number_lower_case);

     terminal_read.EM := 0;

END_IF;
```

调试指令
(BPT、 TPT)

简介

使用调试指令可监视逻辑在所确定的条件下的状态。这些指令仅与 RSLogix Emulate 5000 软件兼容，利用这款软件，您可以在个人计算机上对 Logix5000
控制器进行仿真。

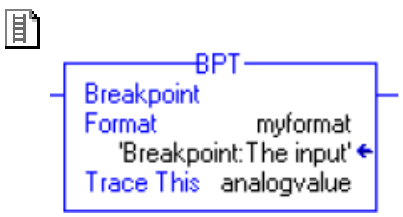
如果要	使用以下指令	可用编程语言	页码
在梯级为真时停止程序仿真	BPT	梯形图	635
在梯级为真时记录所选择的数据	TPT	梯形图	639

断点 (BPT)

断点用于在梯级为真时停止程序仿真。

操作数：

梯形图



操作数	类型	格式	说明
格式 (Format)	字符串	标签	为断点的跟踪窗口中显示的文本设置格式的字符串。
跟踪此项 (Trace This)	BOOL、 SINT、 INT、 DINT、 REAL	标签	具有要在跟踪窗口中显示的值的标签。

说明：

断点用断点输出指令 (BPT) 编程。当含有 BPT 指令的梯级上的输入为真时，
BPT 指令将停止程序的执行。软件将显示一个窗口，指示所触发的断点以及触发该断点的值。

断点触发后，仿真器将显示一个窗口，通知您出现了断点。窗口的标题栏将显示含有遇到断点的仿真器的插槽。

单击“ 确定 ”(OK) 后，仿真器继续执行程序。如果触发断点的条件依然存在，断点将再次出现。

而且，仿真器将打开断点的跟踪窗口。跟踪窗口将显示断点和值的信息。

重要事项	断点触发后，您将无法对项目进行编辑，直到允许程序继续执行。您可以在线连接到仿真器来观察项目的状态，但无法对项目进行编辑。如果试图在触发了断点的情况下接受梯级编辑，您将看到一个对话框，上面显示控制器的模式不正确。

字符串格式

通过跟踪点和断点指令中的“ 格式 ”字符串，可以控制被跟踪的标签在跟踪窗口或断点窗口中的显示方式。字符串的格式为：

heading:(text)%(type)

其中 *heading* 是确定跟踪点或断点的文本字符串， *text* 是描述标签 (或所选择的其它任何文本) 的字符串， %(*type*) 表示标签的格式。对于正在通过跟踪点指令或断点指令跟踪的每个标签，您都需要一个类型指示符。

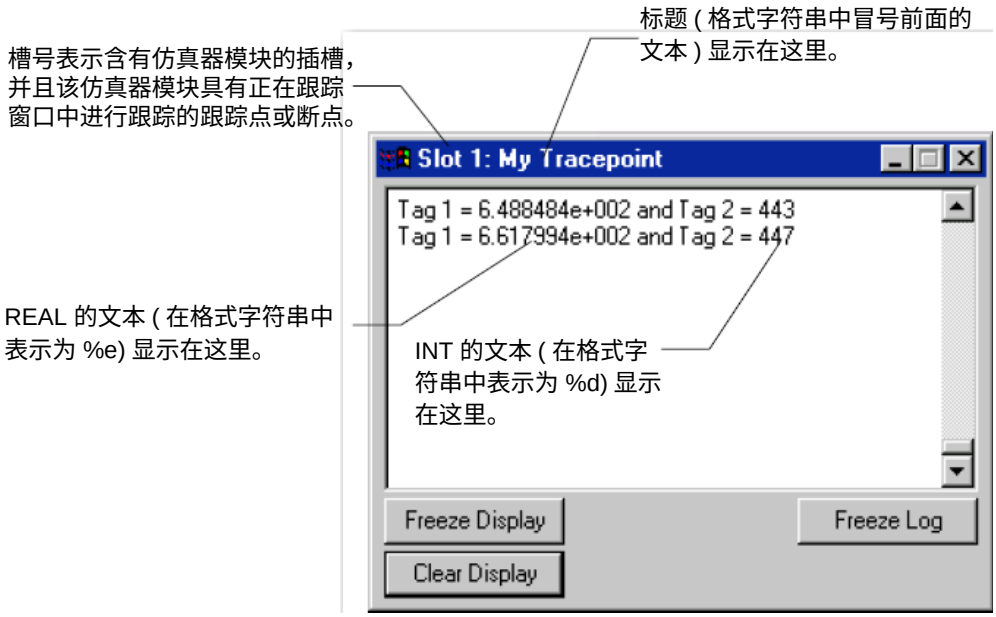
例如，可以按如下所示格式化跟踪点字符串。

My tracepoint:Tag 1 = %e and Tag 2 = %d

%e 将第一个所跟踪标签的格式设为带指数的双精度浮点型， %d 将第二个所跟踪标签的格式设为有符号的十进制整型。

这种情况下，您的跟踪点指令将具有两个“ 跟踪此项 ”操作数 (一个用于 REAL，一个用于 INT，尽管任何标签的值都能通过任何标记来格式化)。

跟踪点触发时显示的生成的跟踪窗口跟示例中的相似。



算术状态标志： 不受影响

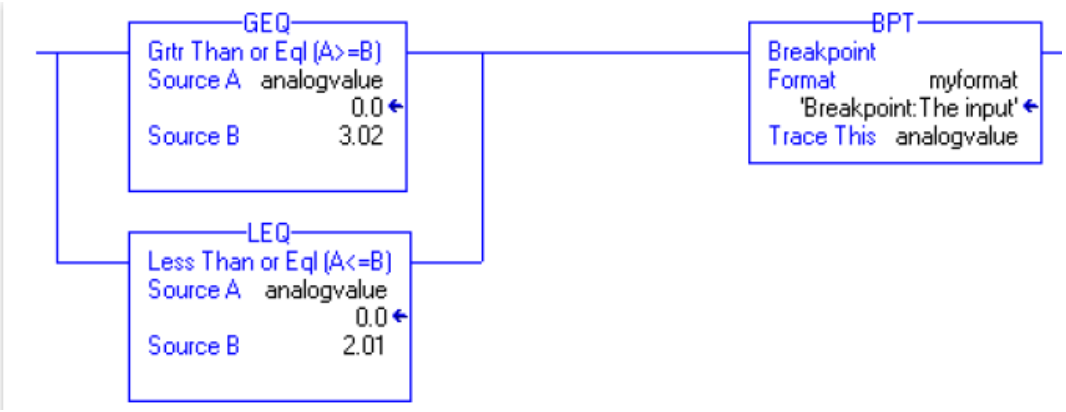
故障条件： 无

执行：

条件	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	梯级输出条件设置为真。 执行跳转到包含带参考标签名称的 LBL 指令的梯级。
后扫描	梯级输出条件设置为假。

示例： 可以使用 BPT 指令显示许多标签值。但是，格式化字符串只能含有 82 个字符。由于在格式化字符串中，要位于断点中的每个标签需要两个字符，因此无法用一条 BPT 指令跟踪 41 个以上的标签。不过，要分隔跟踪中的标签数据，需要在其中包含空格和其它格式，从而将一条 BPT 指令可有效显示的标签值的数量减至远小于 41。

此梯级显示的是在模拟值大于 3.02 或小于 2.01 时使程序停止执行的断点。

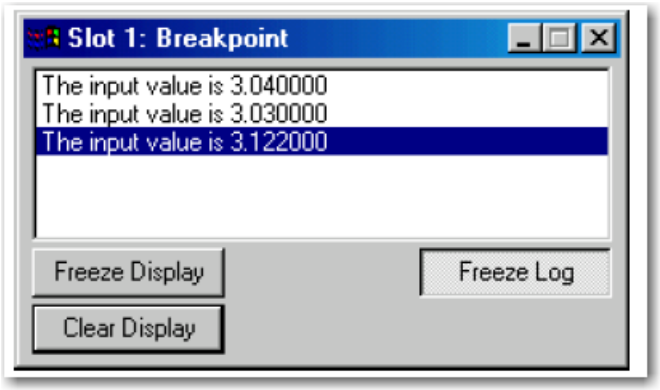


您想要在 “ 格式 ” 字符串 (myformat) 中显示断点信息。这种情况下，格式字符串含有以下文本：

```
Breakpoint:The input value is %f
```

断点触发后，断点跟踪窗口的标题栏中显示冒号前面的字符 (“ Breakpoint ”)。其它字符将构成跟踪。本例中， %f 表示要跟踪的第一个 (在本例中也是惟一一个) 标签 (“ analogvalue ”)。

生成的跟踪结果如下所示。

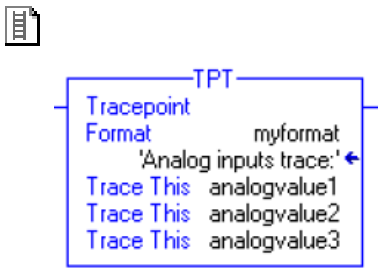


跟踪点 (TPT)

跟踪点用于在梯级为真时记录所选的数据。

操作数：

梯形图



操作数	类型	格式	说明
格式 (Format)	字符串	标签	为跟踪报告 (包括屏幕上的和记录在磁盘上的) 设置格式的字符串。
跟踪此项 (Trace This)	BOOL、SINT、 INT、DINT、REAL	标签	要跟踪的标签。

说明：

跟踪点是通过跟踪点输出指令 (TPT) 来编程的。当含有 TPT 指令的梯级上的输入为真时，
TPT 指令将跟踪条目写入跟踪显示画面或日志文件。

您可以利用 TPT 指令跟踪许多标签。但是，格式化字符串只能含有 82 个字符。由于在格式化字符串中，要跟踪的每个标签需要两个字符，因此无法用一条 TPT 指令跟踪 41 个以上的标签。不过，要分隔跟踪中的标签数据，需要在其中包含空格和其它格式，从而将一条 TPT 指令可有效跟踪的标签的数量减至远小于 41。

字符串格式

通过跟踪点和断点指令中的 “ 格式 ” 字符串，可以控制被跟踪的标签在跟踪窗口或断点窗口中的显示方式。字符串的格式如下所示：

heading:(text)%(type)

其中 *heading* 是确定跟踪点或断点的文本字符串， *text* 是描述标签 (或所选择的其它任何文本) 的字符串， %(*type*) 表示标签的格式。对于正在通过跟踪点指令或断点指令跟踪的每个标签，您都需要一个类型指示符。

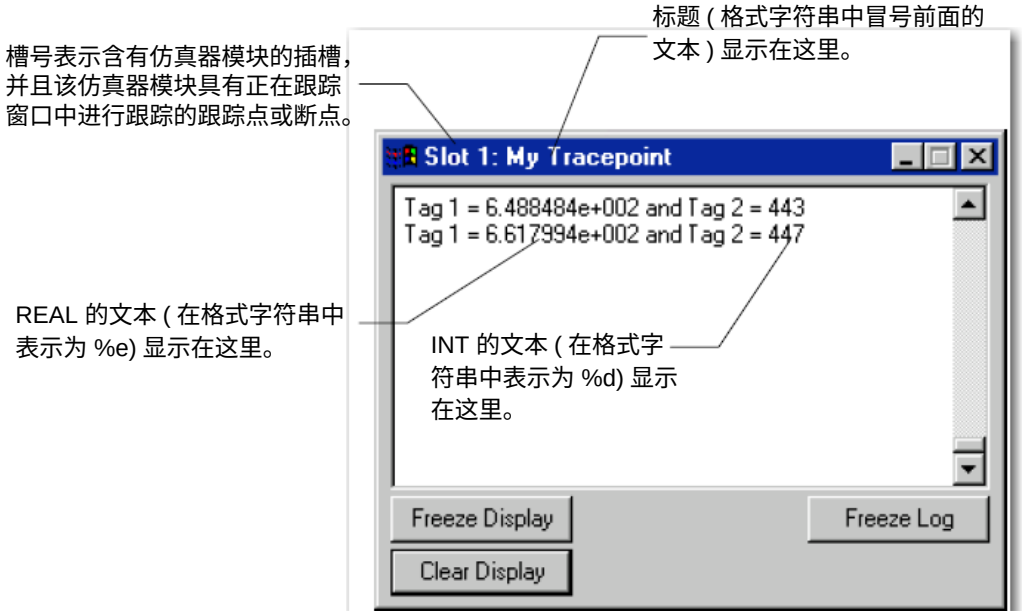
例如，可以按如下所示格式化跟踪点字符串。

My tracepoint:Tag 1 = %e and Tag 2 = %d

%e 将第一个所跟踪标签的格式设为带指数的双精度浮点型， %d 将第二个所跟踪标签的格式设为有符号的十进制整型。

这种情况下，您的跟踪点指令将具有两个 “ 跟踪此项 ” 操作数 (一个用于 REAL，一个用于 INT，尽管任何标签的值都能通过任何标记来格式化)。

跟踪点触发时显示的生成的跟踪窗口跟示例中的相似。



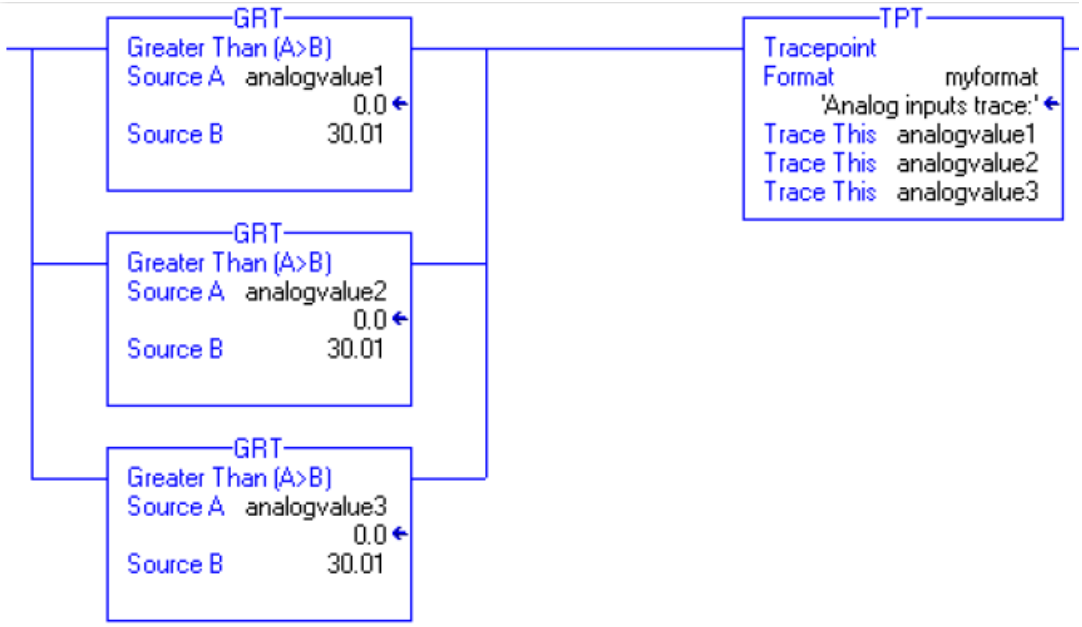
算术状态标志： 不受影响

故障条件： 无

执行：

条件：	梯形图操作
预扫描	梯级输出条件设置为假。
梯级输入条件为假	梯级输出条件设置为假。
梯级输入条件为真	梯级输出条件设置为真。 执行跳转到包含带参考标签名称的 LBL 指令的梯级。
后扫描	梯级输出条件设置为假。

示例： 此梯级在三个模拟值中的任意一个超出给定值 (30.01) 时，触发对这三个模拟值的跟踪。

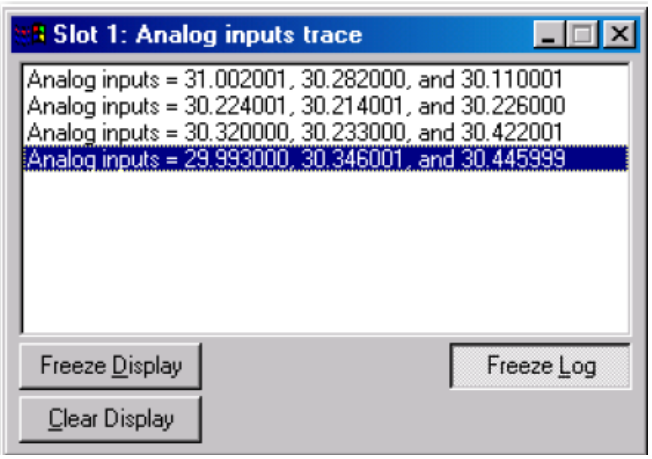


您想要在 “ 格式 ” 字符串 (myformat) 中显示跟踪点信息。这种情况下，格式字符串含有以下文本：

```
Analog inputs trace:Analog inputs = %f, %f, and %f
```

跟踪点触发后，冒号前面的字符 (“ Analog inputs trace ”) 将显示在跟踪窗口的标题栏中。其它字符将构成跟踪。本例中， %f 表示要跟踪的标签 (“ analogvalue1 ”、“ analogvalue2 ” 和 “ analogvalue3 ”)。

生成的跟踪结果如下所示。



此跟踪记录到磁盘时，冒号前面的字符将显示在跟踪中。

这可以指示出哪个跟踪点引起了哪个跟踪条目。下面是一个跟踪条目示例。“ Analog inputs trace: ” 是跟踪点的格式字符串的标题文本。

Analog inputs trace:Analog inputs = 31.00201, 30.282000, and 30.110001。

通用属性

简介

本附录介绍了 Logix 指令的通用属性。

立即值

以十进制格式键入立即值 (常数) 时 (例如 -2、 3)，控制器将使用 32 位存储该值。如果以非十进制基数格式 (如二进制或十六进制) 键入值，并且没有指定全部 32 位，则控制器会将未指定的位设为零 (填零)。

举 例 立即值的填零

如果键入	控制器将存储
-1	16#ffff ffff (-1)
16#ffff (-1)	16#0000 ffff (65535)
8#1234 (668)	16#0000 029c (668)
2#1010 (10)	16#0000 000a (10)

数据转换

如果在编程中混用数据类型，则会发生数据转换。

编程对象	在以下情况下会发生转换
继电器梯形图逻辑	一条指令内的各参数混用数据类型
功能块	连接具有不同数据类型的两个参数

如果指令的所有操作数都符合以下要求，则指令执行速度更快且所需内存更少：

- 使用相同数据类型。
- 使用最佳数据类型：
  - 在本手册中每条指令的 “ 操作数 ” 部分，粗体显示的数据类型表示最佳数据类型。
  - DINT 和 REAL 数据类型是常见的最佳数据类型。
  - 大多数功能块指令只支持其操作数使用一种数据类型 (最佳数据类型)。

如果混用数据类型并使用不是最佳数据类型的标签，则控制器将根据以下规则转换数据。

- 有操作数是 REAL 值吗？

如果	则键入操作数 (例如，源、表达式中的标签、限值) 将转换为
是	REAL
否	DINT

- 指令执行后，结果 (DINT 或 REAL 值) 将根据情况转换为目标数据类型。

不能在对整型或 REAL 数据类型执行操作的指令中指定 BOOL 标签。

由于数据转换需要额外的时间和内存，因此可通过以下方法提高程序的效率：

- 在整个指令中使用同一种数据类型。
- 尽可能少用 SINT 或 INT 数据类型。

换言之，在指令中全部使用 DINT 标签或全部使用 REAL 标签以及立即值。

以下各部分阐述了使用 SINT 或 INT 标签或混用数据类型时如何转换数据。

SINT 或 INT 转换为 DINT

对于将 SINT 或 INT 值转换为 DINT 值的指令，本手册中的 “ 操作数 ” 部分指明了转换方法。

转换方法	通过以下操作转换数据
符号扩展	将最左位 (值的符号) 的值填入现有位左侧的各个位，直到填满 32 位。
填零	将零填入现有位的左侧，直到填满 32 位。

下面的示例显示了使用符号扩展和填零来转换值的结果。

值	2#1111_1111_1111_1111	(-1)
通过符号扩展转换为此值	2#1111_1111_1111_1111_1111_1111_1111_1111	(-1)
通过填零转换为此值	2#0000_0000_0000_0000_1111_1111_1111_1111	(65535)

由于立即值都是填零的，因此 SINT 或 INT 值的转换可能产生意外的结果。在下面的示例中，由于 Source A (INT) 通过符号扩展转换，而 Source B(立即值) 通过填零转换，因此比较结果为假。

如果在通过符号扩展转换数据的指令中使用 SINT 或 INT 标签和立即值，请使用以下方法之一处理立即值。

- 以十进制基数指定任意立即值。
- 如果以非十进制基数格式键入值，则应指定立即值的全部 32 位。为此，将最左位的值填入其左侧的各个位，直到填满 32 位。
- 为每个操作数创建一个标签，并在整个指令中使用同一种数据类型。要分配常数值，请执行以下操作之一：
  - 将其键入到一个标签中。
  - 添加 MOV 指令将该值移入一个标签中。
- 使用 MEQ 指令，只检查需要的位。

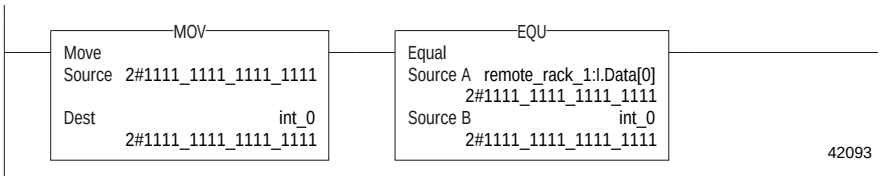
下面的示例显示了混用立即值与 INT 标签的两种方法。这两个示例均检查 1771 I/O 模块的各个位，以确定是否所有位都已置位。由于 1771 I/O 模块的键入数据字是 INT 标签，因此使用 16 位常数值最为简单。

举 例 混用 INT 标签与立即值

由于 *remote_rack_1:I.Data[0]* 是 INT 标签，因此用于检查该标签的值也以 INT 标签形式输入。

举 例 混用 INT 标签与立即值

由于 *remote_rack_1:I.Data[0]* 是 INT 标签，因此用于检查该标签的值首先移入 *int_0*(也是 INT 标签) 中。然后， EQU 指令将对这两个标签进行比较。



整型转换为 REAL

控制器以 IEEE 单精度浮点数格式存储 REAL 值。它使用 1 位表示值的符号，23 位表示底数，8 位表示指数 (总共 32 位)。如果在同一指令中将整型标签 (SINT、 INT 或 DINT) 和 REAL 标签作为输入混合使用，则控制器会在执行指令前将整型值转换为 REAL 值。

- SINT 或 INT 值总是转换为相同的 REAL 值。
- DINT 值不一定能转换为相同的 REAL 值：
  - REAL 值最多使用 24 位来表示底数 (23 个存储位加上 1 个 “ 隐藏 ” 位)。
  - DINT 值最多使用 32 位来表示值 (一位表示符号，31 位表示值)。
  - 如果 DINT 值需要 24 个以上的有效位，则*可能无法*转换为相同的 REAL 值。如果不能，则控制器将利用 24 个有效位舍入为最接近的 REAL 值。

DINT 转换为 SINT 或 INT

为了将 DINT 值转换为 SINT 或 INT 值，控制器会截断 DINT 的上部，并在必要时置位溢出状态标志。下面的示例显示的是 DINT 到 SINT 或 INT 转换的结果。

举 例　DINT 到 INT 和 SINT 的转换

DINT 值	转换为此较小的值	
16#0001_0081 (65,665)	INT：	16#0081 (129)
	SINT：	16#81 (-127)

REAL 转换为整型

为了将 REAL 值转换为整型值，控制器将舍入小数部分，并截断非小数部分的高位部分。如果数据丢失，则控制器将置位溢出状态标志。数字舍入方式如下。

- *X*.5 以外的数字舍入为最接近的整数。
- *X*.5 舍入到最接近的偶数。

下面的示例显示了将 REAL 值转换为 DINT 值的结果。

举例 REAL 值到 DINT 值的转换

REAL 值	转换为此 DINT 值
-2.5	-2
-1.6	-2
-1.5	-2
-1.4	-1
1.4	1
1.5	2
1.6	2
2.5	2

重要事项 算术状态标志是否置位取决于存储的值。如果由于混用指令参数的数据类型而发生类型转换，则通常不影响算术状态关键字的指令也可能产生影响。类型转换过程会设置算术状态关键字。

功能块属性

简介

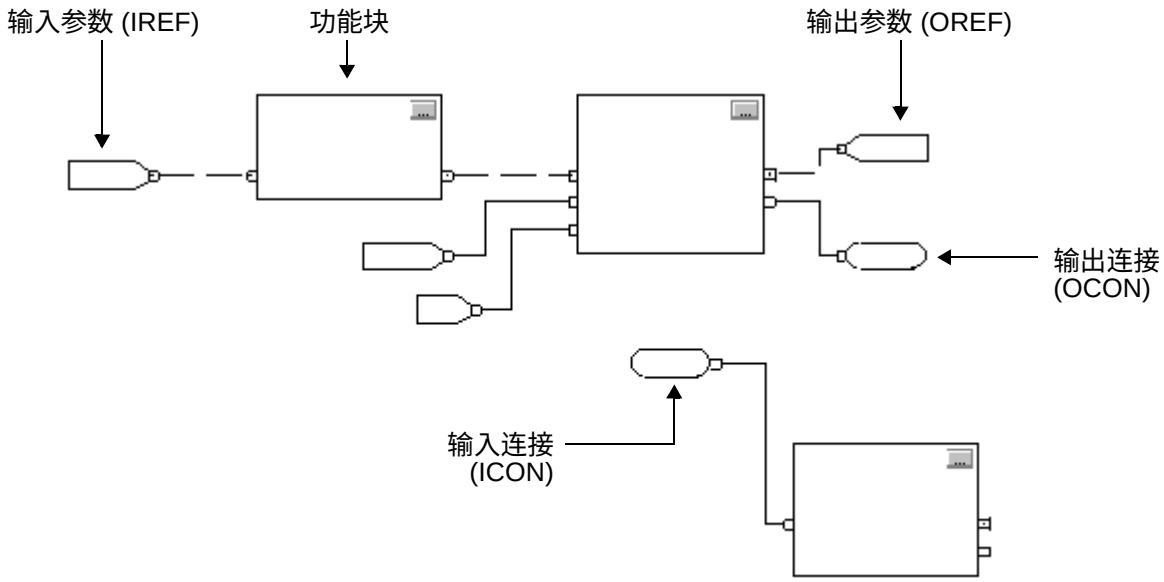
本附录介绍功能块指令所特有的问题。查阅本附录中的信息以确保您理解功能块例程的运行方式。

重要事项

在功能块中编程时，将工程单位的范围限制为 +/-$10^{+/-15}$，这是因为内部浮点计算是通过使用单精度浮点来完成的。如果结果接近单精度浮点 (+/-$10^{+/-38}$) 的限度，此范围以外的工程单位可能造成精度降低。

功能块元素

要控制设备，请使用以下元素。

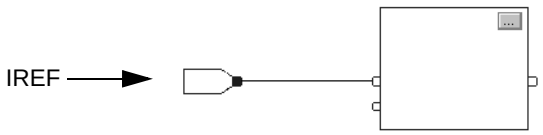


使用下表来选择功能块元素。

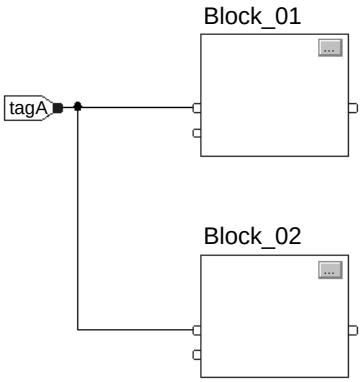
如果要	则使用
由输入设备或标签提供值	输入参数 (IREF)
将值发送给输出设备或标签	输出参数 (OREF)
对一个或多个输入值执行操作并生成一个或多个输出值	功能块
当功能块符合下列条件时，在功能块间传送数据： • 在同一版面内相隔较远。 • 在同一例程内的不同版面上。	输出连接 (OCON) 和输入连接 (ICON)
将数据分散到例程中的多个点上	单个输出连接 (OCON) 和多个输入连接 (ICON)

锁存数据

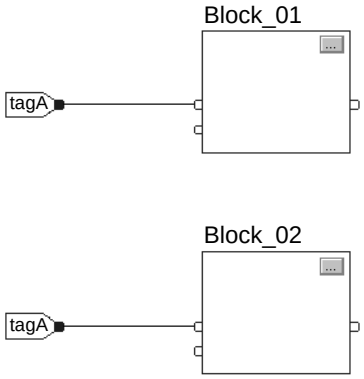
如果用户使用 IREF 为功能块指令指定输入数据，为了扫描功能块例程，将锁存该 IREF 中的数据。 IREF 将锁存程序域标签和控制器域标签中的数据。控制器在每次扫描开始时都会更新所有的 IREF 数据。



在本示例中，开始执行例程时存储 tagA 的值。执行 Block_01 时，使用所存储的值。执行 Blcock_02 时，也使用相同的存储值。如果在例程执行期间 tagA 的值改变，则 IREF 内 tagA 的存储值将在下一次执行例程时改变。 .

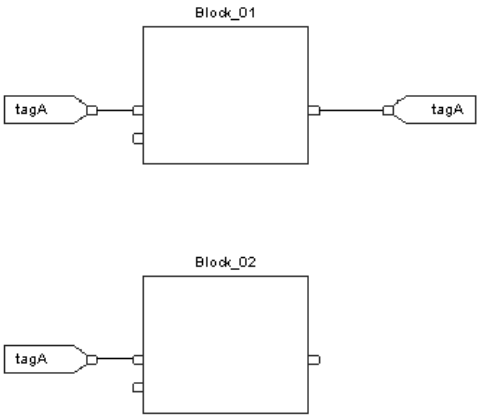


这个例子和上面的例子相同。 tagA 的值只有在开始执行例程时才被存储一次。在例程的整个运行过程中，例程将一直使用该存储值。



从 RSLogix 5000 软件的版本 11 开始，用户可以在同一例程中的多个 IREF 和一个 OREF 中使用同一个标签。由于每次扫描例程时 IREF 中的标签值都被锁存，所以所有的 IREF 都将使用相同的值，即使在例程的运行过程中 OREF 获得不同的标签值。

在本示例中，如果当例程开始执行此次扫描时 tagA 的值是 25.4，并且 Block_01 将 tagA 的值改为 50.9，那么当 Block_02 执行本次扫描时，接入 Block_02 的第二个 IREF 将仍使用值 25.4。只有在下次扫描开始后， tagA 的新值 50.9 才能被此例程中的 IREF 使用。



执行顺序

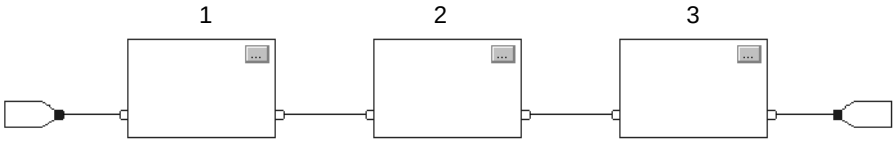
当用户执行如下操作时， RSLogix 5000 编程软件将自动确定例程中的功能块的执行顺序：

- 校验功能块例程。
- 校验包含功能块例程的项目。
- 下载包含功能块例程的项目。

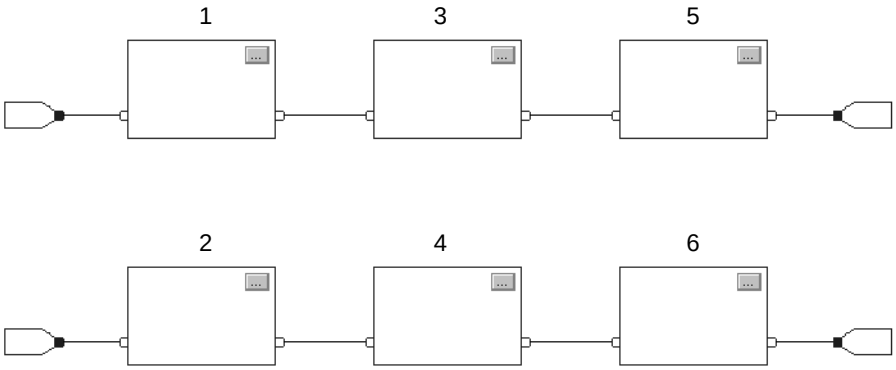
必要时，用户可以通过将功能块连接在一起并指示各反馈线的数据流来定义执行顺序。

如果功能块没有连接在一起，则哪个块第一个执行均无关紧要。此时，块间无数据流。

如果用户将块顺序连接，执行顺序将从输入移向输出。块的输入要求数据在控制器执行该块前可用。例如，块 2 必须在块 3 前执行，因为块 2 的输出向块 3 提供输入。

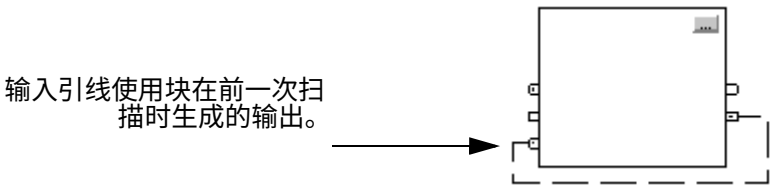


执行顺序只和连在一起的块有关。下面的示例就很好，因为这两组块并没有接在一起。特定组内的块将按照适当顺序 (与该组内的块有关) 执行。

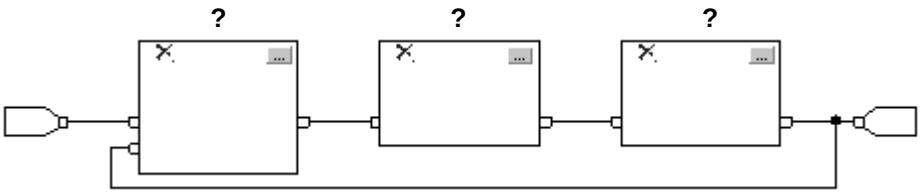


解析回路

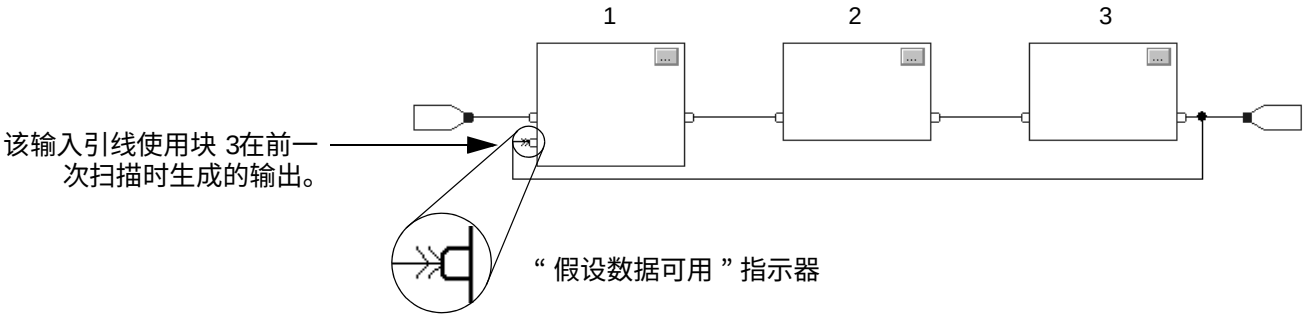
要在块的周围创建反馈回路，需要将块的一个输出引线同该块的一个输入引线相连接。下面的示例是可行的。该回路只包含一个单独的块，所以不必考虑执行顺序。



如果回路中包含一组块，控制器将无法确定哪个块先执行。换言之，它无法解析该回路。

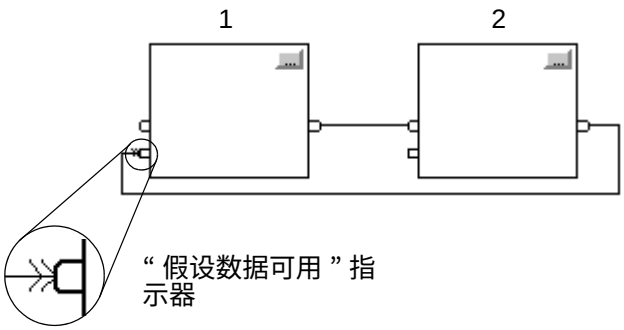
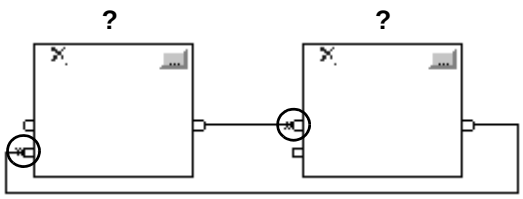


要确定先执行哪个块，需要用 *“假设数据可用”(Assume Data Available)* 指示器对创建回路的输入线 (反馈线) 进行标记。在下面的示例中，块 1 使用前一次执行例程时块 3 生成的输出。



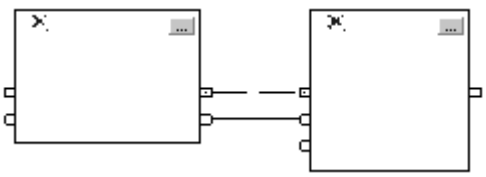
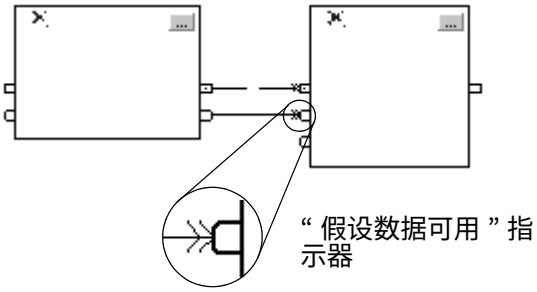
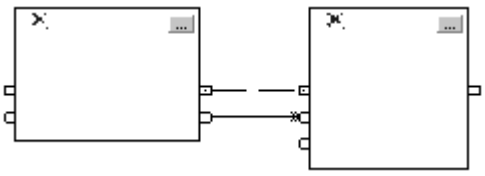
“假设数据可用”(Assume Data Available) 指示器定义回路中的数据流。箭头指示该数据用作回路中第一个块的输入。

不**要**使用“ 假设数据可用 ”(Assume Data Available) 指示器来标记回路中的所有线。

可以	不可以
 “ 假设数据可用 ” (Assume Data Available) 指示器定义回路中的数据流。	 由于所有的线都使用“ 假设数据可用 ” (Assume Data Available) 指示器，所以控制器无法解析该回路。

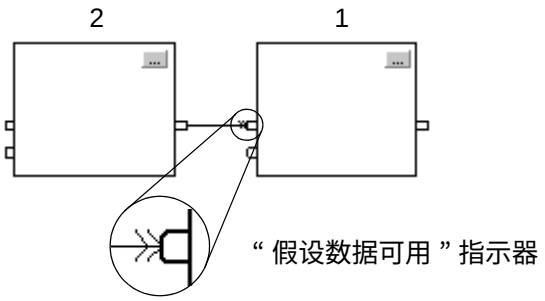
解析两个块之间的数据流

如果用户用两根或两根以上的线来连接两个块，那么对于这两个块之间的所有连线，请都使用相同的数据流指示器。

可以	不可以
 两根线都没有使用“ 假设数据可用 ” (Assume Data Available) 指示器。  两根线都用了“ 假设数据可用 ” (Assume Data Available) 指示器。	 一根线使用“ 假设数据可用 ” (Assume Data Available) 指示器，而另一根没有使用。

创建一个扫描延迟

要在块间生成一次扫描延迟，使用 *“ 假设数据可用 ” (Assume Data Available)* 指示器。在下面的示例中，块 1 先执行。它使用在前一次扫描例程时生成的块 2 的输出。



总结

功能块例程按以下顺序执行。

1. 控制器锁存 IREF 中的所有数据值。
2. 控制器按照由连接方式确定的顺序执行其它功能块。
3. 控制器将输出写入到 OREF 中。

功能块对溢出条件的响应

通常，存有历史记录的功能块指令在发生溢出时不会使用 ± NAN 或 ± INF 值来更新历史记录。每条指令都会对溢出条件产生以下响应之一。

响应 1 块执行其算法并检查 ± NAN 或 ± INF 的结果。如果 ± NAN 或 ± INF，块将输出 ± NAN 或 ± INF。	响应 2 具有输出限制的块执行其算法并检查 ± NAN 或 ± INF 的结果。输出限值由输入参数 HighLimit 和 LowLimit 来确定。如果 ± INF，块将输出受限的结果。如果 ± NAN，则不会使用输出限值，并且块将输出 ± NAN。	响应 3 溢出条件不适用。这些指令通常具有布尔输出。
ALMNTCH	HLL	BANDOSRI
DEDTPMUL	INTG	BNOTRESD
DERVPOSP	PI	BORRTOR
ESELRLIM	PIDE	BXORSETD
FGENRMPS	SCL	CUTDTOFR
HPFSCRV	SOC	D2SDTONR
LDL2SEL		D3SD
LDLGSNEG		DFF
LPFSRTP		JKFF
MAVESSUM		OSFI
MAXCTOT		
MINCUPDN		
MSTD		
MUX		

计时模式

这些过程控制和驱动指令支持不同的计时模式。

DEDT	LDLG	RLIM
DERV	LPF	SCRV
HPF	NTCH	SOC
INTG	PI	TOT
LDL2	PIDE	

有三种不同的计时模式。

计时模式	说明	
周期性	周期性模式是默认模式，适合大多数控制应用。我们建议您将采用该模式的指令放在在周期性任务中执行的例程中。指令的增量时间 (DeltaT) 按以下方式确定：	
	指令的执行位置	**DeltaT** 的大小
	周期性任务	任务的周期
	事件或连续任务	从上次执行起经历的时间 控制器将经过的时间截断为整毫秒 (ms)。例如，如果经过的时间等于 10.5 ms，则控制器将 DeltaT 设置为 10 ms。
	过程输入的更新需要和任务的执行同步，或其采样速度需要比任务的执行速度快 5-10 倍，以便将输入与指令之间出现采样错误的几率降到最低。	
密集采样	在密集采样模式下，指令使用的增量时间 (DeltaT) 是写入指令的 OversampleDT 参数的值。如果过程输入具有时间戳值，则需要使用实时采样模式。 执行指令时，向要控制的程序添加逻辑。例如，可以使用设置为 OversampleDeltaT 值的计时器并通过指令的 EnableIn 输入来控制执行。 过程输入的采样速度需要比指令的执行速度快 5-10 倍，以便将输入与指令之间出现采样错误的几率降到最低。	
实时采样	在实时采样模式下，指令使用的增量时间 (DeltaT) 是两个对应于过程输入更新的时间戳值的差。当过程输入具有一个与更新关联的时间戳而且您需要精确协调时，使用此模式。 时间戳的值将从为指令的 RTSTimeStamp 参数所输入的标签名称中读取。通常，此标签名称是与过程输入关联的输入模块上的一个参数。 指令将组态后的 RTSTime 值 （预期的更新周期）与计算得到的 DeltaT 进行比较，以确定过程输入的每次更新是否都被指令读取。如果 DeltaT 不在 1 毫秒的组态时间内，指令将设置 RTSMissed 状态位，以表明读取模块上输入的更新时出现了问题。	

基于时间的指令要求 DeltaT 是一个常量值，以便控制算法能够正确地计算过程输出。如果 DeltaT 发生变化，则过程输出中将出现不连续的现象。不连续的严重程度取决于指令和 DeltaT 的变化范围。

如果出现以下情况，将出现不连续现象：

- 扫描期间未执行指令。
- 任务运行期间多次执行指令。
- 任务正在运行时，任务的扫描速率或过程输入的采样时间发生变化。
- 任务正在运行时，用户更改时基模式。
- 任务正在运行时，顺序参数在筛选器块上被更改。

  更改顺序参数会在指令内选择其它控制算法。

计时模式的常用指令参数

支持时基模式的指令具有以下输入参数和输出参数。

输入参数

输入参数	数据类型	说明
TimingMode	DINT	选择计时执行模式。 值： 说明： 0 周期性模式 1 密集采样模式 2 实时采样模式 有效值 = 0…2 默认值 = 0 当 TimingMode = 0 且任务是周期性任务时，将使能周期计时， DeltaT 将设置为任务的扫描速率。当 TimingMode = 0 且任务是事件或连续任务时，将使能周期计时， DeltaT 将设置为从上次执行指令起经历的时间长度。 当 TimingMode = 1 时，将使能密集采样计时， DeltaT 将设置为参数 OversampleDT 的值。 当 TimingMode = 2 时，将使能实时采样计时， DeltaT 将设置为从与输入关联的模块中读取的当前时间戳值和先前的时间戳值的差。 如果 TimingMode 无效，指令将置位 Status 中的相应位。
OversampleDT	REAL	密集采样计时的执行时间。用于 DeltaT 的值以秒为单位。如果 TimingMode = 1， OversampleDT = 0.0 将禁止控制算法的执行。如果 TimingMode 无效，指令将设置 DeltaT = 0.0，并且置位 Status 中的相应位。 有效值 = 0…4194.303 秒 默认值 = 0.0
RTSTime	DINT	实时采样计时的模块更新周期。预期的 DeltaT 更新周期以毫秒为单位。更新周期通常是用于组态模块的更新时间的值。如果无效，指令将置位 Status 中的相应位，并禁止 RTSMissed 校验。 有效值 = 1…32,767ms 默认值 = 1

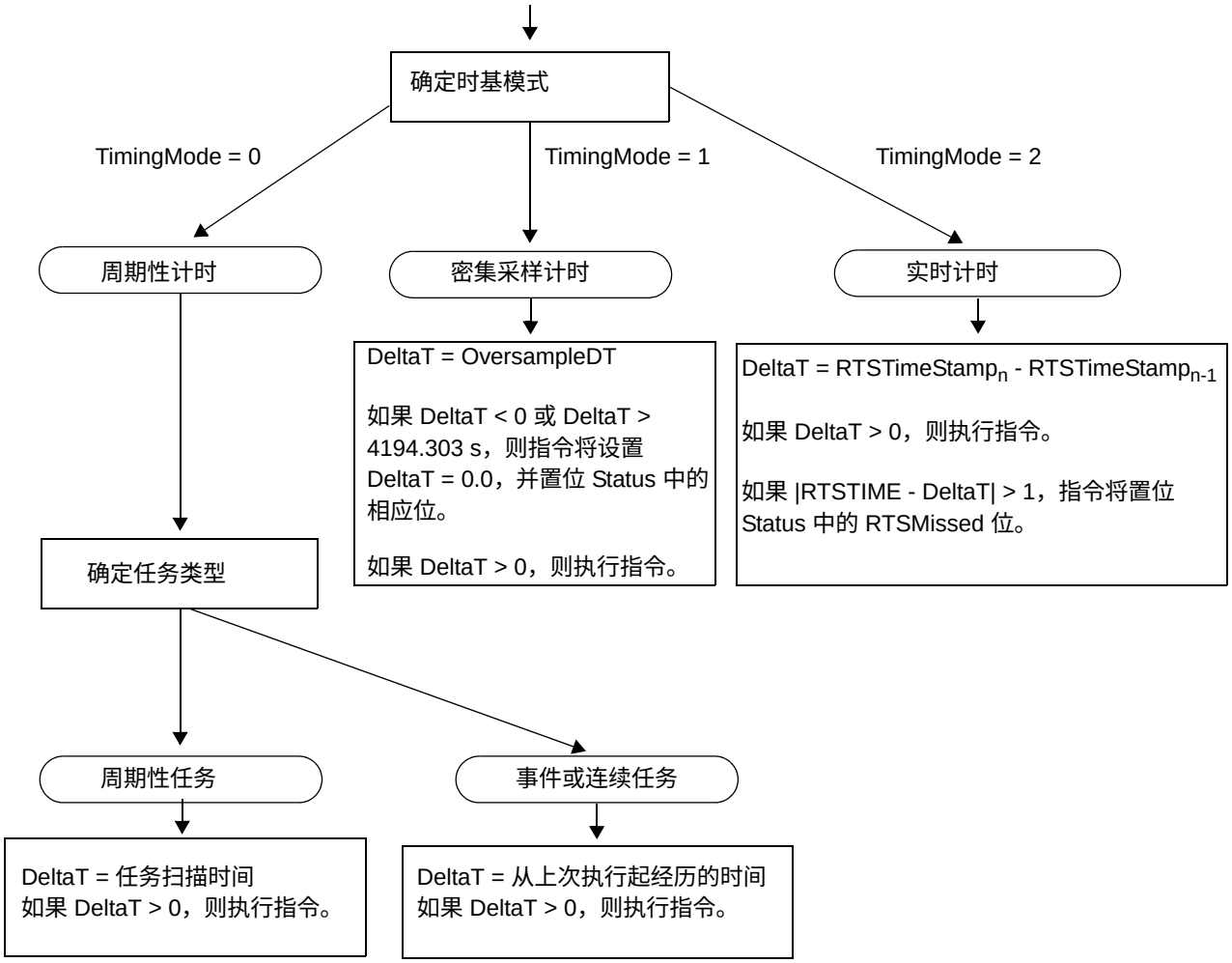
输入参数	数据类型	说明
RTSTimeStamp	DINT	实时采样计时的模块时间戳值。对应于输入信号的上次更新的时间戳值。该值用于计算 DeltaT。如果无效，指令将置位 Status 中的相应位，并禁止控制算法的执行和 RTSMissed 校验。 有效值 = 1...32,767ms （从 32767 绕到 0） 1 计数 = 1 毫秒 默认值 = 0

输出参数

输出参数	数据类型	说明
DeltaT	REAL	两次更新之间经历的时间。这是控制算法计算过程输出时所用的经历的时间，单位是秒。 周期性：如果任务是周期性任务，则 DeltaT = 任务的扫描速率；如果任务是事件或连续任务，则 DeltaT = 从上次执行指令起经历的时间 密集采样：DeltaT = OversampleDT 实时采样：$DeltaT = (RTSTimeStamp_n - RTSTimeStamp_{n-1})$
Status	DINT	功能块的状态。
TimingModeInv (Status.27)	BOOL	无效的 TimingMode 值。
RTSMissed (Status.28)	BOOL	仅在实时采样模式下使用。当 ABS | DeltaT - RTSTime | > 1(.001 秒) 时置位。
RTSTimeInv (Status.29)	BOOL	无效的 RTSTime 值。
RTSTimeStampInv (Status.30)	BOOL	无效的 RTSTimeStamp 值。
DeltaTInv (Status.31)	BOOL	无效的 DeltaT 值。

计时模式概览

下表显示指令如何确定合适的计时模式。



程序 / 操作员控制

多个指令支持程序 / 操作员控制的概念。这些指令包括：

- 增强型选择 (ESEL)
- 累加器 (TOT)
- 增强型 PID (PIDE)
- 斜坡 / 保持 (RMPS)
- 离散 2 态设备 (D2SD)
- 离散 3 态设备 (D3SD)

程序 / 操作员控制允许用户同时从用户程序和操作员界面设备执行对这些指令的控制。当处于程序控制时，指令由指令的程序输入来控制；当处于操作员控制时，指令由指令的操作员输入来控制。

程序或操作员控制通过使用以下输入来确定。

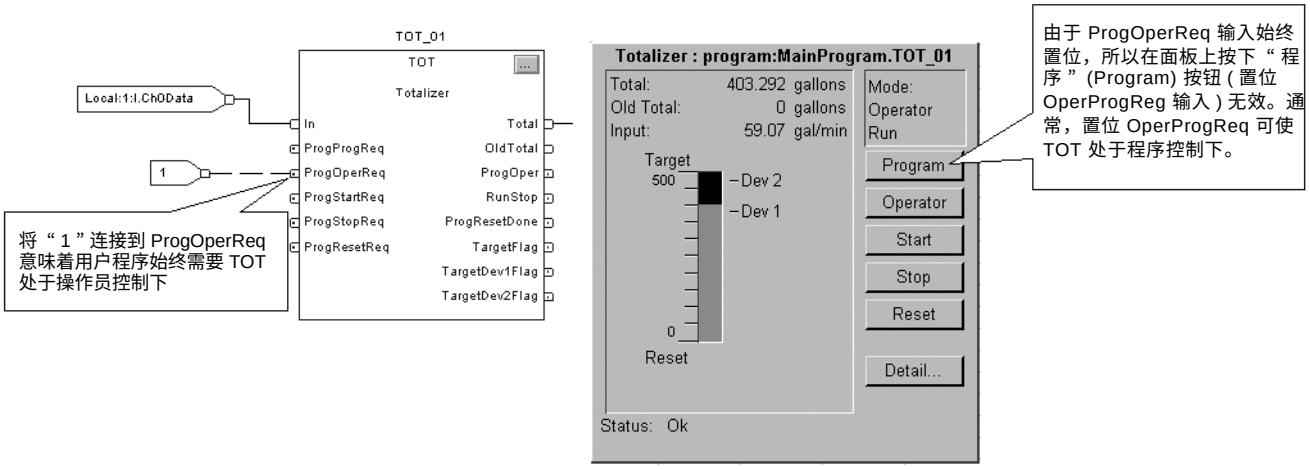
输入	说明
.ProgProgReq	转入程序控制的程序请求。
.ProgOperReq	转入操作员控制的程序请求。
.OperProgReq	转入程序控制的操作员请求。
.OperOperReq	转入操作员控制的操作员请求。

要确定指令是处在程序控制还是操作员控制下，就要检查 ProgOper 输出。如果 ProgOper 被置位，则说明指令处于程序控制下；如果 ProgOper 处于清零状态，则说明指令处于操作员控制下。

当两个输入请求位都被置位时，操作员控制比程序控制优先。例如，如果 ProgProgReq 和 ProgOperReq 都被置位，指令将转入操作员控制。

程序请求输入比操作员请求输入优先。这使得可以使用 ProgProgReq 和 ProgOperReq 输入将指令 “ 锁定 ” 在所需控制下。

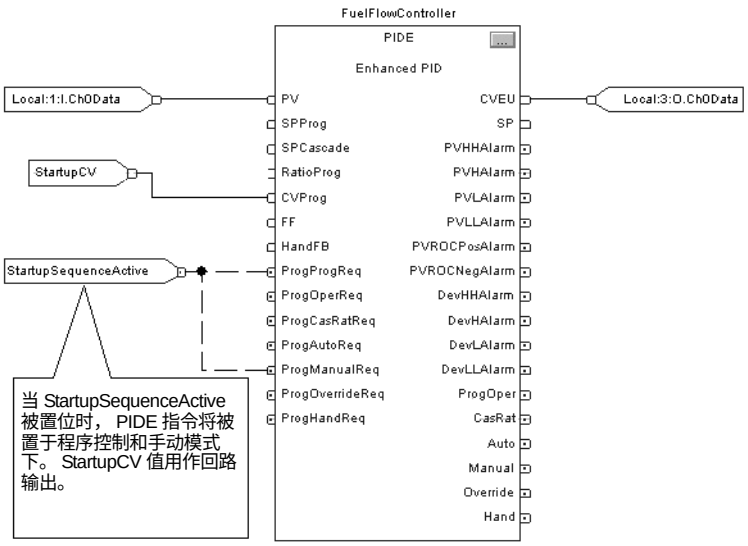
例如，假定一条累加器指令始终处于操作员控制下，并且用户程序始终不能控制该累加计器的运行或停止。这种情况下，用户可以将文字值 1 连接到 ProgOperReq。这就防止了操作员通过操作员界面设备置位 OperProgReq 将累加器置于程序控制下。



同样，始终置位 ProgProgReq 能够将指令 “ 锁定 ” 在程序控制下。当您想让程序控制指令的动作而不想为操作员在不经意间控制指令而担心时，这对于自动启动序列很有用。

在这个示例中，用户在启动时使用程序来置位 ProgProgReq 输入，然后在启动完成后清零 ProgProgReq 输入。一旦 ProgProgReq 输入被清零，指令在接收到变更请求之前，将一直处于程序控制下。例如，操作员可以通过面板置位 ProgOperReq 输入以控制该指令。

以下示例显示如何将指令锁定在程序控制下。



指令的操作员请求输入总是在指令执行时被指令清零。这使得操作员界面可以只通过设置所需的模式请求位来使用这些指令。您不必为了复位请求位而对操作员界面编程。例如，如果操作员界面置位 PIDE 指令的 OperAutoReq 输入，当 PIDE 指令执行时，它将确定适当的响应并清零 OperAutoReq。

程序请求输入一般不由指令清零，因为它们通常作为输入连接到指令。如果指令清零这些输入，这些输入将重新被所连接的输入置位。有可能出现这样的情况，用户想由指令来清零程序请求，从而使用其它逻辑来置位程序请求。在这种情况下，用户可以置位 ProgValueReset 输入，这样指令在执行时始终会清零程序模式请求输入。

在本示例中，另一例程中的梯形图逻辑梯级用于在按下按钮时单步锁存 PIDE 指令的 ProgAutoReq。由于 PIDE 指令可以自动清零程序模式请求，所以用户不必编写任何梯形图逻辑在例程执行后清零 ProgAutoReq，并且每次按下按钮时， PIDE 指令将只接收到一个转到 Auto 的请求。

按下 TIC101AutoReq 按钮时，将单步锁存 PIDE 指令 TIC101 的 ProgAutoReq。 TIC101 已经在 ProgValueReset 输入被置位的情况下组态，所以当 PIDE 指令执行时，将自动清零 ProgAutoReq。

注：

结构化文本编程

简介

本附录介绍结构化文本编程所特有的问题。查阅本附录中的信息以确保您理解结构化文本编程的执行方式。

结构化文本语法

结构化文本是一种使用语句来定义执行内容的文本化编程语言。

- 结构化文本不区分大小写。
- 使用制表符和回车 (单独的行) 可以使结构化文本更容易阅读。它们对结构化文本的执行没有影响。

结构化文本不区分大小写。结构化文本可以包含如下组成部分。

术语	定义		示例
赋值 (请参见 第 669 页)	使用赋值语句将值赋给标签。 运算符 := 为赋值运算符。 赋值语句以分号 “ ; ” 结束。		标签 := 表达式 ;
表达式 (请参见 第 671 页)	表达式是一个完整的赋值或结构语句的一部分。表达式可以求出一个数值 (数值表达式) 或一个真或假的状态 (BOOL 表达式)。 表达式包含：		
	标签	内存中的已命名区域，用来存储数据 (BOOL、SINT、INT、DINT、REAL、字符串)。	value1
	立即值	一个常量值。	4
	运算符	一种用来指定表达式中运算的符号或助记符。	tag1 + tag2 tag1 >= value1
	函数	执行时，函数将生成一个值。函数的操作数需要用圆括号括起来。 虽然函数和指令的语法类似，但它们却是不同的，函数只能用在表达式中。指令不能用在表达式中。	函数 (tag1)
指令 (请参见 第 678 页)	指令是一种独立语句。 指令的操作数需要用圆括号括起来。 不同的指令可能有零个、一个或多个操作数。 运行时，指令将生成一个或多个值，这些值是数据结构的一部分。 指令以分号 (;) 结束。 虽然指令和函数的语法类似，但它们却是不同的，指令不能用在表达式中。函数只能用在表达式中。		指令 (); 指令 (操作数); 指令 (操作数 1, 操作数 2, 操作数 3);

术语	定义	示例
结构 (请参见 第 679 页)	用来触发结构化文本代码 (即其它语句) 的条件语句。 结构以分号 (;) 结束。	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
注释 (请参见 第 695 页)	解释或阐明某段结构化文本的功能的文字。 • 使用注释可以更加容易地解析结构化文本。 • 注释不影响结构化文本的执行。 • 注释可以出现在结构化文本的任何地方。	// 注释 (* 注释开始 .. . 注释结束 *) /* 注释开始 .. . 注释结束 */

赋值

赋值语句可用于更改标签内存储的值。赋值语句的语法如下：

标签 := 表达式 ;

其中：

组成部分	说明	
标签	代表获得新值的标签 标签必须为 BOOL、 SINT、 INT、 DINT 或 REAL 类型	
:=	为赋值符号	
表达式	代表赋给标签的新值	
	如果标签为以下数据类型	使用以下类型表达式
	BOOL	BOOL 表达式
	SINT INT DINT REAL	数值表达式
;	结束赋值	

在其它赋值语句更改所赋的值之前，标签将一直保持该值。

表达式可以很简单 (如立即值或其它标签名)，也可以很复杂，包括多个运算符和 / 或函数。请参见第 671 页的表达式。

指定非保持型赋值

非保持型赋值与上面所描述的常规赋值不同，控制器每次执行以下动作时，非保持型赋值语句中的标签都将复位为零：

- 进入 RUN 模式。
- 离开 SFC 的步 (在 SFC 组态为 *“自动复位”(Automatic reset)* 的情况下)(仅当赋值被嵌入到该步的 Action 中，或使用该 Action 通过 JSR 指令调用结构化文本例程时，这一条才适用。)

非保持型赋值的语法如下：

标签 [:=] 表达式 ;

其中：

组成部分	说明	
标签	代表获得新值的标签 标签必须为 BOOL、SINT、INT、DINT 或 REAL 类型	
[:=]	为非保持型赋值符号	
表达式	代表赋给标签的新值	
	如果标签为以下数据类型	使用以下类型表达式
	BOOL	BOOL 表达式
	SINT INT DINT REAL	数值表达式
;	结束赋值	

将 ASCII 字符赋给字符串

通过赋值运算符可以将 ASCII 字符赋给字符串标签的 DATA 子元素的元素。要将字符赋给字符串，需要指定字符的值或指定标签名称、 DATA 子元素和字符的元素。

下表给出了一些示例。

可以	不可以
`string1.DATA[0]:= 65;`	`string1.DATA[0] := A;`
`string1.DATA[0]:= string2.DATA[0];`	`string1 := string2;`

要将一串字符添加或插入到字符串标签中，请使用以下 ASCII 字符串指令中的一种。

要执行以下操作	使用以下指令
将字符添加到字符串末尾	CONCAT
将字符插入字符串	INSERT

表达式

表达式是一个标签名称、等式或比较关系。要编写一个表达式，可使用以下内容：

- 用来存储值的标签名称 (变量)
- 直接输入到表达式中的数字 (立即值)
- 函数，例如：ABS, TRUNC
- 运算符，例如：+, -,<, >, And, Or

编写表达式时，遵守以下一般规则：

- 表达式中可混合使用大小写字母。例如，可使用这三种形式的 “ AND ”：AND、 And、 and。
- 对于较复杂的要求，可在表达式内使用圆括号组合多个表达式。这样可以提高整个表达式的可读性，并确保表达式按照期望的顺序执行。请参见第 677 页的确定执行顺序。

在结构化文本中，可以使用以下两种类型的表达式。

BOOL 表达式：生成 BOOL 值 1(真) 或 0(假) 的表达式。

- BOOL 表达式使用布尔标签、关系运算符和逻辑运算符来比较值或检查条件是真还是假。例如， tag1>65。
- 一个简单的 BOOL 表达式可以是一个单独的 BOOL 标签。
- 通常，使用 BOOL 表达式来限制其它逻辑的执行。

数值表达式：计算整数值或浮点值的表达式。

- 数值表达式使用算术运算符、算术函数和按位运算符。例如，tag1+5。
- 经常在 BOOL 表达式内嵌套一个数值表达式。例如，(tag1+5)>65。

使用下表为表达式选择运算符。

如果要	则
计算算术值	第 673 页上的使用算术运算符和函数。
比较两个值或字符串	第 674 页上的使用关系运算符。
检查条件是真是假	第 676 页上的使用逻辑运算符。
比较值内各位	第 677 页上的使用按位运算符。

使用算术运算符和函数

可以在算术表达式内使用多个运算符和函数。

算术运算符计算新值。

要执行以下运算	使用以下运算符	最佳数据类型
加	+	DINT、 REAL
减 / 取反	-	DINT、 REAL
乘	*	DINT、 REAL
指数 (x 的 y 次幂)	**	DINT、 REAL
除	/	DINT、 REAL
求模除法	MOD	DINT、 REAL

算术函数执行数学运算。为函数指定一个常量、一个非布尔型标签或一个表达式。

对于	使用以下函数	最佳数据类型
绝对值	ABS (*numeric_expression*)	DINT、 REAL
反余弦	ACOS (*numeric_expression*)	REAL
反正弦	ASIN (*numeric_expression*)	REAL
反正切	ATAN (*numeric_expression*)	REAL
余弦	COS (*numeric_expression*)	REAL
弧度转角度	DEG (*numeric_expression)*	DINT、 REAL
自然对数	LN (*numeric_expression*)	REAL
以 10 为底的对数	LOG (*numeric_expression*)	REAL
角度转弧度	RAD (*numeric_expression*)	DINT、 REAL
正弦	SIN (*numeric_expression*)	REAL
平方根	SQRT (*numeric_expression*)	DINT、 REAL
正切	TAN (*numeric_expression*)	REAL
截断	TRUNC (*numeric_expression*)	DINT、 REAL

下表给出了一些示例。

使用的格式	示例	
	针对的情况	编写
value1 operator value2	如果 gain_4 和 gain_4_adj 均为 DINT 标签并且要求：“ 将 15 与 gain_4 相加，并将结果存储在 gain_4_adj 中 ”	gain_4_adj := gain_4+15;
operator value1	如果 alarm 和 high_alarm 均为 DINT 标签且要求：“ 对 high_alarm 取反，并将结果存储在 alarm 中。”	alarm:= -high_alarm;
function(numeric_expression)	如果 overtravel 和 overtravel_POS 均为 DINT 标签并且要求：“ 计算 overtravel 的绝对值，并将结果存储在 overtravel_POS 中。”	overtravel_POS := ABS(overtravel);
value1 operator (function((value2+value3)/2)	如果 adjustment 和 position 均为 DINT 标签，并且 sensor1 和 sensor2 均为 REAL 标签，而且要求：“ 计算 sensor1 和 sensor2 的平均值，再取这个平均值的绝对值，再将这个绝对值加上 adjustment，并将结果存储在 position 中。”	position := adjustment + ABS((sensor1 + sensor2)/2);

使用关系运算符

关系运算符可以对两个值或字符串进行比较并得出一个或真或假的结果。关系运算的结果是一个 BOOL 值。

如果比较结果为	则结果为
真	1
假	0

使用下列关系运算符。

要进行以下比较	使用以下运算符	最佳数据类型
等于	=	DINT、 REAL、字符串
小于	<	DINT、 REAL、字符串
小于等于	<=	DINT、 REAL、字符串
大于	>	DINT、 REAL、字符串
大于等于	>=	DINT、 REAL、字符串
不等于	<>	DINT、 REAL、字符串

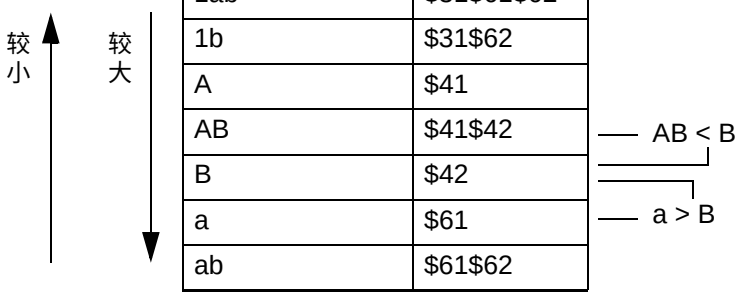
下表给出了一些示例。

使用的格式	示例	
	针对的情况	编写
value1 operator value2	如果 temp 是 DINT 标签且要求：“ 如果 temp 小于 100 · 则 ... ”	IF temp<100 THEN...
stringtag1 operator stringtag2	如果 bar_code 和 dest 均为字符串标签且要求：“ 如果 bar_code 等于 dest，则 ... ”	IF bar_code=dest THEN...
char1 operator char2 要将 ASCII 字符直接输入到表达式中，请输入该字符的十进制值。	如果 bar_code 是字符串标签且要求：“ 如果 bar_code.DATA[0] 等于 “ A ”，则 ... ”	IF bar_code.DATA[0]=65 THEN...
bool_tag := bool_expressions	如果 count 和 length 是 DINT 标签，done 是 BOOL 标签，并且要求：“ 如果 count 大于等于 length，则完成计数。 ”	Done := (count >= length);

如何评估字符串

ASCII 字符对应的十六进制值决定了一个字符串是小于还是大于另一个字符串。

- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。



ASCII 字符	十六进制码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

- 如果字符串的字符匹配，则字符串相等。
- 字符区分大小写。大写的 “ A ” ($41) 不等于小写的 “ a ” ($61)。

使用逻辑运算符

用户可通过逻辑运算符检查多个条件是真还是假。逻辑运算的结果是一个 BOOL 值。

如果比较结果为	则结果为
真	1
假	0

使用下列逻辑运算符。

对于	使用以下运算符	数据类型
逻辑与	&、 AND	BOOL
逻辑或	OR	BOOL
逻辑异或	XOR	BOOL
逻辑取反	NOT	BOOL

下表给出了一些示例。

使用的格式	示例	
	针对的情况	编写
BOOLtag	如果 photoeye 是 BOOL 标签且要求：“ 如果 photoeye 开启，则 ... ”	IF photoeye THEN...
NOT *BOOLtag*	如果 photoeye 是 BOOL 标签且要求：“ 如果 photoeye 关闭，则 ... ”	IF NOT photoeye THEN...
expression1 & *expression2*	如果 photoeye 是 BOOL 标签， temp 是 DINT 标签，且要求：“ 如果 photoeye 开启，且 temp 小于 100．，则 ... ”	IF photoeye & (temp<100) THEN...
expression1 OR *expression2*	如果 photoeye 是 BOOL 标签， temp 是 DINT 标签，且要求：“ 如果 photoeye 开启，或 temp 小于 100．，则 ... ”	IF photoeye OR (temp<100) THEN...
expression1 XOR *expression2*	如果 photoeye1 和 photoeye2 均为 BOOL 标签且要求：“ 如果： • photoeye1 开启而 photoeye 2 关闭 • photoeye1 关闭而 photoeye 2 开启 则 ... ”	IF photoeye1 XOR photoeye2 THEN...
BOOLtag := *expression1* & *expression2*	如果 photoeye1 和 photoeye2 均为 BOOL 标签， open 是 BOOL 标签，且要求：“ 如果 photoeye 和 photoeye2 均开启，则将 open 设置为真。”	Open := photoeye1 & photoeye2;

使用按位运算符

按位运算符在两个值的基础上对值内的位进行操作。

对于	使用以下运算符	最佳数据类型
按位与	&、 AND	DINT
按位或	OR	DINT
按位异或	XOR	DINT
按位取反	NOT	DINT

以下是一个示例。

使用的格式	示例	
	针对的情况	编写
value1 operator value2	如果 input1、 input2 和 result1 是 DINT 标签且要求：“ 计算 input1 和 input2 的按位结果。并将结果存储在 result1 中。”	result1 := input1 AND input2;

确定执行顺序

写入表达式中的运算将按照规定的顺序执行，不一定是从左到右。

- 相同顺序的运算从左到右执行。
- 如果表达式中包含多个运算符或函数，可以将条件括在圆括号 ( ) 中。这样可以确保执行顺序正确并使得表达式更具可读性。

顺序	运算
1.	( )
2.	function (…)
3.	**
4.	– (取反)
5.	NOT
6.	*、 /、 MOD
7.	+、 -(减)
8.	<, <=, >, >=
9.	=, <>
10.	&、 AND
11.	XOR
12.	OR

指令

结构化文本语句也可以是指令。请参见第 15 页上的定位器表，以查看结构化文本中可用的指令列表。结构化文本指令在每次被扫描的时候执行。结构中的结构化文本指令在每次结构的条件为真的时候执行。如果结构的条件为假，就不会对该结构内的语句进行扫描。没有触发执行的梯级条件或者状态转换。

这与使用 EnableIn 触发执行的功能块指令不同。结构化文本指令的执行就像 EnableIn 总是置位一样。

这也和使用梯级输入条件来触发执行的梯形图指令不同。一些梯形图指令只在梯级输入条件从假跳变到真的时候执行。这些是触发执行的梯形图指令。在结构化文本中，指令将在每次被扫描到的时候执行，除非用户预先设定该结构化文本指令的执行条件。

例如，ABL 指令是一个触发执行的梯形图指令。在本示例中，ABL 指令只在扫描到 *tag_xic* 从清零跳变到置位时执行。当 *tag_xic* 停留在置位状态或 *tag_xic* 被清零时，不会执行 ABL 指令。

在结构化文本中，如果用户如下编写本示例：

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

则指令 ABL 将在每次扫描到 *tag_xic* 置位时执行，而不只在 *tag_xic* 从清零状态跳变为置位时执行。

如果要让 ABL 指令只在 *tag_xic* 从清零跳变为置位时执行，则必须对该结构化文本指令的执行加以条件限制。使用
单脉冲触发来触发执行。

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     ABL(0,serial_control);
END_IF;
```

结构

结构可以单独编程，或嵌套进其它结构内。

如果要	使用以下结构	在以下语言中可用	页码
如果满足特定的条件，则执行某些动作	IF...THEN	结构化文本	680
基于数值选择要执行的操作	CASE...OF	结构化文本	683
在执行其它操作前执行某项操作指定的次数	FOR...DO	结构化文本	686
只要某些条件为真就重复执行某项操作	WHILE...DO	结构化文本	689
重复执行某项操作，直到条件为真	REPEAT...UNTIL	结构化文本	692

一些关键字保留

下面的结构不可用：

- GOTO
- REPEAT

RSLogix 5000 软件不允许使用它们作为标签名称或结构名称。

IF...THEN

使用 IF…THEN 可在满足特定条件时执行某项操作。

操作数：

```
IF bool_expression THEN
    <statement>;
END_IF;
```

结构化文本

操作数	类型	格式	输入
布尔 _ 表达式 (Bool_expression)	BOOL	标签 表达式	赋值为 BOOL 值 (BOOL 表达式) 的 BOOL 标签或表达式

说明： 语法在表中进行说明。

```
IF bool_expression1 THEN
      <statement >;          ◄──  当 bool_expression1 为真时要执行的语句
      .
      .
      .
可选 ┐ ELSIF bool_expression2 THEN
      <statement>;           ◄──  当 bool_expression2 为真时要执行的语句
      .
      .
      .
可选 ┐ ELSE
      <statement>;           ◄──  当两个表达式都为假时要执行的语句
      .
      .
      .
END_IF;
```

根据以下原则使用 ELSIF 或 ELSE。

1. 要从多个可能的语句组中选择，添加一个或多个 ELSIF 语句。
   - 每个 ELSIF 表示一个候选路径。
   - 根据需要指定多个 ELSIF 路径。
   - 控制器执行首个为真的 IF 或 ELSIF，并跳过其余的 ELSIF 和 ELSE。
2. 要在所有 IF 或 ELSIF 条件为假时执行某项操作，则添加一条 ELSE 语句。

下表汇总了 IF、 THEN、 ELSIF 和 ELSE 语句的不同组合。

如果要	并且	使用以下结构
当条件为真时，执行某项操作	如果条件为假，则不执行操作	IF…THEN
	如果条件为假，则执行其它操作	IF…THEN…ELSE
根据输入条件，从多个候选语句 (或语句组) 中选择	如果条件为假，则不执行操作	IF…THEN…ELSIF
	如果所有条件为假，则分配默认语句	IF…THEN…ELSIF…ELSE

算术状态标志　不受影响

故障条件：　无

示例 **1**：　**IF…THEN**

如果要实现以下要求	输入以下结构化文本
如果废品数 > 3，则	`IF rejects > 3 THEN`
传送带 = 关闭 (0)	`conveyor := 0;`
报警器 = 开启 (1)	`alarm:= 1;`
	`END_IF;`

示例 **2**：　**IF…THEN…ELSE**

如果要实现以下要求	输入以下结构化文本
如果传送带方向触点 = 正向 (1)，则	`IF conveyor_direction THEN`
指示灯 = 灭	`light := 0;`
否则，指示灯 = 亮	`ELSE`
	`light [:=] 1;`
	`END_IF;`

每当控制器发生以下情况时， [:=] 都会指示控制器将 *light* 清零：

- 进入 RUN 模式。
- 离开 SFC 的步 (在 SFC 组态为 *“自动复位”(Automatic reset)* 的情况下)(仅当赋值被嵌入到该步的 Action 中，或使用该 Action 通过 JSR 指令调用结构化文本例程时，这一条才适用。)

示例 3： IF…THEN…ELSIF

如果要实现以下要求	输入以下结构化文本
如果糖料低限位开关 = 低 (接通) 并且糖料高限位开关 = 不高 (接通)，则	`IF Sugar.Low & Sugar.High THEN`
进给阀 = 打开 (导通)	`Sugar.Inlet [:=] 1;`
直至糖料高限位开关 = 高 (断开)	`ELSIF NOT(Sugar.High) THEN`
	`Sugar.Inlet := 0;`
	`END_IF;`

每当控制器发生以下情况时，[:=] 都会指示控制器将 *Sugar.Inlet* 清零：

- 进入 RUN 模式。
- 离开 SFC 的步 (在 SFC 组态为 *“自动复位”(Automatic reset)* 的情况下)(仅当赋值被嵌入到该步的 Action 中，或使用该 Action 通过 JSR 指令调用结构化文本例程时，这一条才适用。)

示例 4： IF…THEN…ELSIF…ELSE

如果要实现以下要求	输入以下结构化文本
如果罐温度 > 100	`IF tank.temp > 200 THEN`
则泵 = 缓慢运转	`pump.fast :=1; pump.slow :=0; pump.off :=0;`
如果罐温度 > 200	`ELSIF tank.temp > 100 THEN`
则泵 = 快速运转	`pump.fast :=0; pump.slow :=1; pump.off :=0;`
否则泵 = 关闭	`ELSE`
	`pump.fast :=0; pump.slow :=0; pump.off :=1;`
	`END_IF;`

CASE...OF

使用 CASE 可基于数值选择要执行的操作。

操作数：

```
CASE numeric_expression OF
     selector1:statement;
     selectorN:statement;
ELSE
     statement;
END_CASE;
```

结构化文本

操作数	类型	格式	输入
数值 _ 表达式 (Numeric_expression)	SINT INT DINT REAL	标签 表达式	赋值为数值 (数值表达式) 的标签或表达式
选择器 (Selector)	SINT INT DINT REAL	立即数	与 numeric_expression 的类型相同

重要事项　REAL 值极有可能位于一定的值范围内，而不是与某个特定值精确匹配，因此如果使用 REAL 值，就必须为选择器设定一个值范围。

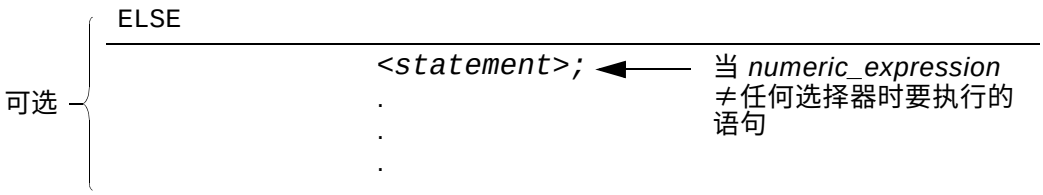
说明：

语法在表中进行说明。

CASE *numeric_expression* OF

根据需要指定多个候选的选择器值 (路径)。

- *selector1* : <*statement*>; ← 当 *numeric_expression* = *selector1* 时要执行的语句
  .
  .
  .
- *selector2* : <*statement*>; ← 当 *numeric_expression* = *selector2* 时要执行的语句
  .
  .
  .
- *selector3* : <*statement*>; ← 当 *numeric_expression* = *selector3* 时要执行的语句
  .
  .
  .



有关有效的选择器值，请参见第 685 页上的表。

下面是输入选择器值时所用的语法。

当选择器为	输入
一个值	*value*:*statement*
多个不同的值	*value1*, *value2*, *valueN* :<*statement*> 使用逗号 (,) 将各个值隔开。
值范围	*value1*..*valueN* :<*statement*> 使用两个句点 (..) 来确定范围。
不同的值加上值范围	*valuea, valueb, value1*..*valueN* :<*statement*>

CASE 结构类似于 C 或 C++ 编程语言中的 switch 语句。但对于 CASE 结构，控制器只执行与最先与选择器值匹配所关联的语句。执行总是在选择器的语句后终止，并转到 END_CASE 语句。

算术状态标志： 不受影响

故障条件： 无

示例

如果要实现以下要求	输入以下结构化文本
如果配方号 = 1，则	CASE recipe_number OF
成分 A 出口 1 = 打开 (1) 成分 B 出口 4 = 打开 (1)	1: Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=1;
如果配方号 = 2 或 3，则 成分 A 出口 4 = 打开 (1) 成分 B 出口 2 = 打开 (1)	2,3: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
如果配方号 = 4、 5、 6 或 7，则 成分 A 出口 4 = 打开 (1) 成分 B 出口 2 = 打开 (1)	4…7: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
如果配方号 = 8、 11、 12 或 13，则 成分 A 出口 1 = 打开 (1) 成分 B 出口 4 = 打开 (1)	8,11…13 Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=1;
否则所有的出口 = 关闭 (0)	ELSE
	Ingredient_A.Outlet_1 [:=]0; Ingredient_A.Outlet_4 [:=]0; Ingredient_B.Outlet_2 [:=]0; Ingredient_B.Outlet_4 [:=]0;
	END_CASE;

每当控制器发生以下情况时， [:=] 都会指示控制器也清零出口标签：

- 进入 RUN 模式。
- 离开 SFC 的步 (在 SFC 组态为 *“自动复位 ”(Automatic reset)* 的情况下)(仅当赋值被嵌入到该步的 Action 中，或使用该 Action 通过 JSR 指令调用结构化文本例程时，这一条才适用。)

FOR···DO

使用 FOR···DO 循环，可在执行其他操作前执行某项操作特定次数。

操作数：

```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

结构化文本

操作数	类型	格式	说明
计数 (count)	SINT INT DINT	标签	FOR···DO 执行时，存储计数位置的标签
初始值 (initial_ value)	SINT INT DINT	标签 表达式 立即数	必须赋值为一个数 指定计数的初始值
最终值 (final_ value)	SINT INT DINT	标签 表达式 立即数	指定计数的最终值，该值确定何时退出循环
增量 (increment)	SINT INT DINT	标签 表达式 立即数	(*可选*) 循环一次时计数值的增量 如果不指定增量，则计数值的增量为 1。

重要事项

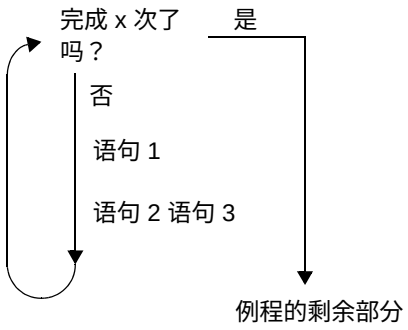
请确保在一个扫描周期内**不要**在循环内迭代过多的次数。

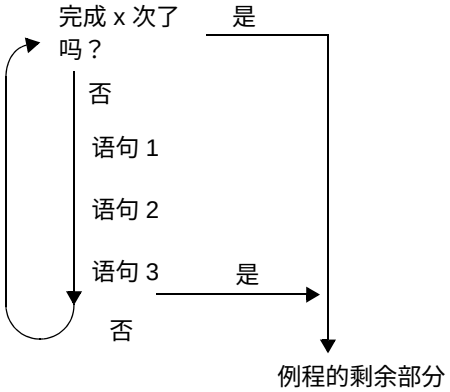
- 控制器在完成循环前不会执行例程中的其它任何语句。
- 如果完成循环所用的时间超过了任务的看门狗计时器的值，那么将出现主要故障。
- 请考虑使用其它结构，如 IF...THEN。

说明： 语法在表中进行说明。

FOR *count* := *initial_value*		
	TO *final_value*	
可选 {	BY *increment*	如果不指定增量，循环将采用 1 作为增量。
	DO	
	<*statement*>;	
可选 {	IF *bool_expression* THEN	
	EXIT;	如果要提前退出循环，请使用其它语句 (如 IF...THEN 结构) 来限制 EXIT 语句。
	END_IF;	
END_FOR;		

下面的图显示出 FOR...DO 循环如何执行以及 EXIT 语句如何提前退出循环。





FOR⋯DO 循环执行特定次数。

要在计数达到最终值前终止循环，请使用 EXIT 语句。

算术状态标志： 不受影响

故障条件：

出现严重故障的条件	故障类型	故障代码
结构循环过长。	6	1

示例 1：

如果要实现以下要求	输入以下结构化文本

清除 BOOL 数组的位 0...31：
1. 将 *subscript* 标签初始化为 0。
2. 清除 *array[subscript]*。例如，当 *subscript* = 5 时，清除 *array[5]*。
3. 将 *subscript* 加 1。
4. 如果 *subscript* ≤ 31，则重复 2 和 3。
否则停止。
	`For subscript:=0 to 31 by 1 do`
	`  array[subscript] := 0;`
	`End_for;`

示例 2：

如果要实现以下要求	输入以下结构化文本

通过用户自定义的数据类型 (结构) 存储库存中货物的下列信息：
• 货物的条形码 ID(字符串数据类型)
• 货物的库存数量 (DINT 数据类型)
如上结构的数组包含了库存中各个不同货物的元素。用户想要搜索特定产品的数组 (使用条形码)，并确定库存数量。
1. 获取 Inventory 数组的尺寸 (货物数)，并将结果存储到 Inventory_Items(DINT 标签) 中。
2. 将 position 标签初始化为 0。
3. 如果 Barcode 与数组中某条目的 ID 匹配，那么：
a. 设置 Quantity 标签 = Inventory[position].Qty。这将生成该货物的库存量。
b. 停止。
Barcode 是字符串标签，用于存储所搜索货物的条形码。例如，当 position = 5 时，将 Barcode 与 Inventory[5].ID 进行比较。
4. 将 position 加 1。
5. 如果 position ≤ (Inventory_Items -1)，则重复 3 和 4。由于元素编号起始于 0，所以最后一个元素的编号比数组的元素数小 1。
否则停止。
	`SIZE(Inventory,0,Inventory_Items);`
	`For position:=0 to Inventory_Items - 1 do`
	`  If Barcode = Inventory[position].ID then`
	`  Quantity := Inventory[position].Qty;`
	`  Exit;`
	`  End_if;`
	`End_for;`

WHILE···DO

使用 WHILE···DO 循环可实现以下要求：只要某些条件为真，就重复执行某项操作。

操作数：

结构化文本

```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

操作数	类型	格式	输入
布尔_表达式 (Bool_expression)	BOOL	标签 表达式	赋值为 BOOL 值的 BOOL 标签或表达式

重要事项

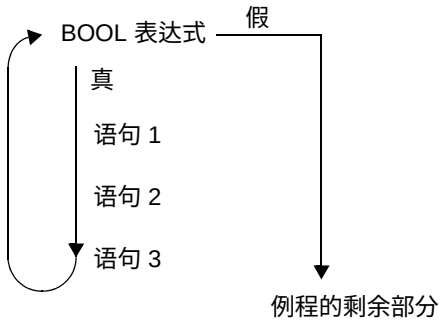
请确保在一个扫描周期内不要在循环内迭代过多的次数。

- 控制器在完成循环前不会执行例程中的其它任何语句。
- 如果完成循环所用的时间超过了任务的看门狗计时器的值，那么将出现主要故障。
- 请考虑使用其它结构，如 IF...THEN。

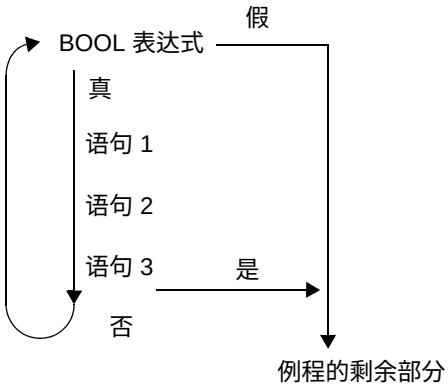
说明： 语法在表中进行说明。

```
WHILE bool_expression1 DO
    <statement>;                    ◄── 当 bool_expression1 为真时要执行的语句
可选 ┌ IF bool_expression2 THEN
     │ EXIT;                        ◄── 如果要提前退出循环，请使用其它语句 ( 如 IF...THEN 结构 ) 来限制 EXIT 语句。
     └ END_IF;
END_WHILE;
```

下面的图显示出 WHILE...DO 循环如何执行以及 EXIT 语句如何提前退出循环。



当 ***bool_expression*** 为真时，控制器仅执行 **WHILE**…**DO** 循环内的语句。



要在条件为真前终止循环，请使用 **EXIT** 语句。

算术状态标志： 不受影响

故障条件：

出现严重故障的条件	故障类型	故障代码
结构循环过长	6	1

示例 **1**：

如果要实现以下要求	输入以下结构化文本

WHILE...DO 循环首先评估其条件。如果条件为真，则控制器执行循环内的语句。
 	
该循环不同于 REPEAT...UNTIL 循环，因为 REPEAT...UNTIL 循环首先执行结构内的语句，然后在再次执行语句前确定条件是否为真。REPEAT...UNTIL 循环中的语句至少执行一次。而 WHILE...DO 循环中的语句可能一次都不执行。
	pos := 0;
	While ((pos <= 100) & structarray[pos].value <> targetvalue)) do
	  pos := pos + 2;
	  String_tag.DATA[pos] := SINT_array[pos];
	end_while;

示例 2：

如果要实现以下要求	输入以下结构化文本

将 SINT 数组中的 ASCII 字符移入字符串标签。(在 SINT 数组中，每个元素保存一个字符。) 到达回车时停止。
 	
1. 将 Element_number 初始化为 0。
2. 计算 SINT_array(含有 ASCII 字符的数组) 中的元素个数，并将结果存入 SINT_array_size(DINT 标签) 中。
3. 如果 SINT_array[element_number] 中的字符 = 13(回车的十进制值)，则停止。
4. 设置 String_tag[element_number] = SINT_array[element_number] 中的字符。
5. 将 element_number 加 1。这使得控制器检查 SINT_array 中的下一个字符。
6. 设置 String_tag 的长度元素 = element_number。(它记录当前 String_tag 中的字符数。)
7. 如果 element_number = SINT_array_size，则停止。 (已执行到数组末尾但无回车。)
8. 转到 3。
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	While SINT_array[element_number] <> 13 do
	　　String_tag.DATA[element_number] := SINT_array[element_number];
	　　element_number := element_number + 1;
	　　String_tag.LEN := element_number;
	　　If element_number = SINT_array_size then
	　　　　exit;
	　　end_if;
	end_while;

REPEAT...UNTIL

使用 REPEAT...UNTIL 循环可重复执行某项操作，直到条件为真结束。

操作数：

```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

结构化文本

操作数	类型	格式	输入
布尔 _ 表达式 (Bool_expression)	BOOL	标签 表达式	赋值为 BOOL 值 (BOOL 表达式) 的 BOOL 标签或表达式

重要事项

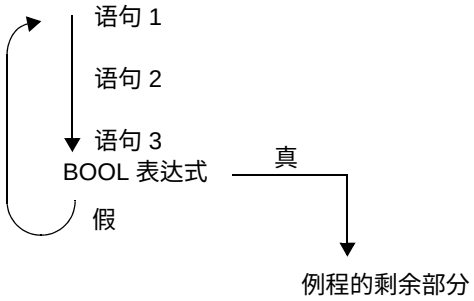
请确保在一个扫描周期内**不要**在循环内迭代过多的次数。

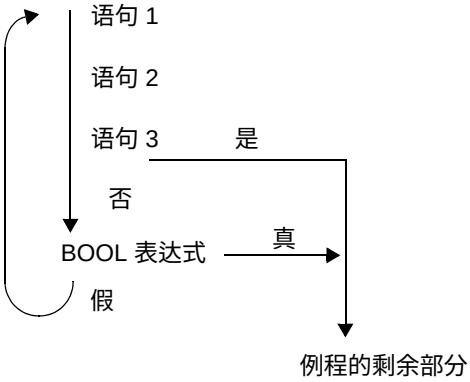
- 控制器在完成循环前不会执行例程中的其它任何语句。
- 如果完成循环所用的时间超过了任务的看门狗计时器的值，那么将出现主要故障。
- 请考虑使用其它结构，如 IF...THEN。

说明： 语法在表中进行说明。

	REPEAT		
	<*statement*>;	←	当 *bool_expression1* 为假时要执行的语句
可选	IF *bool_expression2* THEN		
可选	EXIT;	←	如果要提前退出循环，请使用其它语句 (如 IF...THEN 结构) 来限制 EXIT 语句。
可选	END_IF;		
	UNTIL *bool_expression1*		
	END_REPEAT;		

下面的图显示出 REPEAT...UNTIL 循环如何执行以及 EXIT 语句如何提前退出循环。





当 ***bool_expression*** 为假时，控制器仅执行 **REPEAT...UNTIL** 循环内的语句。

要在条件为假前终止循环，请使用 **EXIT** 语句。

算术状态标志： 不受影响

故障条件：

出现严重故障的条件	故障类型	故障代码
结构循环过长	6	1

示例 **1**：

如果要实现以下要求	输入以下结构化文本

REPEAT...UNTIL 循环首先执行结构内的语句，然后在再次执行语句前确定条件是否为真。
 	
该循环不同于 WHILE...DO 循环，因为 WHILE...DO 循环首先评估其条件。如果条件为真，则控制器执行循环内的语句。
REPEAT...UNTIL 循环中的语句至少执行一次。而 WHILE...DO 循环中的语句可能一次都不执行。
	pos := -1;
	REPEAT
	　　pos := pos + 2;
	UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))
	end_repeat;

示例 2：

如果要实现以下要求	输入以下结构化文本

将 SINT 数组中的 ASCII 字符移入字符串标签。(在 SINT 数组中，每个元素保存一个字符。) 到达回车时停止。
 	
1. 将 Element_number 初始化为 0。
2. 计算 SINT_array(含有 ASCII 字符的数组) 中的元素个数，并将结果存入 SINT_array_size(DINT 标签) 中。
3. 设置 String_tag[element_number] = SINT_array[element_number] 中的字符。
4. 将 element_number 加 1。这使得控制器检查 SINT_array 中的下一个字符。
5. 设置 String_tag 的长度元素 = element_number。(它记录当前 String_tag 中的字符数。)
6. 如果 element_number = SINT_array_size，则停止。 (已执行到数组末尾但无回车。)
7. 如果 SINT_array[element_number] 中的字符 = 13(回车的十进制值)，则停止。
 	
否则，转到步骤 3。
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	Repeat
	  String_tag.DATA[element_number] := SINT_array[element_number];
	  element_number := element_number + 1;
	  String_tag.LEN := element_number;
	  If element_number = SINT_array_size then
	  exit;
	  end_if;
	Until SINT_array[element_number] = 13
	end_repeat;

注释

为使结构化文本更易于解读，需添加注释。

- 通过注释，可以使用清晰的语言来描述结构化文本的工作方式。
- 注释不影响结构化文本的执行。

下表说明了如何向结构化文本添加注释。

要添加注释	使用以下格式之一
在单独一行上	// *注释*
在结构化文本某一行的末尾	(* *注释* *) /* *注释* */
在结构化文本某一行当中	(* *注释* *) /* *注释* */
跨越多行	(* *注释开始 . . . 注释结束* *) /* *注释开始 . . . 注释结束* */

下表给出了一些示例。

格式	示例

//comment	在行的开头
	//Check conveyor belt direction
	
	IF conveyor_direction THEN...
	
	在行的末尾
	ELSE //If conveyor isn't moving, set alarm light
	
	light := 1;
	
	END_IF;

(**comment**)	Sugar.Inlet[:=]1;(*open the inlet*) IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN... (*Controls the speed of the recirculation pump.The speed depends on the temperature in the tank.*) IF tank.temp > 200 THEN...
/**comment*/	Sugar.Inlet:=0;/*close the inlet*/ IF bar_code=65 /*A*/ THEN... /*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/ SIZE(Inventory,0,Inventory_Items);

罗克韦尔自动化公司的技术支持

罗克韦尔自动化公司在网络上提供技术信息以协助您使用其产品。访问 http://www.rockwellautomation.com/support/，您可找到技术手册、常见问题解答知识库、技术与应用说明、示例代码与软件服务包链接以及 MySupport 功能，且您可定制该功能以充分利用这些工具。

我们还提供了 TechConnect 支持项目作为额外的电话技术支持，帮助用户进行安装、组态和故障处理工作。更多相关信息，请联系您当地的分销商或罗克韦尔自动化代表，或者访问 http://www.rockwellautomation.com/support/。

安装协助

如果您在安装后的最初 24 小时内遇到问题，请查阅本手册中包含的信息。
您可联系客户支持来获取首次帮助，以协助您安装好产品并完成试运行。

美国或加拿大	1.440.646.3434
美国或加拿大以外地区	请使用 http://www.rockwellautomation.com/support/americas/phone_en.html 中的全球定位器，或者联系您当地的罗克韦尔自动化代表。

新产品返厂修复

在所有产品出厂前，罗克韦尔自动化公司都会执行测试，以确保产品完全可用。但是，如果您的产品无法正常工作，需要进行退货，请遵守如下程序。

美国	联系分销商。您必须向您的分销商提供客户支持案例号码 (可拨打以上电话号码获取)，才能完成退货流程。
美国境外	请联系您当地的罗克韦尔自动化代表，了解退货程序。

文档反馈

您的意见将帮助我们更好地满足您的文档需求。如果您对于如何改进本文档有任何建议，请填写以下表格：出版号 RA-DU002，网址为 http://www.rockwellautomation.com/literature/。

www.rockwellautomation.com

动力、控制与信息解决方案总部
美洲地区：罗克韦尔自动化，南二大街1201号，密尔沃基市，WI 53204-2496 美国，电话: (1) 414.382.2000，传真: (1) 414.382.4444
欧洲/中东/非洲：罗克韦尔自动化，Vorstlaan/Boulevard du Souverain 36, 1170布鲁塞尔，比利时，电话: (32) 2 663 0600, 传真: (32) 2 663 0640
亚洲地区：罗克韦尔自动化，香港数码港道100号数码港3座F区14楼, 电话: (852) 2887 4788, 传真: (852) 2508 1846
中国总部：上海市漕河泾开发区虹梅路1801号B区宏业大厦1层，邮编：200233，电话: (86 21) 6128 8888, 传真: (86 21) 6128 8899

Publication 1756-RM006M-ZH-P - July 2011